Analyse statique pour l'optimisation des mises à jour de documents XML temporels

Mohamed-Amine Baazizi

UNIVERSITÉ PARIS SUD

ECOLE DOCTORALE D'INFORMATIQUE

THÈSE

présentée pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS SUD

Spécialité : INFORMATIQUE

Soutenue le 7 Septembre 2012

Par

MOHAMED-AMINE BAAZIZI

Analyse statique pour l'optimisation des mises à jour de documents XML temporels

devant le jury composé de :

Nicole Bidoit-Tollu	Pr, Université Paris Sud	Directeur
Bogdan Cautis	MdC HdR, Télécom ParisTech	Examinateur
Dario Colazzo	MdC HdR, Université Paris Sud	Directeur
Stéphane Gançarski	MdC HdR, Université Pierre et Marie Curie	Rapporteur
Mírian Halfeld Ferrari Alves	Pr, Université d'Orléans	Rapporteur
Sophie Laplante	Pr, Université Paris Sud	Examinatrice

Laboratoire de Recherche en Informatique (U.M.R. CNRS 8623) , INRIA Saclay-île-de-France,
Université Paris-Sud, 91405 Orsay Cedex, France

Table des matières

Chapitre 1

Introduction

L'essor du web a révolutionné la façon d'accéder et de traiter les données à de multiples niveaux. Nombreuses sont les applications qui s'appuient sur le web pour l'échange et le partage de données, tant le nombre d'utilisateurs et la complexité de leurs besoins ne cessent de croître. Les données semi-structurées et en particulier le standard *XML* introduit par le W3C, ont contribué dans une large mesure à cette évolution en offrant un format de données compréhensible par un large panel d'utilisateurs et fournissant une panoplie de mécanismes facilitant l'accès et l'échange de données.

Le développement du standard XML s'est accompagné de la mise en place d'une famille de langages tels que XPath [XPa], XQuery [XQu] et XSLT [XSL] permettant de naviguer, d'interroger et de transformer des documents XML. La communauté scientifique a mené de nombreux travaux autour de ces langages pour en étudier l'expressivité, la complexité, pour proposer des techniques d'optimisation dans des contextes différents : intégration de données, sécurité, ...

Le stockage et l'interrogation des données XML sont des aspects importants qui ont suscité le développement de plusieurs systèmes de gestion de données XML. Les premiers systèmes développés reposent sur un stockage en mémoire secondaire et bénéficient des techniques d'optimisation développées pour les SGBD relationnels. L'exemple le plus connu de tels systèmes est sans doute Monet-db [BGvK+06]. L'avantage de ces systèmes est qu'ils permettent un chargement sélectif des données en fonction des besoins des requêtes à évaluer ce qui leur permet de manipuler des documents quasiment sans limite de taille. Toutefois, ces systèmes nécessitent un effort considérable d'administration et de déploiement. Leur utilisation reste réservée à un cadre spécifique tel que l'analyse et le traitement intensifs de données XML.

La plupart des applications manipulant les données XML ne nécessitent pas les fonctionnalités avancées des systèmes précédemment cités. Afin de répondre aux besoins de ces applications, plusieurs systèmes légers connus sous le nom de moteurs mémoire-centrale ont été développés. Ces moteurs sont compatibles avec les principaux standards XML tels que XQuery et XQuery update et permettent le traitement des documents XML directement en mémoire centrale. Intuitivement, ces systèmes sont limités quant à la taille des documents pouvant être traités.

Plusieurs techniques d'optimisation ont été proposées pour faire face aux limitations des moteurs mémoire-centrale [MS03, BCL+05, BCCN06]. Ces techniques ciblent essentiellement les requêtes XML et ne traitent pas les mises à jour. Or, l'utilisation des moteurs mémoire-centrale pour mettre à jour les documents XML se heurte au même problème de limitation en terme de mémoire

posé pour le cas des requêtes.

Les schémas peuvent être utilisés pour spécifier différentes sortes de contraintes sur les documents XML. Dans le cas de XML, les schémas jouent un rôle important dans l'optimisation des requêtes, l'intégration de données et dans le développement d'applications sûres. Il existe plusieurs langages de schémas tels que DTD [DTD] et XSD [XSD] qui diffèrent essentiellement par leur expressivité.

L'avènement de XML a ravivé l'intérêt pour la gestion de l'aspect temporel tel que peut en témoigner l'état de l'art maintenu par [Gra04] sur le sujet. Dans le cas des données XML, la gestion de la dimension temporelle couvre un spectre plus large et s'intéresse à des problématiques issues de domaines tels que la gestion des versions ou l'archivage des données du web.

Les travaux sur XML temporel s'accordent sur l'utilisation de l'estampillage comme moyen d'incorporer la dimension temporelle dans les documents XML. Cette idée, utilisée précédemment dans le cadre relationnel, est particulièrement intéressante pour le stockage des données temporelles. Elle consiste à associer aux données la période de temps durant laquelle elles sont valides et permet ainsi de ne stocker les données que si elles changent dans le temps. Bien que plusieurs travaux s'intéressent à l'interrogation temporelle, peu d'entre eux attachent une importance à la construction et à la maintenance des documents estampillés qui sont pourtant indispensables au stockage des données temporelles. La construction et la maintenance de ces documents est une problématique importante pour le cas de XML et ce pour deux raisons. La première concerne le fait que, très souvent, les données sont manipulées par des moteurs mémoire-centrale et sont donc sujettes à des limitations en terme de mémoire centrale. L'objectif ici est de manipuler des documents volumineux en utilisant ces moteurs. La deuxième raison est liée au fait qu'il peut y avoir différents documents temporels équivalents, certains documents pouvant être plus efficaces en terme de stockage (compacts) que d'autres. Le but est bien évidemment de générer des documents les plus compacts possibles.

La contribution de cette thèse s'articule en deux parties. La première partie s'intéresse à la mise à jour des documents volumineux sous des contraintes de mémoire. La seconde partie s'intéresse à la construction et à la maintenance de documents XML temporels en considérant comme pour la première partie des restrictions en terme de mémoire.

1.1 La projection pour l'optimisation des mises à jour XML

La *projection XML* est une technique d'optimisation proposée dans le but de surmonter les limitations des moteurs mémoire-centrale pour l'interrogation des documents XML. Cette technique repose sur une observation simple selon laquelle les requêtes sont en général sélectives càd qu'elles ciblent seulement une sous-partie des documents interrogés. L'idée consiste alors à identifier de manière statique les parties nécessaires à l'évaluation des requêtes et à utiliser cette information pour ne charger en mémoire centrale que les parties du document qui sont accédées par la requête. La projection permet ainsi de traiter des documents volumineux même sous des contraintes de mémoire importantes.

La projection a été utilisée pour la première fois dans [MS03] puis étendue dans [BCCN06] en prenant en compte le schéma du document interrogé. L'utilisation des schémas permet de réduire

la taille de la projection en exploitant la possibilité d'inférer de manière précise les données nécessaires à l'évaluation d'une requête. Dans la technique de [BCCN06], l'information inférée consiste en l'ensemble des étiquettes des noeuds nécessaires à l'évaluation des requêtes. Cet ensemble est appelé *type-porjecteur*.

Bien qu'étudiée de manière approfondie dans le cadre des requêtes XML [MS03, BCL$^+$05, BCCN06], la projection n'a jamais été considérée dans le cadre des mises à jour XML. D'autre part, les techniques développées pour les requêtes ne peuvent pas être réutilisées de manière directe pour les mises à jour.

La technique que nous proposons pour les mises à jour s'appuie sur l'utilisation des schémas pour inférer des projecteurs pour les mises à jour. Le scénario de notre technique introduit une étape supplémentaire permettant de propager les effets des mises à jour sur le document original. Le projecteur inféré pour les mises à jour est conçu dans un souci de précision (réduire le plus possible la taille des documents projetés) et d'efficacité (permettre d'exécuter rapidement la projection d'un document). Outre le fait de permettre le traitement de documents volumineux, cette technique possède deux autres avantages, celui de ne pas avoir à réécrire les mises à jour et celui de pouvoir utiliser les moteurs mémoire-centrale tels quels, sans aucune modification.

Les expérimentations réalisées pour cette technique montrent que la projection permet non seulement le traitement de documents de taille importante mais conduit aussi à réduire le temps d'exécution des mises à jour.

1.2 Gestion des documents XML temporels

L'objectif du travail réalisé dans cette partie est de proposer deux techniques permettant de construire et de maintenir des documents XML temporels. L'objectif est d'abord de traiter des documents volumineux puisque les documents temporels ont une taille plus importante que la taille les documents statiques. Le deuxième objectif est de générer des documents temporels compacts.

La première technique s'applique dans le cas général pour lequel aucune information n'est utilisée pour construire le document temporel. Cette technique est conçue de sorte à pouvoir être réalisée en streaming en utilisant le parcours SAX [saxa]. Ceci lui permet de traiter des documents volumineux mais l'empêche, dans certains cas, de générer des documents aussi compacts que possible. Ceci s'explique parfaitement par le fait que cette technique n'utilise aucune information sur le changement des données et qu'elle procède en streaming.

La deuxième technique s'applique dans un cas particulier où les changements sont spécifiés par des mises à jour. Dans ce cas les mises à jour sont utilisées pour la construction des documents temporels en utilisant le paradigme de projection. Ceci lui permet de manipuler des documents volumineux et de générer des documents compacts de surcroît. L'avantage de cette technique est qu'elle ne nécessite pas de modifier les moteurs XML pour mettre à jour les documents temporels ni ne requiert de réécrire les mises à jour en des mises à jour temporelles.

La comparaison des documents temporels relativement à leur efficacité de stockage est étudiée. Les expérimentations modestes menées sur les deux techniques permettent de confirmer le fait que sous certaines hypothèses, la deuxième technique produit des documents plus compacts que le première.

1.3 Organisation du manuscrit

Ce manuscrit est composé de cinq chapitres dont un chapitre préliminaire. Ce chapitre est consacré à la présentation des notations et des langages de requêtes et de mises à jour utilisés tout au long de ce manuscrit. Les quatre autres chapitres sont organisés en deux parties :

Première partie : Optimisation des mises à jour par projection
Cette partie regroupe deux chapitres :
Le chapitre 3 présente les différents techniques de projection pour les requêtes développées dans la littérature.
Le chapitre 4 introduit la technique de projection pour les mises à jour que nous proposons.
Les preuves détaillées de cette partie sont données dans les annexes de ce document.

Deuxième partie : Mises à jour de documents XML temporels
Cette partie est organisée en deux chapitres :
Le chapitre 5 est consacré à la présentation des principales approches de la gestion du temps pour XML.
Le chapitre 6 présente la contribution dans ce domaine.

Chapitre 2

Préliminaires

Ce chapitre a pour vocation d'introduire et de formaliser le modèle de données XML et les langages d'interrogation et de mise à jour pour XML.

2.1 XML : Données et schémas

XML, pour *eXtensible Markup Language*, est un format textuel conçu par le W3C [XML] pour structurer les données sous forme arborescente. Inspiré de SGML, il se base sur l'utilisation de balises pour encoder la structure arborescente des données modélisées. Son objectif principal est de permettre de stocker et de transférer des données sur le web sans se soucier de la plate-forme où elles vont être utilisées.

Sous un angle différent, XML est une syntaxe basée sur des balises permettant de décrire un arbre ayant un nombre variable de noeuds (appelés aussi *éléments*) étiquetés et pour lequel l'ordre entre les fils d'un noeud peut être important. Un arbre XML possède une racine unique qui contient tous les autres éléments. Chaque élément peut contenir soit une séquence (éventuellement vide) d'éléments pouvant à leur tour contenir d'autres éléments, soit une chaîne de caractères qu'on appelle aussi *noeud texte* qui joue le même rôle que les enregistrements dans le modèle relationnel.

Un exemple [1] de document XML simplifié est présenté dans la Figure 2.1 où sont introduits sa représentation arborescente (data tree) d'une part et sa représentation séquentielle d'autre part. Dans cet exemple, le document est un élément dont la racine est le noeud étiqueté Solar ; dans la représentation séquentielle, il est indiqué par la balise ouvrante <Solar> qui se trouve au début du document et la balise fermante </Solar> qui ferme le document. Le rôle des balises est clair : elles délimitent le contenu de chaque élément. Dans les cas les plus fréquents, ce contenu peut être simplement un noeud texte correspondant aux feuilles de l'arbre XML comme dans <name>TheSun</name>, ou bien une séquence d'éléments à leur tour composés d'autres éléments comme c'est le cas de <planet><name>Mercury</name></planet>. Parfois, le contenu d'un élément peut être un mélange de noeuds textes et d'éléments qu'on appelle *mixed-content* dans le jargon XML. Le contenu du noeud étiqueté par note est un exemple de mixed-content.

1. Cet exemple est une adaptation des documents traitant du système solaire diffusés sur internet.

Simplication : Par souci de simplicité, dans cette présentation, nous ne nous intéressons qu'au contenu d'un document XML c'est à dire au 'document element'. Ainsi le noeud document (dans la littérature applelé 'document node' ou encore 'document root') n'est pas considéré. Rappelons que le noeud document n'est pas un noeud élément et qu'il sert essentiellement comme point d'entrée au document. Nous insistons sur le fait que dans la Figure 2.1, le noeud étiqueté Solar est la racine de l'arbre des éléments du document, que nous nommons racine du document.

```
<Solar>
<galaxy>The milky way</galaxy>
<star>The sun</star>
<planet>
  <name>Mercury</name>
  <orbit>57.9</orbit>
  <note>The innermost
   <bold>planet</bold>
         in the system
  </note>
</planet>
</Solar>
```

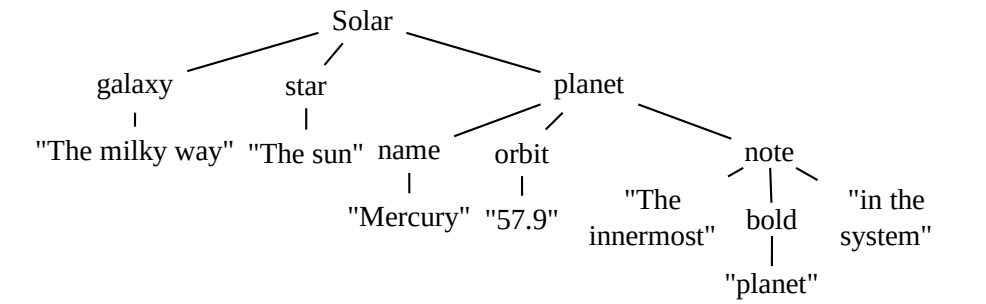


FIGURE 2.1 – Un Document XML et sa représentation arborescente

Un des aspects importants de XML concerne l'ordre entre les noeuds. Lorsqu'on fait référence à l'ordre des fils d'un noeud, on parle de *sequence order*. Lorsqu'on fait référence à l'ordre des noeuds d'un document parcouru en profondeur et en partant de la gauche, on parle de *document order*. Dans les deux cas, l'ordre est naturellement reflété dans la représentation textuelle par la précédence de l'ouverture des balises des noeuds concernés. Dans notre exemple, star précède planet suivant le *sequence-order* et précède note suivant le *document order*.

Bien que le standard XML ne restreint pas la façon dont les éléments d'un document XML doivent être organisés, il requiert que les documents soient au moins bien parenthésés (*well-formed* dans le jargon XML). Cela consiste à s'assurer qu'à chaque balise ouvrante correspond une balise fermante et que l'ordre de fermeture des balises respecte l'ordre de leur ouverture. Il est facile de vérifier que le document de la Figure 2.1 est bien parenthésé.

Il est parfois intéressant de pouvoir spécifier d'autres types de contraintes portant sur la structure des documents. Ce besoin peut découler soit de la nécessité de respecter un format précis imposé par

l'application manipulant les données soit du simple besoin d'associer une sémantique aux données. Dans les deux cas de figure, les schémas s'avèrent être la solution privilégiée pour spécifier l'ensemble des instances *valides*. Ils permettent principalement de spécifier le nom de balises permises, l'ordre d'apparition des éléments et leur occurrence ainsi que le contenu de chacun des éléments.

Il existe différents langages pour spécifier un schéma XML. Les deux langages les plus connus et les plus utilisés sont :

– DTD (Document Type Definition) [DTD] ,
– XSD (XML Schema Definition) [XSD].

La différence entre ces deux langages porte d'abord sur leur syntaxe mais aussi sur leur expressivité, XSD étant strictement plus expressif que DTD. Par simplicité, nous nous focalisons sur le langage DTD.

Un schéma DTD (ou une DTD) consiste en une suite de déclarations qui contraignent le contenu de chaque élément du document. La syntaxe de base d'une déclaration est donnée sous la forme suivante :

```
<!ELEMENT NomElem (ContenuElem)>
```

où `NomElem` est l'étiquette de l'élément défini et `ContenuElem` est une expression régulière définissant son contenu. La Figure 2.2 illustre une DTD pour les données du système solaire. La première ligne (qui débute par `<!DOCTYPE>...`) est spécifique aux DTD et sert à donner un nom au schéma (`Solar` dans notre exemple). La deuxième ligne définit le contenu de l'élément `Solar` qui consiste en `galaxy`, `star`, et en une ou plusieurs occurrences de `planet` suivies de zéro ou plusieurs occurrences d'éléments `comet`. On comprend alors le rôle des symboles + et $*$ placés après les étiquettes. La troisième ligne définit le contenu de l'élément `galaxy` qui consiste en une chaîne de caractères, indiquée par le mot clé `PCDATA` (`ParsedCharacterDATA`). La cinquième ligne définit le contenu de `planet` qui consiste en `name` suivi soit de `orbit` ou de `diameter` mais pas des deux, éventuellement suivi de l'élément `note` qui est optionnel lui même suivi de zéro ou de plusieurs occurrence de `satellite`. Il aisé de vérifier que le document de la Figure 2.1 est valide pour la DTD de la Figure 2.2.

```
1. <!DOCTYPE Solar [
2. <!ELEMENT Solar (galaxy,star,planet+,comet*)>
3. <!ELEMENT galaxy (#PCDATA) >
4. <!ELEMENT star (#PCDATA) >
5. <!ELEMENT planet (name, (orbit | diameter),note?,satellite*)>
6. <!ELEMENT name (#PCDATA) >
7. <!ELEMENT diameter (#PCDATA) >
8. <!ELEMENT orbit (#PCDATA) >
9. <!ELEMENT note (#PCDATA | bold | emph | href)* >
10.<!ELEMENT satellite (name, note) >
11.<!ELEMENT comet (desc?,note) >
12.<!ELEMENT desc (#PCDATA) >
]>
```

FIGURE 2.2 – DTD pour le document SolarSystem

Avant d'entamer les définitions formelles du modèle de données XML, il convient de souligner quelques points concernant les conventions utilisées dans la suite de ce manuscrit ainsi que la

simplification du modèle de données par rapport à celui suggéré par W3C [XDM]. Comme il est d'usage dans les travaux relatifs à XML, nous ne considérons pas dans notre étude tous les types d'éléments définis par cette spécification, tels que les attributs et les instructions de pré-traitement. Notre travail peut facilement être étendu pour prendre en compte les types d'éléments. Concernant la terminologie utilisée, nous entendons par document XML l'arbre XML sous-jacent. De la même manière, la représentation textuelle est abandonnée en faveur de la représentation arborescente qui offre une meilleure compréhension de la structure des données.

2.2 Modèle de données XML

Le modèle de données que nous utilisons est celui défini dans [BC09] qui introduit la notion de *store XML*. Cette notion capture les noeuds par des identifiants pour lesquels quelques notations sont présentées ci-dessous.

Notation
Nous noterons les identifiants associés aux noeuds d'un store par les lettres en gras $\mathbf{i}, \mathbf{j}, \ldots$. Nous aurons souvent à manipuler des ensembles d'identifiants (id-sets) ainsi que des listes d'identifiants (id-seqs). Nous utiliserons $\mathrm{I}, \mathrm{J}, \mathrm{K}...$ pour noter à la fois les id-sets et les id-seqs. Par ailleurs, nous noterons :

- $()$ l'id-seq vide,
- $\mathrm{I} \cdot \mathrm{J}$ la concaténation de deux id-seqs I et J, et
- $\mathrm{I}_{|\mathrm{J}}$ l'intersection d'un id-seq I avec un id-set (resp. id-seq) J préservant l'ordre des identifiants dans I.

Définition 1 (Store XML)**.**
Soit I un id-set.
Un *store* σ défini sur I est une application associant chaque identifiant $\mathbf{i} \in \mathrm{I}$

- soit à un élément $a[\mathrm{J}]$ donné par son étiquette a et la liste J des identifiants de ses fils,
- soit à un noeud texte $text[st]$ où st est une chaîne de caractères.

Dans la suite, le domaine I de σ est noté $dom(\sigma)$. Le store vide sera noté $\varnothing$. Nous utiliserons parfois la notation $\sigma@i$ pour désigner le noeud du store σ identifié par $\mathbf{i}$.

Exemple 1. La Figure 2.3 présente le document 'Solarsystem' de la Figure 2.1 auquel, pour chaque noeud, a été ajouté un identifiant entre crochets en-dessous de l'étiquette du noeud. Ainsi, la Figure 2.3 est une représentation arborescente d'un store σ associé à ce document et qui est aussi spécifié par la table ci-dessous. Notons que $dom(\sigma) = \{\mathbf{i}_0, \ldots, \mathbf{i}_{14}\}$.

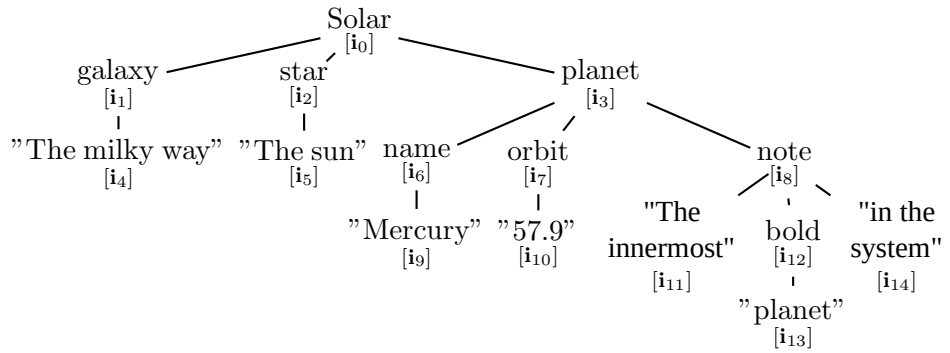


FIGURE 2.3 – Un document XML annoté avec les identifiants de ses noeuds

Identifiant	**étiquette**	**id-seq des fils / valeur texte**
$\mathbf{i}_0$	Solar	$\mathbf{i}_1 \cdot \mathbf{i}_2 \cdot \mathbf{i}_3$
$\mathbf{i}_1$	galaxy	$\mathbf{i}_4$
$\mathbf{i}_2$	star	$\mathbf{i}_5$
$\mathbf{i}_3$	planet	$\mathbf{i}_6 \cdot \mathbf{i}_7 \cdot \mathbf{i}_8$
$\mathbf{i}_4$	$text$	"The milky way"
$\mathbf{i}_5$	$text$	"The sun"
$\mathbf{i}_6$	name	$\mathbf{i}_9$
$\mathbf{i}_7$	orbit	$\mathbf{i}_{10}$
$\mathbf{i}_8$	note	$\mathbf{i}_{11} \cdot \mathbf{i}_{12} \cdot \mathbf{i}_{14}$
$\mathbf{i}_9$	$text$	"Mercury"
$\mathbf{i}_{10}$	$text$	"57.9"
$\mathbf{i}_{11}$	$text$	"The innermost"
$\mathbf{i}_{12}$	bold	$\mathbf{i}_{13}$
$\mathbf{i}_{13}$	$text$	"planet"
$\mathbf{i}_{14}$	$text$	"in the system"

■

Les fonctions suivantes sont classiques et utiles à la formalisation du modèle de données et des langages manipulant des stores.

Définition 2.
Soit I, J deux id-sets tel que J $\subseteq$ I. Soit σ un store sur I. On définit :

- *lab*$_\sigma(\mathbf{i})$=a si $\sigma(\mathbf{i})$=a[K], et *lab*$_\sigma(\mathbf{i})$=*String* si $\sigma(\mathbf{i})$=$text[st]$.
- *Child*$(\sigma, \mathrm{J}) = \{\mathbf{i} \mid \exists \mathbf{j} \in \mathrm{J}, \sigma(\mathbf{j})=a[\mathrm{K}]$ et $\mathbf{i} \in \mathrm{K}\}$.
- *Parent*$(\sigma, \mathrm{J}) = \{\mathbf{i} \mid \mathrm{J} \cap$ *Child*$(\sigma, \{\mathbf{i}\}) \neq \emptyset\}$.
- *roots*$(\sigma) = \{\mathbf{i} \mid \neg\exists\, \mathbf{j}, \mathbf{i} \in$ *Child*$(\sigma, \{\mathbf{j}\})\}$
- *Desc*(σ, J) est défini classiquement par fermeture transitive, réflexive de *Child*(σ, J)
- *Ancs*(σ, J) est défini classiquement par fermeture transitive, réflexive de *Parent*(σ, J)

Dans la suite de ce manuscrit, par simplicité et sans ambiguïté, la fonction *lab*$_\sigma(\mathbf{i})$ ne sera pas indicée avec le store σ.

Les fonctions suivantes permettent de créer un nouveau store ou de modifier le contenu d'un store existant.

Définition 3.
Soient σ et σ' deux stores sur I et J respectivement, tels que $\mathrm{I} \cap \mathrm{J} = \emptyset$. Soit un id-seq L tel que $\mathrm{L} \subseteq (\mathrm{I} \cup \mathrm{J})$. Soient $\mathbf{i}$ et $\mathbf{j}_1$ deux identifiants tels que $\mathbf{i} \in \mathrm{I}$ et $\mathbf{j}_1 \notin (\mathrm{I} \cup \mathrm{J})$. Alors :

- $\sigma{\cdot}\sigma'$ désigne l'union des deux stores σ et σ'.
- $Store(\mathbf{j}_1, a, \sigma)$ désigne le store σ_1 obtenu en ajoutant le noeud $\mathbf{j}_1$ étiqueté par a comme racine des éléments du store σ. On a donc :
  - $\mathit{dom}(\sigma_1) = \mathit{dom}(\sigma) \cup \{\mathbf{j}_1\}$,
  - $\sigma_1(\mathbf{j}_1) = a[\mathrm{K}]$ où $\mathrm{K}{=}\mathit{roots}(\sigma)$, et
  - pour tout $\mathbf{j}{\in}\mathit{dom}(\sigma)$, $\sigma_1(\mathbf{j}) = \sigma(\mathbf{j})$.

  Notons que $\mathit{roots}(\sigma_1){=}\{\mathbf{j}_1\}$.

- $Texte(\mathbf{j}_1, st)$ est le store σ_1 constitué d'un noeud texte st donc :
  - $\mathit{dom}(\sigma_1) = \{\mathbf{j}_1\}$, et
  - $\sigma_1(\mathbf{j}_1) = text[st]$.

- $SubStore(\sigma, \mathbf{i}, b[\mathrm{L}], \sigma')$ est le store σ_1 obtenu en remplaçant l'étiquette du noeud $\mathbf{i}$ par b et en remplaçant ses fils par L, sachant que la description des éléments dont la racine est L est contenue dans $\sigma{\cdot}\sigma'$. Donc :
  - $\mathit{dom}(\sigma_1){=}\mathit{dom}(\sigma) - \mathrm{K}^* \cup \mathrm{L}^*$ où
    $\mathrm{K}^*{=}\mathit{Desc}(\sigma, \mathrm{K})$ avec $\sigma(\mathbf{i}){=}a[\mathrm{K}]$, et
    $\mathrm{L}^*{=}\mathit{Desc}(\sigma{\cdot}\sigma', \mathrm{L})$.
  - $\sigma_1(\mathbf{i}){=}b[\mathrm{L}]$, et
  - pour tout $\mathbf{j}{\in}\mathit{dom}(\sigma_1)$ tel que $\mathbf{j}{\neq}\mathbf{i}$, $\sigma_1(\mathbf{j}){=}\sigma(\mathbf{j})$ si $\mathbf{j}{\in}\mathit{dom}(\sigma)$ et $\sigma_1(\mathbf{j}){=}\sigma'(\mathbf{j})$ si $\mathbf{j}{\in}\mathit{dom}(\sigma')$.

La fonction $SubStore$ est illustrée à l'aide de l'exemple suivant.

Exemple 2. Soient σ et σ' deux stores sur I et J respectivement tels que $\mathrm{I} \cap \mathrm{J} = \emptyset$.
Soit $\mathbf{i} \in \mathit{dom}(\sigma)$ avec $\sigma(\mathbf{i}) = a[\mathrm{K}_1 \cdot k \cdot \mathrm{K}_2]$.
Soit $\mathrm{L} = \mathrm{K}_1 \cdot \mathbf{j}_1 \cdot \mathbf{j}_2 \cdot \mathrm{K}_2$ un id-seq tel que $\mathbf{j}_1, \mathbf{j}_2 \in \mathit{dom}(\sigma')$.
La Figure 2.4 présente :

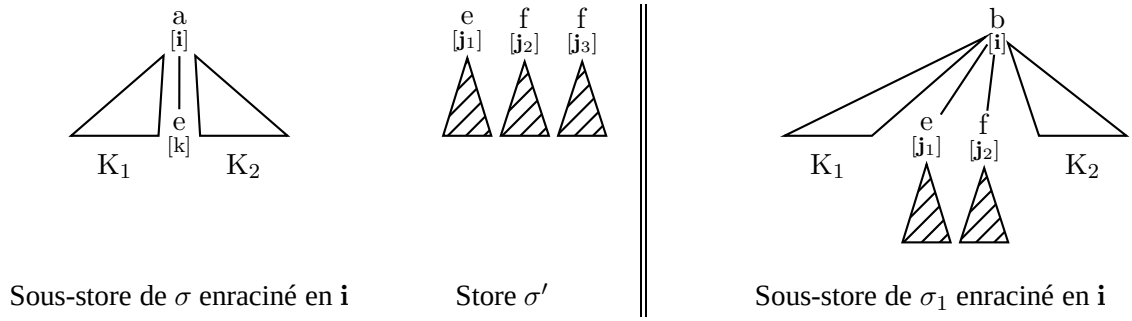


FIGURE 2.4 – Illustration de $SubStore$

- le sous-store de σ enraciné en **i**,
- le store σ' et
- le résultat de l'application de la fonction $SubStore$ sur σ remplaçant le contenu de son élément identifié **i** par l'élément $b[\mathrm{L}]$.

■

Nous définissons la projection d'un store σ comme étant une restriction de l'application σ relativement à un id-set.

Définition 4 (Projection $\Pi_{\mathrm{J}}(\sigma)$)**.**
Soient I et J deux id-sets tels que J $\subseteq$ I. Soit σ un store sur I.
La *projection* de σ suivant J, notée $\Pi_{\mathrm{J}}(\sigma)$, est définie, pour tout **j** $\in$ J, comme suit :
- si $\sigma(\mathbf{j}) = a[\mathrm{K}]$ alors $\Pi_{\mathrm{J}}(\sigma)(\mathbf{j}) = a[\mathrm{K}_{|\mathrm{J}}]$,
- sinon $\sigma(\mathbf{j}) = text[st]$ et $\Pi_{\mathrm{J}}(\sigma)(\mathbf{j}) = \sigma(\mathbf{j})$.

Notons que le domaine et le co-domaine de la projection de σ suivant J est J.

Exemple 3. Considérons σ le store défini dans l'exemple 1.
Soit J $= \{\mathbf{i}_0, \mathbf{i}_2, \mathbf{i}_3, \mathbf{i}_6, \mathbf{i}_9, \mathbf{i}_{12}\}$. $\Pi_{\mathrm{J}}(\sigma)$ la projection de σ suivant J est décrite par la table ci-dessous.

Identifiant	**étiquette**	**id-seq fils / valeur texte**
$\mathbf{i}_0$	Solar	$\mathbf{i}_2 \cdot \mathbf{i}_3$
$\mathbf{i}_2$	star	()
$\mathbf{i}_3$	planet	$\mathbf{i}_6$
$\mathbf{i}_6$	name	$\mathbf{i}_9$
$\mathbf{i}_9$	$text$	"Mercury"
$\mathbf{i}_{12}$	bold	()

■

Nous constatons que la projection d'un store correspondant à un arbre ne retourne pas forcément un arbre. Dans l'exemple 3, $\Pi_{\mathrm{J}}(\sigma)$ n'est pas un arbre parce que le noeud $\mathbf{i}_{12}$ n'est pas connecté au noeud $\mathbf{i}_3$ (le noeud $\mathbf{i}_8$ parent de $\mathbf{i}_{12}$ n'est pas projeté).

Dans notre étude, nous ne considérons que les stores correspondant à des arbres et à des forêts XML. Les arbres et les forêts sont notés de la façon suivante (I et J sont deux id-sets) :

[Forêts] Une forêt f sur I est donnée par la paire (J, σ_f) où σ_f est un store sur I et J est l'id-seq des racines de f, i.e J $=$ *roots*(σ_f). Par abus, on utilisera parfois f pour désigner le store σ_f.

[Arbres] Un arbre t sur I est donné par (r_t, σ_t) où r_t est l'identifiant de la racine de t, i.e. *roots*(σ_t)=$\{r_t\}$. Par abus, on utilisera parfois t pour désigner le store σ_t.
subfor(t) désigne la sous-forêt de t correspondant aux fils de la racine de t, autrement dit : *subfor*(t)=$\Pi_{\mathrm{I}-\{r_t\}}(t)$.

Pour les besoins du développement formel, nous serons amenés à considérer des stores, appelés *p-store* (ou *p-document*) dont la particularité est que leurs identifiants servent à coder la position des noeuds au sein du document XML et plus précisément à coder le *document-order*. Le codage choisi est connu sous le nom de *Dewey Order* dans la littérature [TVB+02]. Dans ce système, l'identifiant pour un noeud consiste en l'identifiant de son parent concaténé à la position de ce noeud parmi ses frères.

Exemple 4. Le document de la Figure 2.5 est un *p-document*.

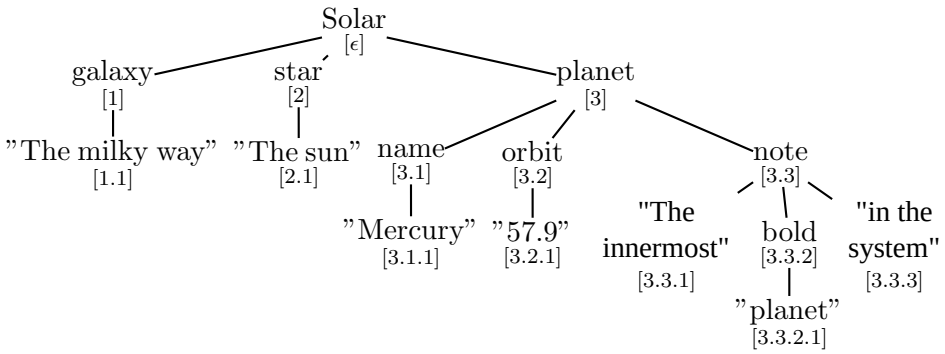


FIGURE 2.5 – Exemple d'un p-document

■

Nous considérons dans cette thèse des documents XML valides relativement à un schéma défini au moyen de DTD. La définition qui suit s'inspire de [GMN07].

Définition 5 (DTD)**.**
Soit Σ un ensemble d'étiquettes, soit *String* un symbole réservé. Une DTD sur Σ est un couple (D, s_D) où D est une fonction totale de Σ vers l'ensemble des expressions régulières définies sur $\Sigma \cup \{\textit{String}\}$ et où $s_D \in \Sigma$ est le symbole de départ.

Dans la suite du manuscrit, par simplicité, nous notons une DTD (D, s_D) par D.

Etant donnée une expression régulière r, $\mathcal{L}(r)$ désigne le langage généré par r et $\mathcal{S}(r)$ désigne l'ensemble de symboles de Σ apparaissant dans r.

Exemple 5. Soit D la DTD de la Figure 2.2.
Le contenu du noeud Solar est donné par la règle : $\text{Solar} \rightarrow \text{galaxy}, \text{star}, \text{planet}+, \text{comet}*$.
Il est facile de voir que :
- $\text{galaxy} \cdot \text{star} \cdot \text{planet} \in \mathcal{L}(\text{D}(\text{Solar}))$, et
- $\mathcal{S}(\text{D}(\text{Solar})) = \{\text{galaxy}, \text{star}, \text{planet}, \text{comet}\}$.

■

Définition 6 (Document valide relativement à une DTD)**.**
On dit que l'arbre XML t est valide par rapport à la DTD D, noté $t \in D$ ssi :
- $\sigma_t(r_t)=s_D[\text{J}]$ et
- pour tout $\mathbf{i}\in\text{J}$, si $\sigma_t(\mathbf{i})=a[\text{K}]$ alors :
  (a) $\text{K}=()$ implique que $\epsilon \in \mathcal{L}(D(a))$
  (b) $\text{K}=\mathbf{i}_1 \cdot ... \cdot \mathbf{i}_n$ implique que $\textit{lab}(\mathbf{i}_1) \cdot \ldots \cdot \textit{lab}(\mathbf{i}_n) \in \mathcal{L}(D(a))$.

Exemple 6. Considérons le document de la Figure 2.3 et appelons-le t. Soit la DTD de la Figure 2.2 qu'on note D. Il est aisé de vérifier que $t \in D$. En effet, on a :
- s_D =Solar et $\sigma_t(r_t) = Solar[\mathbf{i}_1 \cdot \mathbf{i}_2 \cdot \mathbf{i}_3]$ et
- $lab(\mathbf{i}_1) \cdot lab(\mathbf{i}_2) \cdot lab(\mathbf{i}_3)$=galaxy $\cdot$ star $\cdot$ planet $\in \mathcal{L}(\mathrm{D(Solar)})$, ...

■

2.3 Le langage XPath

L'interrogation, la transformation et la mise à jour sont des fonctionnalités qui requièrent de naviguer dans la structure arborescente des documents XML. XPath est le langage permettant d'assurer cette fonctionnalité. Il a été introduit par le W3C [XPa] pour être utilisé comme sous-langage des langages d'interrogation XQuery [XQu], de transformation XSLT [XSL] ainsi que dans tous les langages manipulant des données XML.

XPath permet de sélectionner les noeuds d'un document XML sur la base de contraintes spécifiées par des expressions appelées *chemins*. Un chemin est spécifé comme une suite d'étapes notées $\boldsymbol{step}$ et séparées par des "/"

$$[/]\boldsymbol{step_1}/\boldsymbol{step_2}/\ldots/\boldsymbol{step_n}$$

Un chemin commence parfois par le symbole "/" indiquant qu'il s'évalue par rapport au noeud du document (document node). Dans ce cas, le chemin est dit *absolu*. Rappelons ici que, dans notre modèle de données dit simplifié, nous avons fait abstraction du 'document node'. Donc pour cette présentation, l'évaluation d'un chemin absolu est décalée au niveau de la racine de l'arbre des éléments (data tree). Cela a pour conséquence une écriture légèrement différente des chemins qui pourra être observée dans nos exemples.

Il est parfois utile qu'un chemin puisse s'évaluer relativement à un noeud du document autre que le document node. Dans ce cas, on parle de chemins *relatifs*. Dans tous les cas, l'évaluation d'un chemin s'effectue à partir d'un ensemble de noeuds *sources* (appelé aussi *contexte* dans la littérature), et produit un ensemble de noeuds *cibles* obtenus en évaluant successivement toutes les étapes composant le chemin.

Comme nous venons de le mentionner, les étapes constituent les composants de base d'un chemin. Leur forme générale est :

$$step = \boldsymbol{Axis}::\boldsymbol{Test}[\boldsymbol{cond}]$$

Les deux principaux composants d'une étape $step$ sont l'axe de navigation $\boldsymbol{Axis}$ qui spécifie la direction de navigation et le filtre $\boldsymbol{Test}$ qui sert à restreindre les noeuds sur la base de leur type (noeuds éléments, noeuds textes, ...) ou de leur étiquette. Comme il est parfois utile d'exprimer des conditions sur la valeur ou la structure des noeuds qu'une étape doit sélectionner, XPath permet l'utilisation de prédicats notés $\boldsymbol{cond}$ exprimant ces conditions.

Nous allons maintenant détailler chacun des trois composants d'une étape.

Les axes Comme précédemment évoqué, l'axe spécifie la direction empruntée par la navigation dans le document interrogé. Etant donnée la structure arborescente des documents XML, on peut considérer deux sortes de navigation : en profondeur et en largeur. Les axes permettant une navigation en profondeur sont appelés axes *verticaux*. Parmi ces axes, certains peuvent naviguer niveau par niveau dans l'arbre tels que l'axe `child` et l'axe `parent` ou en traversant plusieurs niveaux tels que l'axe `descendant` (`desc`) et l'axe `ancestor` (`ancs`). La signification de ces axes est assez simple : l'axe `child`, respectivement `parent` sélectionne les noeuds fils, respectivement le noeud parent du noeud source. L'axe `desc`, respectivement `ancs` sélectionne les noeuds descendants, respectivement les noeuds ancêtres du noeud source. Les axes permettant une navigation en largeur sont appelés axes *horizontaux*. Les plus simples sont `following-sibling` et `preceding-sibling` qui permettent d'atteindre les noeuds frères à droite (respectivement à gauche) du noeud source. Les autres axes horizontaux `preceding` et `following` permettent d'atteindre tous les noeuds "précédents" (respectivement "suivants") du noeud source suivant le document-order en excluant ses descendants et ses ancêtres. La Figure 2.6 illustre les noeuds sélectionnés par les axes `preceding` et `following` pour un noeud source donné.

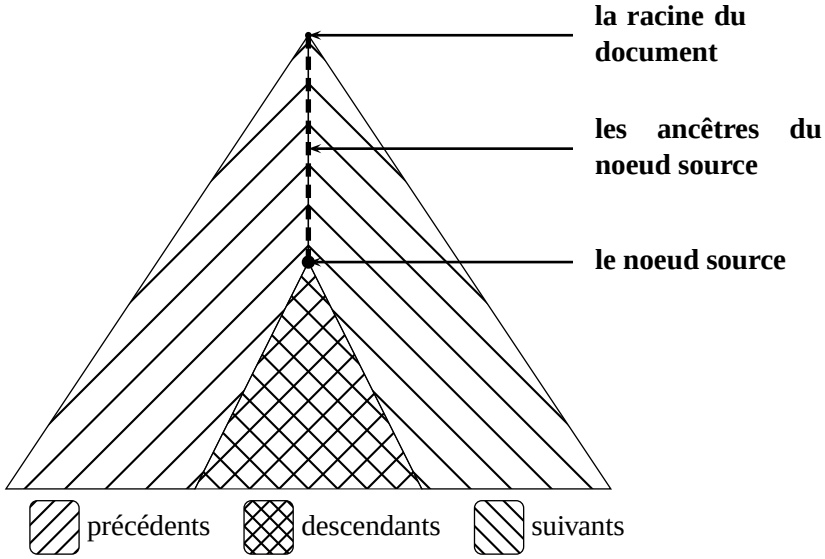


FIGURE 2.6 – Illustration des axes `preceding` et `following`

Les filtres Une fois les noeuds sélectionnés au moyen des axes de navigation, le filtre est utilisé pour spécifier quels noeuds retenir en fonction de leur type ou de leur étiquette. Les filtres utilisés sont :

- `node()` qui permet de sélectionner à la fois les éléments et les textes,
- `text()` qui permet de sélectionner uniquement les textes et
- $etiq$ qui spécifie l'étiquette que doivent avoir les noeuds retournés.

Les conditions Les axes et les tests se limitent à sélectionner des noeuds sur la base d'information structurelle. Il est parfois utile de sélectionner des noeuds sur la base de leur contenu ce qui nécessite de pouvoir exprimer des conditions au niveau des étapes. Ces conditions peuvent être soit

structurelles, auquel cas elles sont exprimées au moyen de chemins, soit faire appel à des fonctions pré-définies dans le but de faire un traitement particulier (concaténation des noeuds textes par exemple).

Avant d'entamer la définition de la syntaxe du langage de chemins que nous considérons, il est important de souligner que plusieurs fragments de XPath ont été étudiés dans la littérature [MdR05], [GKPS05] et [BK08]. Ces fragments sont définis sur des critères d'expressivité et de complexité. Le fragment *Core XPath* proposé par [GKP02] capture les fonctionnalités de base de XPath. Il permet d'exprimer des chemins avec des conditions. Ces conditions se limitent toutefois à des conjonctions et des disjonctions d'expressions. Ce fragment exclut l'utilisation des fonctions pré-définies et la comparaison des textes. Le fragment *Conditional XPath* proposé par [Mar05] étend le fragment *Core XPath* en permettant d'exprimer des chemins dits itératifs qui n'existent pas dans le langage XPath tel que défini par le W3C.

2.3.1 Syntaxe de XPath

Nous présentons à présent la syntaxe du langage de chemins. Comme notre étude s'appuie sur la technique de projection développée dans [BCCN06], nous utilisons un langage de chemins équivalent à celui défini dans [BCCN06]. La particularité de ce langage est qu'il ne permet pas d'exprimer de conditions imbriquées. Ceci requiert d'avoir une syntaxe à deux niveaux :

- un niveau pour définir les chemins *simples* utilisés par les conditions et dans lesquels il est interdit d'imbriquer des conditions, et
- un autre niveau pour définir les chemins avec conditions.

P	::=	$STEP \mid STEP/P$	
$STEP$	::=	$step \mid step[cond]$	
$step$	::=	$Axis :: Test$	
$Axis$	::=	`self` | `child` | `parent` | `desc` | `ancs` | `desc-or-self` | `ancs-or-self` |	
		`following` | `following-sibling` | `preceding` | `preceding-sibling`	
$Test$	::=	`text`() | `node`() | `tag`	
$cond$	::=	$Exp \mid cond\ ct\ cond$	$ct \in \{\vee, \wedge\}$
Exp	::=	$Arit \mid Exp\ cp\ Exp$	$cp \in \{=, \neq, <, >, \leq, \geq\}$
$Arit$	::=	$Atom \mid Arit\ op\ Arit$	$op \in \{+, -, *, div, mod\}$
$Atom$	::=	$f(Exp, \ldots, Exp) \mid p \mid st$	
p	::=	$step \mid step/p \mid /p$	

FIGURE 2.7 – Syntaxe du langage de chemins

La syntaxe du langage de chemins que nous considérons est donnée en Figure 2.7.

Les chemins (avec condition) sont générés par le non-terminal P. Il est composé d'un nombre arbitraire d'étapes $STEP$ pouvant contenir une condition $cond$. Une condition $cond$ est construite soit par conjonction $(\wedge)$ soit par disjonction $(\vee)$ de conditions. Elle peut être aussi une expression

de comparaison Exp construite à partir d'expressions de comparaison en utilisant les opérateurs de comparaison $=, \neq, \ldots$ ou d'expressions arithmétiques $Arit$ elles mêmes construites à partir d'un litéral st qui consiste en une constante de type *String*, d'un chemin simple p (qui peut être absolu) ou d'une fonction f ayant pour argument une ou plusieurs expressions de comparaisons. Il existe plusieurs sortes de fonctions dans XPath. Les plus utilisées restent les fonctions d'agrégat telles que `count(exp)` ou `sum(exp)` et les fonctions de positions `first()`, `last()` et `position()`. Notons que dans XPath l'opérateur de négation est exprimée à l'aide d'une fonction `not(exp)`.

Les chemins simples sont générés avec le non-terminal p et peuvent contenir un nombre arbitraire d'étapes. Une étape est générée par le non-terminal $step$, elle est composée d'un axe *Axis* et d'un test *Test*. Noter que cette syntaxe permet de générer des chemins simples relatifs et absolus à la fois. Ceci répond au besoin d'exprimer dans les conditions les deux types de chemins.

Dans la suite de manuscrit, nous ne considérons pas les fonctions sauf la négation.

Exemple 7. Considérons les chemins suivants :
- $\mathrm{P}_1 = \texttt{self} :: \mathrm{Solar}/\mathrm{child} :: \mathrm{planet}$,
- $\mathrm{P}_2 = \texttt{self} :: \mathrm{Solar}/\mathrm{desc} :: \mathrm{name}$,
- $\mathrm{P}_3 = \texttt{self} :: \mathrm{Solar}/\mathrm{child} :: \mathrm{planet}[\mathrm{child} :: \mathrm{note}]$, et
- $\mathrm{P}_4 = \texttt{self} :: \mathrm{Solar}/\mathrm{child} :: \mathrm{planet}[\mathrm{child} :: \mathrm{name} = \text{"Uranus"}]/\mathrm{satellite}$.

Les chemins P_1 et P_2 ne contiennent pas de conditions : ce sont des chemins simples. L'évaluation du chemin P_1 (à partir du noeud racine du document) retourne simplement les noeuds planet qui sont fils du noeud Solar. Avec la même convention, le chemin P_2 retourne les noeuds name qui sont descendants du noeud Solar. Les chemins P_3 et P_4 sont des chemins avec conditions. Le chemin P_3 a pour préfixe P_1 et sélectionne parmi les noeuds retournés par P_1 ceux qui possèdent un noeud fils étiqueté note. Cette condition est simplement exprimée par le chemin relatif $\mathrm{child} :: \mathrm{note}$. Le chemin P_4 sélectionne d'abord les noeuds planet dont le noeud fils name a pour valeur "Uranus" puis retourne leurs noeuds fils $\mathrm{satellite}$. ■

Dans la suite de ce manuscrit, nous utilisons les abbréviations suivantes :
- $/Test$ pour $/\texttt{child} :: Test$, et
- $//\mathrm{Test}$ pour $/\texttt{desc-or-self} :: node()/\texttt{child} :: Test$.

2.3.2 Sémantique

La sémantique de XPath a été définie par le W3C dans [XS] et dans plusieurs travaux de recherche tels que [GKP02] et [BK08]. La définition que nous avons choisie pour la sémantique s'inspire fortement de celle définie par [BK08].

La sémantique des chemins, utilise le jugement principal suivant :

$$\sigma, \mathcal{S} \models P \overset{\mathsf{step}}{\Rightarrow} \mathcal{R} \quad (\mathbf{Jug{-}Path})$$

Ce jugement entend spécifier qu'étant donné un store σ, l'évaluation d'un chemin P, à partir d'un id-set source $\mathcal{S}$, retourne un id-set cible $\mathcal{R}$ contenant les réponses de l'évaluation de P sur σ.

Afin de traiter les conditions exprimées dans les chemins, la spécification du jugement **Jug-Path** utilise le jugement intermédiaire ci-dessous :

$$\sigma, \mathcal{S} \models cond \overset{\mathsf{cond}}{\Rightarrow} \mathcal{S}' \quad (\mathbf{Jug-Cond})$$

Ce jugement entend spécifier qu'étant donné un store σ, l'évaluation de la condition $cond$, à partir d'un id-set source $\mathcal{S}$, retourne un id-set cible $\mathcal{S}'$ correspondant aux noeuds de $\mathcal{S}$ qui satisfont la condition $cond$.

Nous introduisons deux fonctions $PEval(-,-)$ et $PEvalCond(-,-,-)$ comme notations alternatives aux jugements **Jug-Path** et **Jug-Cond** et qui sont définis comme suit :

$PEval_{\mathcal{S}}(\sigma, P) = \mathcal{R}$	ssi	$\sigma, \mathcal{S} \models P \overset{\mathsf{step}}{\Rightarrow} \mathcal{R}$
$PEvalCond(\sigma, \mathcal{S}, cond) = \mathcal{S}'$	ssi	$\sigma, \mathcal{S} \models cond \overset{\mathsf{cond}}{\Rightarrow} \mathcal{S}'$

$$PEval(\sigma, P) \quad = \quad PEval_{\mathit{roots}(\sigma)}(\sigma, P)$$

Notons que la fonction $PEval(\sigma, P)$ initialise l'évaluation de P avec le noeud racine du store.

Enfin, les fonctions **Nav**$(\sigma, \mathrm{J}, Axis)$ et **Filtre**$(\sigma, \mathrm{J}, Test)$ sont définies, respectivement
- pour sélectionner les noeuds du store σ atteignables à partir des noeuds de J suivant l'axe $Axis$,
- pour éliminer de J les noeuds ne staisfaisant pas $Test$.

$$\begin{array}{lcll}
\textbf{Nav}(\sigma, \mathrm{J}, Axis) & ::= & \mathrm{J} & \text{if } Axis = \texttt{self} \\
 & & \mathit{Child}(\sigma, \mathrm{J}) & \text{if } Axis = \texttt{child} \\
 & & \mathit{Desc}(\sigma, \mathrm{J}) & \text{if } Axis = \texttt{desc} \\
 & & \mathit{Parent}(\sigma, \mathrm{J}) & \text{if } Axis = \texttt{parent} \\
 & & \mathit{Ancs}(\sigma, \mathrm{J}) & \text{if } Axis = \texttt{ancs} \\
 & & & \\
\textbf{Filtre}(\sigma, \mathrm{J}, Test) & ::= & \{\mathbf{i} \in \mathrm{J} \mid \mathit{lab}(\mathbf{i}) = tag\} & \text{if } Test = tag \\
 & & \{\mathbf{i} \in \mathrm{J} \mid \mathit{lab}(\mathbf{i}) = \mathit{String}\} & \text{if } Test = text \\
 & & \{\mathbf{i} \in \mathrm{J} \mid \mathit{lab}(\mathbf{i}) \neq \mathit{String}\} & \text{if } Test = node
\end{array}$$

Nous sommes maintenant prêts pour présenter les règles d'inférence spécifiant les jugements **Jug-Path** et **Jug-Cond**. Ces règles sont données dans la Figure 2.8. Elles sont bien évidemment regroupées en suivant la définition syntaxique des chemins. Chaque groupe de règles est commenté en ne ciblant que les aspects spécifiques.

Les trois premières règles constituent le point d'entrée. Les règles restantes spécifient l'évaluation des conditions structurelles, les seules à pouvoir être exprimées par notre fragment.

La règle (**Path**) spécifie que l'évaluation d'un chemin passe par l'évaluation successive de chacune de ses étapes. Cette règle s'applique aussi bien aux chemins simples (générés par le symbole

$$(\textbf{Path}) \quad \frac{\sigma, \mathcal{S} \models STEP \overset{\mathsf{step}}{\Rightarrow} \mathcal{R}_S \quad \sigma, \mathcal{R}_S \models P \overset{\mathsf{step}}{\Rightarrow} \mathcal{R}}{\sigma, \mathcal{S} \models STEP/P \overset{\mathsf{step}}{\Rightarrow} \mathcal{R}}$$

$$(\textbf{Step:simple}) \quad \frac{\mathrm{J} = \textbf{Nav}(\sigma, \mathcal{S}, Axis) \quad \mathcal{R} = \textbf{Filtre}(\sigma, \mathrm{J}, Test)}{\sigma, \mathcal{S} \models Axis :: Test \overset{\mathsf{step}}{\Rightarrow} \mathcal{R}}$$

$$(\textbf{Step:cond}) \quad \frac{\sigma, \mathcal{S} \models step \overset{\mathsf{step}}{\Rightarrow} \mathcal{R}_s \quad \sigma, \mathcal{R}_s \models cond \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}_c}{\sigma, \mathcal{S} \models step[cond] \overset{\mathsf{step}}{\Rightarrow} \mathcal{R}_c}$$

$$(\textbf{Cond:conj}) \quad \frac{\sigma, \mathcal{S} \models cond_1 \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}_1 \quad \sigma, \mathcal{R}_1 \models cond_2 \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}_2}{\sigma, \mathcal{S} \models cond_1 \wedge cond_2 \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}_2}$$

$$(\textbf{Cond:disj}) \quad \frac{\sigma, \mathcal{S} \models cond_1 \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}_1 \quad \sigma, \mathcal{S} \models cond_2 \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}_2}{\sigma, \mathcal{S} \models cond_1 \vee cond_2 \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}_1 \cup \mathcal{R}_2}$$

$$(\textbf{Cond:Neg}) \quad \frac{\sigma, \mathcal{S} \models cond \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}}{\sigma, \mathcal{S} \models not(cond) \overset{\mathsf{cond}}{\Rightarrow} \mathcal{S} - \mathcal{R}}$$

$$(\textbf{Cond:Exp}) \quad \frac{\mathcal{R} = \{\mathbf{i} \in \mathcal{S} \mid (Eval_{\{\mathbf{i}\}}(\sigma, Exp_1)\ cp\ Eval_{\{\mathbf{i}\}}(\sigma, Exp_2))\}}{\sigma, \mathcal{S} \models Exp_1\ cp\ Exp_2 \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}}$$

$$(\textbf{Cond:rel-path}) \quad \frac{\mathcal{R} = \{\mathbf{i} \in \mathcal{S} \mid PEval_{\{\mathbf{i}\}}(\sigma, p) \neq \emptyset\}}{\sigma, \mathcal{S} \models p \overset{\mathsf{cond}}{\Rightarrow} \mathcal{R}}$$

$$(\textbf{Cond:abs-path}) \quad \frac{PEval(\sigma, p) \neq \emptyset}{\sigma, \mathcal{S} \models /p \overset{\mathsf{cond}}{\Rightarrow} \mathcal{S}}$$

$$(\textbf{Cond:abs-path-empty}) \quad \frac{PEval(\sigma, p) = \emptyset}{\sigma, \mathcal{S} \models /p \overset{\mathsf{cond}}{\Rightarrow} \emptyset}$$

FIGURE 2.8 – Sémantique du langage de chemins

p de notre syntaxe) qu'aux chemins pouvant exprimer des conditions (ceux qui sont générés par le symbole P).

La règle (**Step:simple**) spécifie l'évaluation d'une étape simple ne contenant pas de condition. Cette évaluation consiste à pré-sélectionner les noeuds qui sont atteints à partir des noeuds de l'ensemble source $\mathcal{S}$ suivant l'axe $Axis$ puis à filtrer ces noeuds en utilisant $Test$.

La règle (**Step:cond**) spécifie l'évaluation d'une étape composée. Cette évaluation consiste à pré-sélectionner les noeuds accédés par $step$ puis à vérifier pour chacun d'eux la condition $cond$.

Les règles spécifiant l'évaluation des conditions sont présentées dans le deuxième encadré de la Figure 2.8. Cette évaluation s'effectue à partir d'un id-set source $\mathcal{S}$ et retourne un sous-ensemble de noeuds de $\mathcal{S}$ qui vérifient $cond$.

La règle (**Cond:conj**) spécifie l'évaluation d'une conjonction de conditions. Cette évaluation est triviale : l'une des deux conditions est évaluée puis la seconde condition est utilisée pour restreindre le résultat de la première.

La règle (**Cond:disj**) spécifie l'évaluation d'une disjonction de conditions. Dans ce cas, les réponses sont les noeuds qui vérifient l'une des deux conditions.

Le règle (**Cond:Neg**) capture l'évaluation de la négation d'une condition. Cette évaluation est triviale, elle retourne les noeuds de $\mathcal{S}$ qui ne satisfont pas $cond$.

La règle (**Cond:Exp**) capture l'évaluation d'une condition spécifiée par une expression comparaison. Le résultat de cette évaluation est l'id-set $\mathcal{S}$ restreint aux noeuds pour lesquels la comparaison est vérifiée. L'évaluation des expressions est assurée par la fonction $Eval()$ que nous ne spécifions pas ici.

La règle (**Cond:rel−path**) spécifie l'évaluation d'une condition réduite à un chemin relatif p. Le résultat de cette évaluation est la restriction de $\mathcal{S}$ aux noeuds à partir desquels l'évaluation de p est non vide.

Exemple 8. Soit t le document de la Figure 2.3.
Soit le chemin $\mathrm{P}_5 = \mathrm{self} :: \mathrm{Solar/planet/}\texttt{node}()[\mathrm{bold}]$ qui retourne les noeuds fils de planet ayant un fils bold.
L'évaluation de P_5 sur t retourne $\{\mathbf{i}_8\}$ puisque :
$-$ $PEval(t, \mathrm{self} :: \mathrm{Solar/planet/}\texttt{node}()) = \{\mathbf{i}_6, \mathbf{i}_7, \mathbf{i}_8\}$, et
$-$ $PEval_{\{\mathbf{i}_6\}}(t, \mathrm{bold})$=$PEval_{\{\mathbf{i}_7\}}(t, \mathrm{bold}) = \emptyset$ alors que $PEval_{\{\mathbf{i}_8\}}(t, \mathrm{bold}) = \{\mathbf{i}_{12}\}$. ■

Les règles (**Cond:abs−path**) et (**Cond:abs−path−empty**) spécifient l'évaluation d'une condition réduite à un chemin absolu $/p$. Cette évaluation est triviale : l'id-set $\mathcal{S}$ est retourné si l'évaluation de p à partir de la racine de σ est non vide. Dans le cas contraire, l'évaluation de la condition retourne l'ensemble vide.

2.4 Le langage de requêtes XQuery

Le langage de chemins que nous venons d'introduire est insuffisant pour assurer les fonctionalités d'interrogation des documents XML. Il se limite à sélectionner des noeuds sur la base de

conditions relativement simples alors que l'interrogation requiert de spécifier des conditions complexes et de pouvoir éventuellement construire de nouveaux éléments. C'est la raison pour laquelle l'interrogation des documents XML est assurée par le langage XQuery proposé par le W3C [XQu].

XQuery est un langage fonctionnel qui s'appuie sur XPath pour naviguer dans le document interrogé et sur des expressions pré-définies pour spécifier des traitements complexes. Les expressions *FLWOR* (For, Let, Where, Order by et Return) sont les plus utilisées dans XQuery. La forme générale des requêtes XQuery est donnée ci-dessous.

let *var* **:=** *Exp*
for *var* **in** *Exp*
where *Filtre*
(order by *Order***) ?**
return *Exp*

Chaque ligne correspond à une clause qui porte le nom du mot-clé qui y est exprimé, par exemple la première ligne correspond à la clause **let**. Chaque clause joue un rôle particulier dans la construction du résultat final de la requête. Une requête XQuery commence souvent par une clause **let** qui permet de pré-sélectionner des noeuds en évaluant l'expression Exp ou par une clause **for** qui permet de spécifier un traitement itératif. La clause **where** est utilisée pour filtrer les noeuds pré-sélectionnés par l'une des clause précédentes. La clause **order by** qui est facultative permet de trier, suivant un ordre particulier spécifié par $Order$, les résultats obtenus à partir des clauses précédentes. La clause **return** permet de construire le résultat final.

XQuery permet aussi de définir des variables et de leur associer des résultats intermédiaires. Lorsqu'elles sont utilisées dans d'autres clauses, les variables sont substituées par les résultats auxquels elles ont été associées. Dans XQuery, l'introduction et l'affectation des variables se fait au moyen des clauses **let** et **for**. La première permet d'associer d'un seul coup le résultat l'évaluation de Exp alors que la seconde permet d'itérer sur ce résultat.

L'exemple suivant a pour vocation d'illustrer le rôle de chacune des clauses d'une requête *FLWOR*.

Exemple 9. Considérons la requête suivante qui retourne le diamètre de la planète Mars.

$\boldsymbol{Q_1}$: **let** $\$x$:= $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}$ ■
where $\$x/\mathrm{name}/\mathrm{text}() = \text{"Mars"}$
return $\$x/\mathrm{diameter}$

Cette requête possède les clauses **let**, **where** et **return**. La clause **let** extrait les noeuds planet du document interrogé en utilisant le chemin $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}$ et conserve ces noeuds dans la variable $\$x$. La clause **where** sélectionne à partir des noeuds extraits par le **let** et transmis via la variable $\$x$ les noeuds qui correspondent à la planète Mars. La clause **return** construit le résultat final en retournant les éléments $\mathrm{diameter}$ des noeuds sélectionnés par les clauses précédentes.

XQuery permet de construire des requêtes complexes par composition de requêtes. L'imbrication est un cas particulier de la composition qui consiste à emboîter des requêtes les unes dans les autres et de faire en sorte que le résultat d'une requête soit l'entrée d'une autre requête. Pour les

requêtes *FLWOR*, cela revient à exprimer des requêtes *FLWOR* à la place de l'expression Exp des clauses **for**, **let** et **return**. L'exemple 10 montre une requête composée dont la clause **return** est constituée d'une requête *FLWOR* notée Q_{21}. La construction de requêtes par imbrication est importante puisqu'elle permet d'exprimer des jointures. L'évaluation d'une requête composée passe par l'évaluation de ses sous-requêtes et chaque sous-requête produit des résultats intermédiaires susceptibles d'être utilisés par d'autres sous-requêtes. La définition des variables est l'un des mécanismes utilisés pour transmettre les résultats intermédiaires entre les sous-requêtes. Dans notre exemple, la variable x définie par la clause **let** de la requête principale et contenant la liste des noeuds planet correspondants à Mars [2] est utilisée au niveau de la clause **where** de la requête imbriquée Q_{21} pour sélectionner les noeuds planet associés cette fois à la variable y sur la base de leur noeud $\mathrm{diameter}$.

Exemple 10. Considérons la requête suivante qui retourne la liste des satellites naturels des planètes ayant le même diamètre que Mars.

$\boldsymbol{Q_2}$ **for** $\$x$ in $\mathrm{self :: Solar/planet}$
where $\$x/\mathrm{name/text()} = \text{"Mars"}$
return
Q_{21} **for** $\$y$ in $\mathrm{self :: Solar/planet}$
where $\$y/\mathrm{name/text()}$!="Mars" and $\$y/\mathrm{diameter} = \$\mathrm{x/diameter}$
return $\$y$/satellite

■

L'une des fonctionnalités importantes de XQuery est la possibilité de construire de nouveaux éléments à partir des résultats des requêtes. Cela s'avère utile pour restructurer les réponses d'une requête suivant un format différent de celui du document interrogé.

Exemple 11. Considérons la requête suivante qui construit un nouvel élément $\mathrm{innerPlanets}$ qui devra contenir les planètes internes du système solaire, à savoir : Mercure, Vénus, la Terre, et Mars.

$\boldsymbol{Q_3}$: **let** $\$x$:= $\mathrm{self :: Solar/planet}$ ■
return
<innerPlanets>
Q_{31} {**for** $\$y$ in $\$x$
where $\$y/\mathrm{name/text()} = \text{"Mercury"}$ or $\$y/\mathrm{name/text()} = \text{"Venus"}$ or
$\$y/\mathrm{name/text()} = \text{"Earth"}$ or $\$y/\mathrm{name/text()} = \text{"Mars"}$
return $\$y$}
</innerPlanets>

La requête $\boldsymbol{Q_3}$ construit un élément $\mathrm{innerPlanets}$ dont le contenu est obtenu par évaluation de la requête imbriquée Q_{31}. Il est important d'indiquer que ce contenu est une simple copie du résultat original de Q_{31} et que le document interrogé n'est évidemment pas modifié par l'évaluation de cette requête.

2.4.1 Syntaxe du langage de requêtes

La grammaire du langage de requêtes que nous considérons est donnée dans la Figure 2.9.

2. Cette liste est sensée être un singleton

q	$::=$	$()$	
		$\mid q_1, q_2$	
		$\mid$ <a>q</a>	
		$\mid$ **for** x `in` q_0 **return** q_1	
		$\mid$ **let** x = q_0 **return** q_1	
		$\mid$ **if** q_0 **then** q_1 **else** q_2	
		$\mid Exp$	
Exp	$::=$	$CExp$	
		$\mid Exp_1\ ct\ Exp_2$	$ct \in$ `Log`
$CExp$	$::=$	$AExp$	
		$\mid CExp_1\ cp\ CExp_2$	$cp \in$ `NComp`$\cup$`VComp`
$AExp$	$::=$	$PExp$	
		$\mid AExp_1\ op\ AExp_2$	$op \in$ `AritOp`
$PExp$	$::=$	$st \mid x \mid Path \mid x/Path \mid f(q_0, \ldots, q_n)$	

`Log`$=\{and, or\}$ `NComp`$=\{\ll, \gg, is\}$
`VComp`$=\{eq, ne, lt, gt, le, ge, !=, <, >, \leq, \geq\}$
`AritOp`$=\{+, -, *, div, mod\}$

FIGURE 2.9 – Syntaxe du langage de requêtes

Outre la requête vide $()$ et la composition séquentielle q_1, q_2, cette grammaire permet d'exprimer :

- la construction d'élément <a>q</a> qui permet de construire un nouvel élément dont le contenu est obtenu par évaluation de q et dont la racine est étiquetée a ;
- l'itération **for** x `in` q_0 **return** q_1 qui permet d'évaluer q_1 pour chaque noeud obtenu de l'évaluation de q_0 ;
- le binding **let** x = q_0 **return** q_1 qui permet d'assigner à x le résultat de l'évaluation de q_0 pour être ensuite utilisé dans q_1 ;
- le branchement conditionnel **if** q_0 **then** q_1 **else** q_2 qui permet d'évaluer q_1 ou q_2 en fonction du résultat de l'évaluation de q_0 ;
- les requêtes atomiques générées par le non-terminal $PExp$ et qui consistent en :
  - ◇ les littéraux qui sont ici uniquement des constantes st de type *String*,
  - ◇ les variables notées par x,
  - ◇ les chemins $Path$ qui peuvent être préfixés par une variable x ($x/Path$) dont le rôle est de fournir le contexte d'évaluation de $Path$.
    Le non-terminal $Path$ qui permet donc de spécifier des chemins au sein des requêtes n'est pas défini dans la grammaire. Sa définition se réfère au non-terminal P de la syntaxe des chemins donnée en Figure 2.7 modifiée de sorte à pouvoir utiliser des variables au sein des conditions. La modification à apporter à la grammaire des chemins concerne uniquement la règle de production

$$Atom\ ::=\ f(Exp, \ldots, Exp) \mid p \mid st$$

qui devra être remplacée par la règle

$$Atom \ ::= \ f(Exp, \ldots, Exp) \mid x \mid p \mid x/p \mid st$$

◇ les fonctions notées par f et ayant pour argument un certain nombre de requêtes $q_0, \ldots, q_n$.

- les expressions arithmétiques $AExp_1 \ op \ AExp_2$ construites par combinaison de requêtes atomiques $PExp_i$ en utilisant les opérateurs arithmétiques `AritOp`;
- les expressions de comparaison $CExp_1 \ cp \ CExp_2$ construites par combinaison d'expressions arithmétiques en utilisant les opérateurs de comparaison de noeuds `NComp` et de valeur `VComp`;
- les expressions booléennes $Exp_1 \ ct \ Exp_2$ construites par combinaison d'expressions de comparaison en utilisant les connecteurs logiques and, or (la négation étant exprimée par la fonction `not`() dans XQuery).

Dans la suite $GExp_1 \ gop \ GExp_2$ désigne l'une des trois expressions : $Exp_1 \ ct \ Exp_2$, $CExp_1 \ cp \ CExp_2$ ou $AExp_1 \ op \ AExp_2$.

La requête de l'exemple suivant montre comment des variables sont utilisées dans des chemins.

Exemple 12. Considérons la requête suivante qui retourne le liste des satellites naturels des planètes ayant le même diamètre que Mars.

$\boldsymbol{Q_4}$: **for** $\$x$ in self :: Solar/planet[name/text() = "Mars"]
return
Q_{42} **for** $\$y$ in self :: Solar/planet[name/text() !="Mars" and diameter = $x/diameter]
return $\$y$/satellite

Dans cette requête, la variable x liée au résultat de la requête (chemin)

$$\text{self :: Solar/planet[name/text() = "Mars"]}$$

est utilisée dans la sous-requête Q_{42} pour exprimer la condition de jointure

$$\text{diameter} = \$\text{x/diameter}$$

■

Simplification du langage de requêtes Comme c'est le cas dans certains travaux [BC09, BK09], nous ne formalisons pas toutes les fonctionalités permises par le W3C [XQu] et simplifions certaines expressions XQuery. Un exemple de simplification concerne les expressions *FLWOR* pour lesquelles nous n'utilisons pas la clause **where** puisque sa fonction de filtre peut être exprimée par des prédicats au niveau des chemins de la clause **for** ou **let**. Ainsi, la requête $\boldsymbol{Q_2}$ de l'exemple 10 est équivalente à la requête $\boldsymbol{Q_4}$ de l'exemple 12.

Dans les expressions d'itération **for** x `in` q_0 **return** q_1, de binding **let** x = q_0 **return** q_1 et de branchement conditionnel **if** q_0 **then** q_1 **else** q_2, nous distinguons deux sortes de requêtes :

– la sous-requête q_0 qui permet de pré-sélectionner des noeuds susceptibles d'être utilisés pour la construction du résultat final, elle sera appelée requête *contextuelle*,

– les sous-requêtes q_1 et q_2 permettant de spécifier le résultat global de la requête, elles sont appelées requêtes *résultat*.

Dans la suite de ce manuscrit, nous utilisons $\mathbf{bind}(x, q_0)$ $\mathbf{return}$ q_1 pour désigner **for** x `in` q_0 $\mathbf{return}$ q_1 ou **let** x = q_0 $\mathbf{return}$ q_1.

2.4.2 Sémantique

La sémantique des requêtes définie ci-après s'appuie évidemment sur la sémantique des chemins. Cependant, il existe certaines différences entre l'évaluation des chemins et celle des requêtes que nous soulignons dans ce qui suit :

(i) la sémantique des chemins est ensembliste alors que la sémantique des requêtes utilise des séquences d'éléments. Cela implique que les réponses retournées par l'évaluation des chemins doivent être triées pour pouvoir être exploitées par les requêtes.

(ii) la sémantique des chemins considère uniquement les racines d'éléments ciblés par les expressions de chemins tandis que la sémantique des requêtes extrait, en général, les éléments ciblés par les requêtes.

(iii) l'évaluation des requêtes a pour effet de bord la construction de nouveaux éléments dont la description doit être retournée avec les réponses aux requêtes alors que l'évaluation des chemins ne permet pas de construire de nouveaux éléments.

La notion de variables libres et de variables liées pour les requêtes est définie de la manière suivante.

Définition 7 (Variables libres)**.**
Soit q une requête. L'ensemble des variables libres de q, noté $VarLib(q)$, est défini par induction sur la structure de q comme suit :

$$
\begin{aligned}
VarLib(()) &= \emptyset \\
VarLib(q_1, q_2) &= VarLib(q_1) \cup VarLib(q_2) \\
VarLib(\texttt{<a>}q\texttt{}) &= VarLib(q) \\
VarLib(\mathbf{bind}(x, q_0)\ \mathbf{return}\ q_1) &= VarLib(q_0) \cup (VarLib(q_1) - \{x\}) \\
VarLib(\mathbf{if}\ q_0\ \mathbf{then}\ q_1\ \mathbf{else}\ q_2) &= \bigcup_{i=0}^{2} VarLib(q_i) \\
VarLib(GExp_1\ gop\ GExp_2) &= VarLib(GExp_1) \cup VarLib(GExp_2) \\
VarLib(x) &= \{x\} \\
VarLib(/Path) &= \emptyset \\
VarLib(x/Path) &= \{x\} \\
VarLib(f(q_0, .., q_n)) &= \bigcup_{i=0}^{n} VarLib(q_i)
\end{aligned}
$$

Définition 8 (Variables liées)**.**
Soit q une requête. L'ensemble des variables liées de q, noté $VarDef(q)$, est défini comme suit :

$$
\begin{aligned}
VarDef(()) &= \emptyset \\
VarDef(q_1, q_2) &= VarDef(q_1) \cup VarDef(q_2) \\
VarDef(\texttt{<a>}q\texttt{}) &= VarDef(q) \\
VarDef(\mathbf{bind}(x, q_0)\ \mathbf{return}\ q_1) &= \{x\} \cup VarDef(q_0) \cup VarDef(q_1) \\
VarDef(\mathbf{if}\ q_0\ \mathbf{then}\ q_1\ \mathbf{else}\ q_2) &= \bigcup_{i=0}^{2} VarDef(q_i) \\
VarDef(GExp_1\ gop\ GExp_2) &= VarDef(GExp_1) \cup VarDef(GExp_2) \\
VarDef(x) &= \emptyset \\
VarDef(/Path) &= \emptyset \\
VarDef(x/Path) &= \emptyset \\
VarDef(f(q_0, .., q_n)) &= \bigcup_{i=0}^{n} VarDef(q_i)
\end{aligned}
$$

Exemple 13. Soit la requête q_1 spécifiée par :

for $\$x\ in\ \$z/a$
return $\$x/c$.

Alors $VarLib(q_1) = \{z\}$ et $VarDef(q_1) = \{x\}$. ■

Nous supposons dans la suite que les requêtes sont *bien formées*. Cette notion classique est définie comme suit.

Définition 9 (Requête bien formée)**.**
Une requête q est bien formée si ses variables libres et ses variables liées sont distinctes, i.e $VarDef(q) \cap VarLib(q) = \emptyset$.

Exemple 14. Le requête q_1 de l'exemple 13 est bien formée.
Soit la requête q_2 spécifiée par q', q'' où

- $q' = \mathbf{for}\ \$x\ in\ /doc/a\ \mathbf{return}\ \$x$ et
- $q'' = \mathbf{let}\ \$y := \$x/b\ \mathbf{return}\ \$y$.

Cette requête n'est pas bien formée car la variable x définie dans q' est utilisée dans q''. On a :

- $VarDef(q_2) = VarDef(q') \cup VarDef(q'') = \{x, y\}$,
- $VarLib(q_2) = VarLib(q') \cup VarLib(q'') = \{x\}$ et donc
- $VarDef(q_2) \cap VarLib(q_2) = \{x\}$.

■

Lorsqu'une requête n'est pas bien formée, elle pourra être réécrite par renommage des variables afin de séparer les variables libres et liées. En fait, pour simplifier la définition de la sémantique des requêtes, nous supposons, sans perte de généralité qu'aucune variable n'est réutilisée dans l'expression d'une requête. A nouveau, si une requête ne satisfait pas cette condition, il est facile de la réécrire en renommant ses variables pour que la condition soit satisfaite.

Finalement, nous aurons besoin assez souvent de supposer que les requêtes sont closes au sens suivant.

Définition 10 (Requêtes closes)**.**
Soit q une requête. On dit que q est close si $VarLib(q) = \emptyset$.

La requête q_1 de l'Exemple 13 n'est pas close car la variable z utilisée dans q_1 n'est pas définie dans q_1.

L'évaluation d'une requête non-close nécessite de disposer de valeurs pour ses variables libres. D'où l'utilisation d'un environnement produisant des liaisons variables–valeurs.

Définition 11 (Environnement dynamique)**.**
Un environnement dynamique γ pour l'ensemble des variables de V est une application définie sur V telle que $\gamma(x)$ est un id-seq.

Notation
Dans la suite,

- la liaison $\gamma(x)$=I est aussi simplement notée par x=I,
- l'environnement vide est noté $()$,
- $Var(\gamma)$ désigne l'ensemble des variables, domaine de γ,
- nous notons $dom(\gamma)$ l'ensemble des identifiants [3] image de γ i.e $dom(\gamma) = \bigcup_{x \in Var(\gamma)} \gamma(x)$.

Définition 12 (Extension de l'environnement)**.**
Soit γ un environnement, x une variable et I un id-seq. L'environnement $ext(\gamma, x, \mathrm{I})$ est obtenu par extension de γ avec la nouvelle liaison x=I.

3. Le choix de cette notation est motivée par le fait que $dom(\gamma)$ est un ensemble d'identifiants comme l'est le domaine $dom(\sigma)$ d'un store σ.

La notion d'environnement fermé pour un ensemble de variables est définie de manière classique.

Définition 13 (Environnement fermé)**.**
Soit X un ensemble de variables. On dit qu'un environnement γ est fermé pour X si $X \subseteq Var(\gamma)$.

Nous supposons que l'évaluation d'une requête q s'effectue relativement à un environnement γ fermé pour les variables libres de q, i.e. $VarLib(q) \subseteq Var(\gamma)$.

La sémantique des requêtes que nous définissons s'inspire de celle formalisée par Benedikt et Cheney dans [BC09]. Nous proposons une variante de cette formalisation appropriée à nos besoins.

Le jugement que nous introduisons est donnée par :

$$\sigma, \sigma_\epsilon, d, \gamma \models q \Rightarrow \sigma_q, \mathrm{I}_q \quad (\mathbf{Query})$$

où les composants en entrée à l'évaluation de la requête q sont :

- σ le *store initial* correspondant strictement au store du document fourni pour l'évaluation de q, ce store est invariant ;
- σ_ϵ le *store auxiliaire* contenant des éléments construits pour une autre requête et transmis à l'évaluation de q ;
- d le *domaine des identifiants* que l'évaluation de q doit s'interdire d'utiliser lors d'éventuelles constructions d'éléments ;
- γ un environnement des variables de q tel que $VarLib(q) \subseteq Var(\gamma)$.

Les composants du jugement qui correspondent à ce que retourne l'évaluation de la requête q sont :

- σ_q le store contenant exclusivement les éléments construits par évaluation de q, et
- I_q l'*id-seq des racines des réponses* à q et dont la description peut se trouver soit dans σ soit dans σ_q.

Notation
Nous appellerons le tuple $(\sigma_\epsilon, d, \gamma)$ le contexte dynamique de l'évaluation d'une requête et nous le noterons Υ. Par ailleurs, *dom*(Υ) est défini par *dom*$(\sigma_\epsilon) \cup d \cup$ *dom*(γ).

Afin de pouvoir expliquer la sémantique des requêtes, nous énonçons les propriétés relatives au jugement **(Query)** pour les requêtes avant même de donner les règles de la sémantique.

Propriété 1 (Evaluation des requêtes)**.**
Le jugement $\sigma, \sigma_\epsilon, d, \gamma \models q \Rightarrow \sigma_q, \mathrm{I}_q$ $(\mathbf{Query})$ vérifie les propriétés suivantes :

(i) $\mathrm{I}_q \subseteq$ *dom*$(\sigma) \cup$ *roots*(σ_q),

(ii) *dom*$(\sigma) \cap$ *dom*$(\sigma_q) = \emptyset$, et

(iii) *dom*$(\sigma_q) \cap d = \emptyset$.

Le point (i) énonce que les réponses à la requête q sont décrites soit dans le store initial σ soit dans le store des éléments construits σ_q.

Le point (ii) énonce que les éléments construits n'utilisent pas d'identifiants du store initial.

Le point (iii) énonce que l'évaluation des requêtes n'utilise pas les identifiants interdits pour la construction de nouveaux éléments.

Dans un premier temps, afin de formaliser la sémantique des requêtes dans le cas général, nous considérons que les requêtes peuvent naviguer sur les éléments qu'elles construisent. Dans ce cas, les identifiants mémorisés dans l'environnement des variables γ peuvent appartenir au store initial σ mais aussi au store auxiliaire σ_ϵ. Nous faisons la convention suivante :

$(\dagger - \boldsymbol{const})$ les identifiants de γ appartenant à σ_ϵ ciblent uniquement les racines de σ_ϵ, i.e. $\mathit{dom}(\gamma) \cap \mathit{dom}(\sigma_\epsilon) \subseteq \mathit{roots}(\sigma_\epsilon)$.

Il est nécessaire d'introduire un jugement capturant la copie des réponses à une requête q :

$$\sigma, \sigma_\epsilon, d, \gamma \models q \overset{\mathsf{copie}}{\Longrightarrow} \sigma_c, \mathrm{I}_c \quad (\mathbf{Query - copy})$$

où les composants en entrée sont décrits de la même façon que les composants du jugement **Query** et où les composants résultat sont décrits comme suit :

- σ_c est le store contenant exclusivement la description de la copie des réponses à q ,
- I_c est l'id-seq des racines des copies des réponses à q.

Les propriétés relatives au jugement **(Query-copy)** sont présentées avant de donner les règles sémantiques pour ce jugement.

Propriété 2 (Copie des réponses aux requêtes)**.**
Le jugement $\sigma, \sigma_\epsilon, d, \gamma \models q \overset{\mathsf{copie}}{\Longrightarrow} \sigma_c, \mathrm{I}_c \quad (\mathbf{Query - copy})$ vérifie les propriétés suivantes :

(i) $\mathrm{I}_c = \mathit{roots}(\sigma_c)$,

(ii) $\mathit{dom}(\sigma_c) \cap \mathit{dom}(\sigma) = \emptyset$, et

(iii) $d \cap \mathit{dom}(\sigma_c) = \emptyset$

Le jugement **(Query-copy)** s'appuie à son tour sur un jugement dédié à la copie d'éléments ou de noeuds texte spécifiés par leurs racines. Ce jugement est donné comme suit :

$$\sigma, d, \mathrm{I} \overset{\mathsf{copie}}{\longmapsto} \sigma_c, \mathrm{I}_c \quad (\mathbf{Element - Copy})$$

où, les composants en entrée de ce jugement sont :

- σ le store contenant la description des éléments devant être copiés ;
- d est le domaine des identifiants qu'il est interdit d'utiliser pour la copie ;
- I est l'id-seq des identifiants des racines des éléments à copier.

Les composants en résultat de ce jugement sont, bien évidemment,

- σ_c le store contenant les copies d'éléments de σ et identifiés par I ; et
- I_c l'id-seq permettant d'accéder à ces éléments dans σ_c.

Comme précédemment, nous énonçons les propriétés relatives au jugement **(Element-Copy)** avant de donner les règles sémantiques pour ce jugement.

Propriété 3 (Copie des éléments)**.**
Le jugement $\sigma, d, \mathrm{I} \overset{\mathsf{copie}}{\longmapsto} \sigma_c, \mathrm{I}_c$ $(\mathbf{Element-Copy})$ vérifient les propriétés suivantes :

(i) I_c=*roots*(σ_c),

(ii) *dom*$(\sigma) \cap$ *dom*$(\sigma_c) = \emptyset$, et

(iii) $d \cap$ *dom*$(\sigma_c) = \emptyset$.

Les règles qui capturent la copie d'une forêt sont données dans la Figure 2.10.

$$(\mathbf{Copy{:}Empty}) \quad \frac{}{\sigma, d, () \overset{\mathsf{copie}}{\longmapsto} \varnothing, ()}$$

$$(\mathbf{Copy{:}Seq}) \quad \frac{\sigma, d, \mathbf{i} \overset{\mathsf{copie}}{\longmapsto} \sigma_1, \mathbf{i}_1 \quad \sigma, d \cup \mathit{dom}(\sigma_1), \mathrm{J} \overset{\mathsf{copie}}{\longmapsto} \sigma_\mathrm{J}, \mathrm{J}_c}{\sigma, d, \mathbf{i}{\cdot}\mathrm{J} \overset{\mathsf{copie}}{\longmapsto} \sigma_1{\cdot}\sigma_\mathrm{J}, \mathbf{i}_1{\cdot}\mathrm{J}_c}$$

$$(\mathbf{Copy{:}Elem}) \quad \frac{\sigma(\mathbf{i}){=}a[\mathrm{I}] \quad \sigma, d, \mathrm{I} \overset{\mathsf{copie}}{\longmapsto} \sigma_c, \mathrm{I}_c \quad \mathbf{i}_1 \notin d \cup dom(\sigma) \cup \mathit{dom}(\sigma_c) \quad \sigma_1{=}Store(\mathbf{i}_1, a, \sigma_c)}{\sigma, d, \mathbf{i} \overset{\mathsf{copie}}{\longmapsto} \sigma_1, \mathbf{i}_1}$$

$$(\mathbf{Copy{:}Text}) \quad \frac{\sigma(\mathbf{i}){=}text[st] \quad \mathbf{i}_1 \notin d \cup \mathit{dom}(\sigma) \quad \sigma_1{=}Texte(\mathbf{i}_1, st)}{\sigma, d, \mathbf{i} \overset{\mathsf{copie}}{\longmapsto} \sigma_1, \mathbf{i}_1}$$

FIGURE 2.10 – Sémantique des requêtes : copie de forêts

La règle $(\mathbf{Copy{:}Empty})$ capture le cas terminal.

La règle $(\mathbf{Copy{:}Seq})$ capture la copie d'une forêt donnée par les identifiants de ses racines. Cette règle utilise simplement la structure d'une forêt (concaténation d'un arbre et d'une forêt) tout en s'assurant de transmettre les identifiants alloués lors de la copie d'un arbre à l'opération de copie de la sous-forêt qui suit à travers le composant d.

La règle $(\mathbf{Copy{:}Elem})$ spécifie la copie d'un arbre identifié par sa racine. D'abord la sous-forêt de l'arbre est copiée créant un nouveau store σ_c auquel on va connecter la nouvelle racine $\mathbf{i}_1$ pour obtenir l'arbre copié σ_1. Remarquez que cette règle empêche la réutilisation des identifiants déjà alloués.

La règle $(\mathbf{Copy{:}Text})$ sert à copier un noeud texte.

Nous sommes prêts maintenant à donner la règle spécifiant la copie des réponses d'une requête et les propriétés vérifiées par le jugement **(Element-Copy)**.

$$(\mathbf{Query{:}Copy})\frac{\sigma,\sigma_\epsilon,d,\gamma\models q\Rightarrow\sigma_0,\mathrm{I} \quad \sigma\cdot\sigma_0,d,\mathrm{I}\overset{\mathsf{copie}}{\longmapsto}\sigma_c,\mathrm{I}_c}{\sigma,\sigma_\epsilon,d,\gamma\models q\overset{\mathsf{copie}}{\Longrightarrow}\sigma_c,\mathrm{I}_c}$$

Notation
Comme nous l'avons fait précédemment pour les chemins, nous introduisons des fonctions comme notations alternatives aux jugements. Ces fonctions sont définies dans la Figure 2.11.

$$\begin{array}{lll}
QEval_\Upsilon(t,q)=(\sigma_q,\mathrm{I}_q) & \text{ssi} & \sigma,\sigma_\epsilon,d,\gamma\models q\Rightarrow\sigma_q,\mathrm{I}_q\\
QEvalStore_\Upsilon(t,q)=\sigma_q & \text{ssi} & \sigma,\sigma_\epsilon,d,\gamma\models q\Rightarrow\sigma_q,\mathrm{I}_q\\
QEvalId_\Upsilon(t,q)=\mathrm{I}_q & \text{ssi} & \sigma,\sigma_\epsilon,d,\gamma\models q\Rightarrow\sigma_q,\mathrm{I}_q\\
QCopy_\Upsilon(t,q)=(\sigma_c,\mathrm{I}_c) & \text{ssi} & \sigma,\sigma_\epsilon,d,\gamma\models q\overset{\mathsf{copie}}{\Longrightarrow}\sigma_c,\mathrm{I}_c\\
QCopyStore_\Upsilon(t,q)=\sigma_c & \text{ssi} & \sigma,\sigma_\epsilon,d,\gamma\models q\overset{\mathsf{copie}}{\Longrightarrow}\sigma_c,\mathrm{I}_c\\
QCopyId_\Upsilon(t,q)=\mathrm{I}_c & \text{ssi} & \sigma,\sigma_\epsilon,d,\gamma\models q\overset{\mathsf{copie}}{\Longrightarrow}\sigma_c,\mathrm{I}_c
\end{array}$$
où $\Upsilon=(\sigma_\epsilon,d,\gamma)$

$$\begin{array}{c}
QEval(t,q_1)=QEval_\Upsilon(t,q_1)\\
QEvalStore(t,q_1)=QEvalStore_\Upsilon(t,q_1)\\
QEvalId(t,q_1)=QEvalId_\Upsilon(t,q_1)
\end{array}$$
où $\Upsilon=(\varnothing,\emptyset,())$

FIGURE 2.11 – Sémantique des requêtes : notations alternatives

Nous sommes maintenant prêts à donner les règles capturant la sémantique des requêtes. Comme pour le cas des chemins, les règles sont guidées par la syntaxe. Nous regroupons les règles selon les catégories suivantes :

- les requêtes atomiques qui sont générées par le non-terminal $PExp$ dans notre grammaire,
- les expressions générales (expressions de comparaison et arithmétiques) générées par le non-terminal Exp, et enfin
- requêtes composées i.e la concaténation, les itérations, le binding et les conditionnelles.

2.4.2.1 Sémantique des requêtes atomiques

Par requêtes atomiques, nous entendons la requête vide $()$, les littéraux dénotés par st, les variables, etc. Les règles capturant la sémantique de ces expressions sont données dans la Figure 2.12.

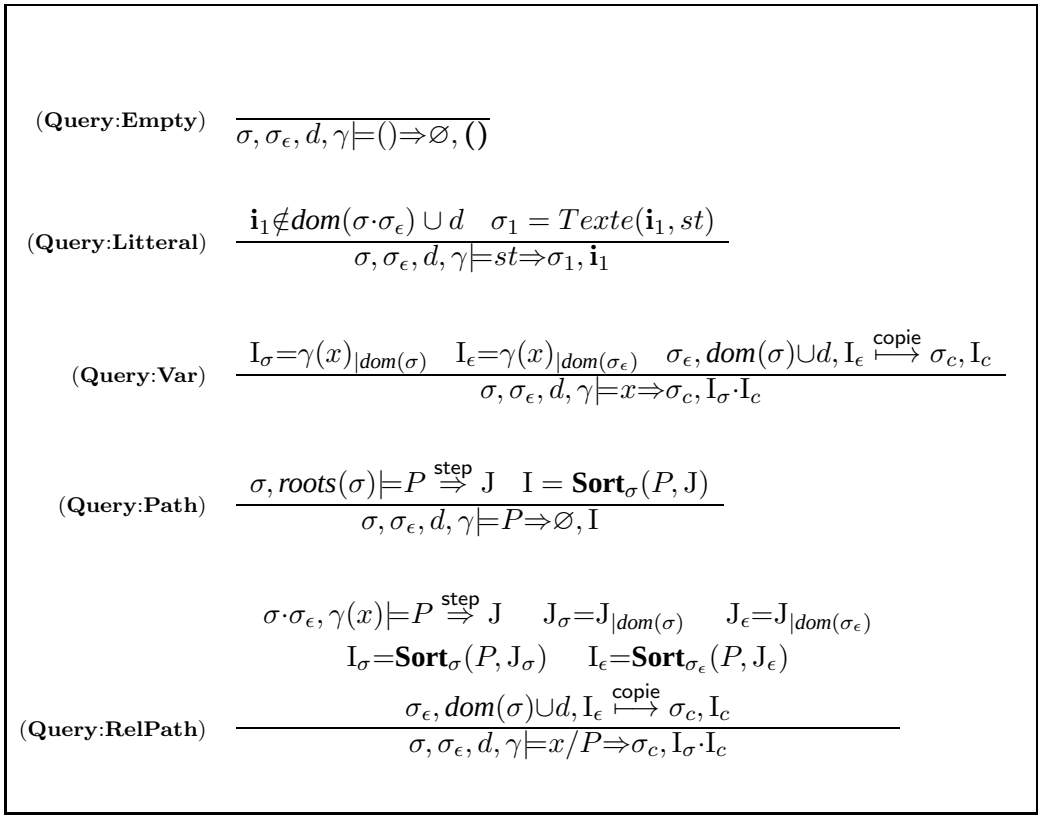


FIGURE 2.12 – Sémantique des requêtes atomiques

Requête vide La règle (**Query**:**Empty**) capture l'évaluation de la requête vide.

Littéral La règle (**Query**:**Litteral**) capture l'évaluation de la requête formée du littéral (constante texte) st. L'évaluation d'un littéral suit le même principe que l'évaluation d'une requête qui construit un nouvel élément : elle retourne un nouveau store contenant un noeud texte dont la valeur est st.

Variables La règle (**Query**:**Var**) capture l'évaluation de la requête x. Intuitivement, cette évaluation retourne les noeuds liés à x qui sont obtenus à partir de l'environnement des variables γ. Ces noeuds peuvent appartenir au store initial σ, (ces noeuds sont donnés par l'id-seq I_σ) auquel cas ils sont retournés directement. Ils peuvent aussi appartenir au store auxiliaire σ_ϵ (ces noeuds sont donnés par l'id-seq I_ϵ). Afin de permettre un traitement uniforme avec celui réservé à l'évaluation des chemins qui sera présentée dans la suite, les éléments enracinés par les noeuds du store auxiliaire sont d'abord copiés. Le store σ_c obtenu de cette copie est retourné avec les nouveaux noeuds I_c comme résultat de l'évaluation de la variable x.

Chemins La règle (**Query**:**Path**) capture l'évaluation des requêtes réduites à un chemin P. Cette évaluation repose, bien évidemment, sur la sémantique des chemins présentée dans la Section 2.3. Etant donnée que cette sémantique est ensembliste, l'id-set J obtenu de l'évaluation du chemins P sur le store σ doit être d'abord trié produisant l'id-seq I qui sera retourné comme résultat de cette

évaluation. L'opération de tri est capturée par la fonction $\mathbf{Sort}_\sigma(P, \mathrm{J})$ que nous ne spécifions pas ici. Une technique permettant le tri des réponses des chemins XPath de manière efficace est proposée dans [FHM+05].

Chemins préfixés de variables La règle ($\mathbf{Query{:}RelPath}$) capture l'évaluation des chemins préfixés d'une variable x. Cette dernière peut être liée à des noeuds du store initial σ ou du store auxiliaire σ_ϵ. Les noeuds du store initial, donnés par l'id-seq I_σ, sont obtenus comme pour le cas des chemins absolus en triant les réponses retournées par évaluation du chemin P. Les noeuds du store auxiliaire I_ϵ nécessitent un traitement particulier dans le but de préserver la pré-condition (†) donnée en page 28. Ce traitement consiste à copier les éléments identifiés par I_ϵ dans un nouveau store σ_c dont les racines J_c sont retournées comme réponse de la requête x/P. L'opération de tri pour les stores auxiliaires est expliquée en détails dans [KHH+09]. Nous ne la spécifions pas ici.

$$(\mathbf{Query{:}Bool}) \quad \frac{\sigma, \sigma_\epsilon, d, \gamma \models Exp_1 \Rightarrow \sigma_1, bool_1 \quad \sigma, \sigma_\epsilon, d \cup \mathit{dom}(\sigma_1), \gamma \models Exp_2 \Rightarrow \sigma_2, bool_2 \quad bool = bool_1\ ct\ bool_2}{\sigma, \sigma_\epsilon, d, \gamma \models Exp_1\ ct\ Exp_2 \Rightarrow \sigma_1 \cdot \sigma_2, bool} \quad ct \in \texttt{Log}$$

$$(\mathbf{Query{:}Comp}) \quad \frac{\sigma, \sigma_\epsilon, d, \gamma \models CExp_1 \Rightarrow \sigma_1, \mathrm{I}_1 \quad \sigma, \sigma_\epsilon, d \cup \mathit{dom}(\sigma_1), \gamma \models CExp_2 \Rightarrow \sigma_2, \mathrm{I}_2 \quad bool = EvalCp(cp, \mathrm{I}_1, \mathrm{I}_2)}{\sigma, \sigma_\epsilon, d, \gamma \models CExp_1\ op\ CExp_2 \Rightarrow \sigma_1 \cdot \sigma_2, bool} \quad op \in \texttt{NComp} \cup \texttt{VComp}$$

$$(\mathbf{Query{:}Arit}) \quad \frac{\sigma, \sigma_\epsilon, d, \gamma \models AExp_1 \Rightarrow \sigma_1, lit_1 \quad \sigma, \sigma_\epsilon, d \cup \mathit{dom}(\sigma_1), \gamma \models AExp_2 \Rightarrow \sigma_2, lit_2 \quad lit = EvalArit(op, lit_1, lit_2)}{\sigma, \sigma_\epsilon, d, \gamma \models AExp_1\ cp\ AExp_2 \Rightarrow \sigma_1 \cdot \sigma_2, lit} \quad cp \in \texttt{AritOp}$$

FIGURE 2.13 – Sémantique des expressions générales

2.4.2.2 Sémantique des expressions

Les règles capturant la sémantique des expressions sont données dans la Figure 2.13. La règle ($\mathbf{Query{:}Bool{-}Comp}$) capture l'évaluation des expressions booléennes et de comparaison. A des fins de simplification de la présentation, nous utilisons le jugement des requêtes pour capturer la sémantique des expressions bien que certaines d'entre elles retournent des booléens ou des littéraux. D'autre part, l'effet d'une comparaison, resp. opération arithmétique est encodé dans une fonction $EvalCp(cp, -, -)$, resp. $EvalArit(op, -, -)$ que nous ne formalisons pas ici. Il est important de souligner que l'évaluation des expressions peut avoir comme effet de bord la construction de nouveaux éléments.

2.4.2.3 Sémantique des requêtes composées

Les règles de la Figure 2.14 spécifient la sémantique des requêtes composées.

$$(\mathbf{Query}{:}\mathbf{Comp}) \quad \frac{\sigma,\sigma_\epsilon,d,\gamma\models q_1\Rightarrow\sigma_1,\mathrm{I}_1 \quad \sigma,\sigma_\epsilon,d\cup\mathit{dom}(\sigma_1),\gamma\models q_2\Rightarrow\sigma_2,\mathrm{I}_2}{\sigma,\sigma_\epsilon,d,\gamma\models q_1,q_2\Rightarrow\sigma_1\cdot\sigma_2,\mathrm{I}_1\cdot\mathrm{I}_2}$$

$$(\mathbf{Query}{:}\mathbf{IfThen}) \quad \frac{\sigma,\sigma_\epsilon,d,\gamma\models q_0\Rightarrow\sigma_0,\mathrm{I}_0 \quad \mathrm{I}_0\neq() \quad \sigma,\sigma_\epsilon,d\cup\mathit{dom}(\sigma_0),\gamma\models q_1\Rightarrow\sigma_1,\mathrm{I}_1}{\sigma,\sigma_\epsilon,d,\gamma\models\mathbf{if}\ q_0\ \mathbf{then}\ q_1\ \mathbf{else}\ q_2\Rightarrow\sigma_1,\mathrm{I}_1}$$

$$(\mathbf{Query}{:}\mathbf{IfElse}) \quad \frac{\sigma,\sigma_\epsilon,d,\gamma\models q_0\Rightarrow\sigma_0,() \quad \sigma,\sigma_\epsilon,d\cup\mathit{dom}(\sigma_0),\gamma\models q_2\Rightarrow\sigma_2,\mathrm{I}_2}{\sigma,\sigma_\epsilon,d,\gamma\models\mathbf{if}\ q_0\ \mathbf{then}\ q_1\ \mathbf{else}\ q_2\Rightarrow\sigma_2,\mathrm{I}_2}$$

$$(\mathbf{Query}{:}\mathbf{Elem}) \quad \frac{\sigma,\sigma_\epsilon,d,\gamma\models q \overset{\text{copie}}{\Longrightarrow} \sigma_0,\mathrm{I}_0 \quad \mathbf{i}_1\notin dom(\sigma\cdot\sigma_\epsilon\cdot\sigma_0)\cup d \quad \sigma_1=Store(\mathbf{i}_1,a,\sigma_0)}{\sigma,\sigma_\epsilon,d,\gamma\models\texttt{<a>}q\texttt{}\Rightarrow\sigma_1,\mathbf{i}_1}$$

$$(\mathbf{Query}{:}\mathbf{Let}) \quad \frac{\sigma,\sigma_\epsilon,d,\gamma\models q_0\Rightarrow\sigma_0,\mathrm{I}_0 \quad \sigma,\sigma_\epsilon\cdot\sigma_0,d,ext(\gamma,x,\mathrm{I}_0)\models q_1\Rightarrow\sigma_1,\mathrm{I}_1}{\sigma,\sigma_\epsilon,\gamma,d\models\mathbf{let}\ x\ \texttt{=}\ q_0\ \mathbf{return}\ q_1\Rightarrow\sigma_1,\mathrm{I}_1}$$

$$(\mathbf{Query}{:}\mathbf{For}) \quad \frac{\sigma,\sigma_\epsilon,d,\gamma\models q_0\Rightarrow\sigma_0,\mathrm{I}_0 \quad \sigma,\sigma_\epsilon\cdot\sigma_0,d,\gamma,x\in\mathrm{I}_0\models^\star q_1\Rightarrow\sigma_1,\mathrm{I}_1}{\sigma,\sigma_\epsilon,d,\gamma\models\mathbf{for}\ x\ \texttt{in}\ q_0\ \mathbf{return}\ q_1\Rightarrow\sigma_1,\mathrm{I}_1}$$

$$(\mathbf{Query}{:}\mathbf{For}{-}\mathbf{end}) \quad \frac{}{\sigma,\sigma_\epsilon,d,\gamma,x\in()\models^\star q\Rightarrow\varnothing,()}$$

$$(\mathbf{Query}{:}\mathbf{For}{-}\mathbf{iter}) \quad \frac{\sigma,\sigma_\epsilon,d,ext(\gamma,x,\mathbf{i}_0)\models q\Rightarrow\sigma_1,\mathrm{I}_1 \quad \sigma,\sigma_\epsilon,d\cup\mathit{dom}(\sigma_1),\gamma,x\in\mathrm{I}_0\models^\star q\Rightarrow\sigma_2,\mathrm{I}_2}{\sigma,\sigma_\epsilon,d,\gamma,x\in\mathbf{i}_0\cdot\mathrm{I}_0\models^\star q\Rightarrow\sigma_1\cdot\sigma_2,\mathrm{I}_1\cdot\mathrm{I}_2}$$

FIGURE 2.14 – Sémantique des requêtes composées

Composition séquentielle L'évaluation de la composition séquentielle de deux requêtes q_1 et q_2 est capturée par la règle (**Query**:**Comp**). Cette règle spécifie que le résultat d'une telle évaluation est obtenu par évaluation de chaque requête séparément : le store retourné est donné par la composition des stores σ_1 et σ_2 décrivant les éléments construits par chaque requête, et l'id-seq des réponses est donné par la concaténation des réponses I_1 I_2 retournées par chaque requête. Rappelons que les requêtes sont bien formées et donc que les variables définies dans q_1 ne peuvent pas être utilisées dans q_2. Par conséquent, q_2 ne peut pas naviguer dans les éléments construits par q_1. De ce fait, il n'est pas utile de transmettre le store σ_1 à l'évaluation de q_2. Toutefois, il est nécessaire de garantir la disjonction des domaines de σ_1 et σ_2. Pour y parvenir, les identifiants utilisés dans σ_1 sont spécifiés comme identifiants interdits dans l'évaluation de q_2.

Le branchement conditionnel procède de la même manière de ce point de vue.

Branchement conditionnel L'évaluation de la requête $\mathbf{if}\ q_0\ \mathbf{then}\ q_1\ \mathbf{else}\ q_2$ est capturée par deux règles : ($\mathbf{Query}$:$\mathbf{IfThen}$) et ($\mathbf{Query}$:$\mathbf{IfElse}$). La première règle spécifie le cas où le résultat de la requête q_0 n'est pas vide. Dans ce cas, le résultat global de la requête est celui obtenu par évaluation de q_1. La seconde règle capture le cas dual, le résultat de q_0 est vide et le résultat global est déterminé par le résultat de q_2. Notons que les identifiants de σ_0 sont transmis pour l'évaluation des requêtes q_1 ou q_2 comme étant des identifiants interdits.

Construction d'éléments La règle ($\mathbf{Query}$:$\mathbf{Elem}$) spécifie l'évaluation de la requête qui construit un élément a à partir du résultat d'une requête q. L'évaluation d'une construction nécessite de copier le résultat de q dans un nouveau store σ_0 et de rattacher une nouvelle racine $\mathbf{i}_1$ au store σ_0 obtenu de cette copie. Le store σ_1 ainsi obtenu est retourné comme résultat de l'évaluation de la requête $<a>q$.

Binding La règle ($\mathbf{Query}$:$\mathbf{Let}$) capture l'évaluation de la requête $\mathbf{let}\ x = q_0\ \mathbf{return}\ q_1$. Elle procède en évaluant d'abord la requête contextuelle q_0 qui enrichit le contexte dynamique en entrée par :

- σ_0 le store décrivant les éléments construits lors de l'évaluation de q_0 et
- x=I_0 la nouvelle liaison pour la variable.

La requête résultat q_1 est ensuite évaluée à partir du nouveau contexte dynamique produisant le résultat final de la requête.

Itération La règle ($\mathbf{Query}$:$\mathbf{For}$) capture l'évaluation de la requête itérative $\mathbf{for}\ x\ \mathtt{in}\ q_0\ \mathbf{return}\ q_1$. Cette évaluation suit les mêmes étapes que l'évaluation du **let** à la différence que la requête résultat est évaluée itérativement pour chaque noeud retourné par évaluation de la requête contextuelle q_0.

L'itération est capturée par un nouveau jugement (indiqué par l'astérisque au niveau du symbole $\models$) et deux règles ($\mathbf{Query}$:$\mathbf{For-end}$) et ($\mathbf{Query}$:$\mathbf{For-iter}$). La première exprime la condition d'arrêt de l'itération. La seconde spécifie le pas de l'itération et le contexte dynamique utilisé dans chaque itération. Elle capture aussi le fait que le résultat de l'évaluation de la requête **for** est obtenu en concaténant dans l'ordre les résultats obtenus à chaque pas.

Nous illustrons les aspects les plus importants de la sémantique des requêtes à l'aide de l'exemple ci-dessous.

Exemple 15. Considérons la requête ci-dessous.

$\boldsymbol{Q_5}$: **let** \$$x$:= $self$::$Solar/planet$
return
<innerPlanets>
Q_{51} Q_{511} **{for** \$$y$ in \$$x$[name/text() = "Mercury" or name/text() = "Venus" or name/text()="Earth" or name/text() = "Mars"]
return \$$y$**}**
</innerPlanets>,
<outerPlanets>
Q_{52} **{for** \$$z$ in \$$x$[name/text() = "Uranus" or name/text() = "Jupiter" or name/text() = "Neptune" or name/text() = "Pluto"]
return \$$z$**}**
</outerPlanets>

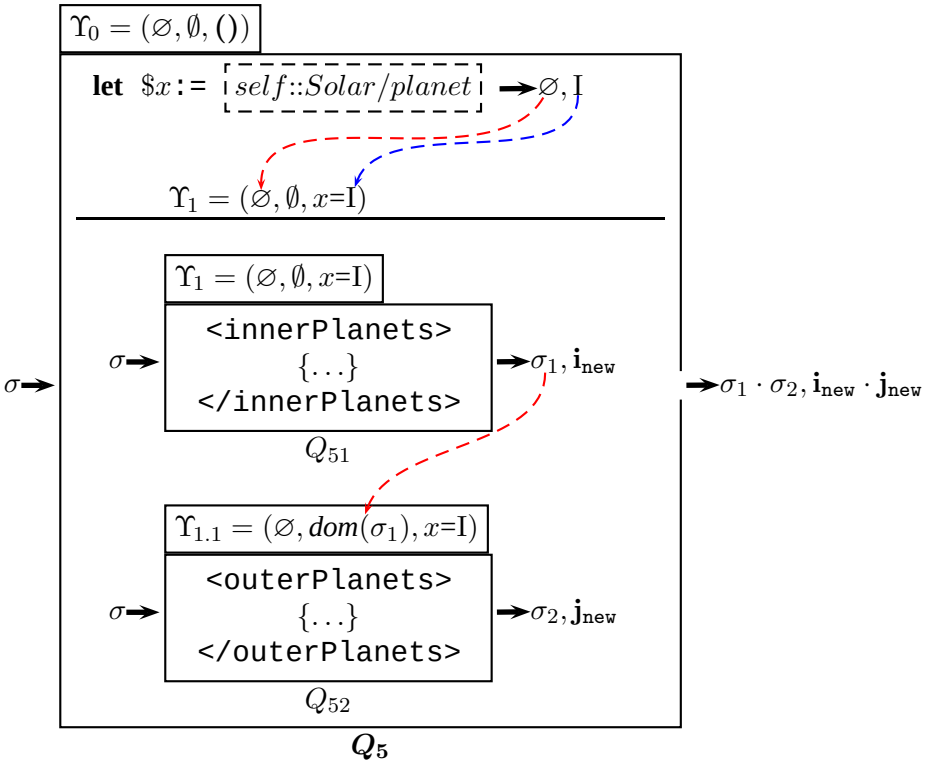


FIGURE 2.15 – Illustration de l'évaluation de la requête $\boldsymbol{Q_5}$

Cette requête est de la forme **let** x = q_0 **return** q_1 où q_0 est réduite au chemin $/\mathrm{Solar}/\mathrm{planet}$ et où q_1 correspond à la composition séquentielle Q_{51}, Q_{52}. Les étapes de l'évaluation de $\boldsymbol{Q_5}$ sont illustrées dans la Figure 2.15 où la valeur de chaque ingrédient du jugement de la sémantique des requêtes est indiquée. Cette requête s'évalue à partir du contexte dynamique vide Υ_0 et suit deux étapes :

- Evaluation de la requête contextuelle q_0 à partir du contexte dynamique initial Υ_0 qui retourne le store vide et la séquence I correspondant aux réponses de $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}$. Cette évaluation permet d'étendre le contexte dynamique Υ_0 qui va devenir Υ_1 de la manière suivante : l'environnement des variables de Υ_0 (qui est vide) est enrichi par la liaison x=I.
  Le nouveau contexte dynamique Υ_1 est fourni à l'évaluation de la requête résultat q_1.

- Evaluation de la requête résultat $q_{\texttt{res}}$ qui consiste en deux étapes :
  - Evaluation de Q_{51} à partir de Υ_1 qui nécessite, à son tour d'évaluer la requête interne Q_{511}. Cette requête se sert de la variable x et repose donc sur l'environnement des variables de Υ_1.
    Le résultat de Q_{51} est un nouvel élément décrit par le store σ_1 et dont l'identifiant de la racine est $\mathbf{i}_{\texttt{new}}$. A nouveau, le contexte dynamique est étendu. Cette fois, Υ_1 devient $\Upsilon_{1.1}$ en plaçant dans le domaine des identifiants interdits de ce dernier l'id-set $\mathit{dom}(\sigma_1)$.
  - Evaluation de Q_{52} à partir de $\Upsilon_{1.1}$. Cette requête construit, elle aussi, un nouvel élément décrit par σ_2 et dont la racine est identifiée par $\mathbf{j}_{\texttt{new}}$. Grâce à l'utilisation du domaine des identifiants interdits, les identifiants de σ_1 ne sont pas réutilisés dans σ_2.

■

Navigation sur les éléments construits Dans la suite, pour simplifier le développement formel, nous supposons que les requêtes ne sont pas autorisées à naviguer sur des éléments construits. Cela signifie que les requêtes contextuelles des expressions **for** x `in` q_0 **return** q_1 ou **let** x = q_0 **return** q_1 doivent impérativement retourner des éléments du store initial.

En pratique, les requêtes qui construisent des éléments pour ensuite les interroger sont rares. La construction d'élément est généralement utilisée pour restructurer le contenu des réponses ou pour construire des éléments devant être insérés par des mises à jour. D'autre part, des techniques comme celle de [PHM+05] proposent de réécrire ces requêtes en des requêtes équivalentes n'utilisant pas de construction. L'hypothèse que nous faisons reste donc raisonnable. L'implication de cette hypothèse sur les propriétés que nous avons énoncées concerne la pré-condition $(\dagger - \boldsymbol{const})$ donnée en page 28 qui n'est plus vérifiée : désormais toute liaison de γ cible exclusivement des noeuds du store initial σ i.e $\mathit{dom}(\gamma) \subseteq \mathit{dom}(\sigma)$.

2.5 Le langage de mise à jour XQuery Update

Nous venons de présenter en détails le langage XQuery qui permet l'interrogation de documents XML au sens classique du terme. En particulier, une requête XML produit un résultat sous la forme d'éléments construits et préserve le document en entrée (la requête n'a pas d'effets de bord sur le document interrogé). Dans cette section, nous présentons le langage de mises à jour XQuery Update, langage proposé par le W3C [XUP].

XQuery Update étend le langage XQuery avec des expressions permettant de spécifier la modification, l'insertion, la suppression ou le remplacement des éléments d'un document. Il s'appuie sur XQuery à la fois pour construire les éléments devant être insérés (par les mises à jour d'insertion ou de remplacement) et pour identifier les noeuds ciblés par les mises à jour.

L'un des choix importants retenus lors de la conception de XQuery Update consiste à dissocier la phase d'interrogation/consultation du document de la phase de mise à jour proprement dite. Cette séparation aide à la définition d'une sémantique déterministe et facilite l'analyse de conflits. D'après cette sémantique, l'effet d'une mise à jour u est obtenu en deux étapes :

- étape de *pré-évaluation* de u qui consiste à évaluer la requête exprimée dans u relativement à un document t afin de localiser les noeuds de t devant subir une mise à jour et/ou construire les éléments devant être insérés. Cette étape produit une séquence de mises à jour primitives appelée PUL (Pending Update List) qui sont les opérations concrètes devant être effectuées sur t.
- étape d'*application* de la PUL qui consiste à appliquer les mises à jour primitives obtenues par l'étape de pré-évaluation afin de produire $u(t)$ la mise à jour du document t. Cette étape suppose qu'un certain nombre de tests ont été effectués pour vérifier que la PUL est conforme aux spécifications du W3C [XUP].

2.5.1 Syntaxe du langage de mises à jour

Le langage de mises à jour que nous considérons est celui défini par Benedikt et al. [BC09] dont la syntaxe est donnée en Figure 2.16.

$$
\begin{array}{lll}
u & ::= & () \\
 & & \mid u_1, u_2 \\
 & & \mid \textbf{insert}\ q_{\texttt{src}}\ \delta\ q_{\texttt{cib}} \qquad \delta \in \{\leftarrow, \rightarrow, \downarrow, \swarrow, \searrow\} \\
 & & \mid \textbf{replace}\ q_{\texttt{src}}\ \texttt{with}\ q_{\texttt{cib}} \\
 & & \mid \textbf{delete}\ q_{\texttt{cib}} \\
 & & \mid \textbf{rename}\ q_{\texttt{cib}}\ \texttt{as}\ a \\
 & & \mid \textbf{let}\ x = q_0\ \textbf{return}\ u_1 \\
 & & \mid \textbf{for}\ x\ \texttt{in}\ q_0\ \textbf{return}\ u_1 \\
 & & \mid \textbf{if}\ q_0\ \textbf{then}\ u_1\ \textbf{else}\ u_2
\end{array}
$$

FIGURE 2.16 – Syntaxe du langage de mises à jour.

Hormis la mise à jour vide $()$ et la composition séquentielle de mises à jour u_1, u_2, cette syntaxe permet de générer :

- les mises à jour atomiques :
  - ◇ **insert** $q_{\texttt{src}}$ δ $q_{\texttt{cib}}$ qui permet d'insérer une forêt dont les racines sont spécifiées par la requête source $q_{\texttt{src}}$ avant ($\leftarrow$), après ($\rightarrow$), sous ($\downarrow$), en premier fils ($\swarrow$), en dernier fils ($\searrow$) d'un emplacement spécifié par la requête cible $q_{\texttt{cib}}$ et la direction δ ; Les directions
  - ◇ **replace** $q_{\texttt{src}}$ `with` $q_{\texttt{cib}}$ qui permet de remplacer des éléments dont les racines sont spécifiées par $q_{\texttt{cib}}$ par une forêt spécifiée par $q_{\texttt{src}}$;
  - ◇ **delete** $q_{\texttt{cib}}$ qui supprime les éléments dont les racines sont spécifiées par $q_{\texttt{cib}}$;
  - ◇ **rename** $q_{\texttt{cib}}$ `as` a qui permet de renommer l'étiquette d'un noeud spécifié par $q_{\texttt{cib}}$.
- les mises à jour composées qui permettent de construire des expressions de mises à jour par imbrication des mises à jour atomiques comme c'était le cas des requêtes complexes, ces expressions sont :

- ◇ **let** x := q_0 **return** u_1 qui permet de pré-évaluer la mise à jour u à partir du résultat intermédiaire obtenu de l'évaluation de la requête environnement q_0 ;
- ◇ **for** x `in` q_0 **return** u_1 qui permet de pré-évaluer la mise à jour u pour chaque noeud retourné par l'évaluation de q_0 ;
- ◇ **if** q_0 **then** u_1 **else** u_2 qui permet le branchement conditionnel vers la pré-évaluation de u_1 ou u_2 en fonction du résultat de l'évaluation de q_0.

Notation
Nous notons l'ensemble des mises à jour atomiques par $AtomUp$ et l'ensemble des mises à jour composées par $CompUp$.

Comme pour les requêtes composées, nous distinguons dans les mises à jour composées entre la partie *contextuelle* $q_{\texttt{ctxt}}$ et la partie *action* u_1 ou u_2. La partie contextuelle (la requête $q_{\texttt{ctxt}}$) vise à construire le contexte dynamique par rapport auquel la partie action est évaluée. L'évaluation de la partie action permet de générer les mises à jour primitives.

Exemple 16. Considérons la mise à jour suivante qui permet d'insérer un élément satellite à l'intérieur des noeuds planet qui n'en ont pas.

u_1 : **for** \$$x$ in self :: Solar/planet[not(satellite)]
return insert <satellite><dist>empty</dist></satellite>
as-last into \$$x$

La partie contextuelle de cette mise à jour est donnée par le chemins :
self :: Solar/planet[not(satellite)].
La partie action est donnée par l'expression :
insert <satellite><dist>empty</dist></satellite>**as−last into** \$x
qui est de la forme **insert** $q_{\texttt{src}}$ δ $q_{\texttt{cib}}$ où la requête source est donnée par :
<satellite>...</satellite>
qui construit un nouvel élément satellite et où la partie cible est donnée par la variable \$$x$ qui se réfère aux noeuds planet spécifiés par le chemin self :: Solar/planet[not(satellite)]. ■

Mises à jour primitives Comme évoqué plus haut, l'effet d'une mise à jour u sur un document t n'est visible qu'après application des mises à jour primitives pré-évaluées. Les mises à jour primitives sont notées par μ et ont la forme générale suivante :

$$\mu \quad ::= \quad \text{ins}(\text{I}_{\texttt{src}}, \delta, \mathbf{i}_{\texttt{cib}}) \mid \text{repl}(\mathbf{i}_{\texttt{cib}}, \text{I}_{\texttt{src}}) \mid \text{del}(\mathbf{i}_{\texttt{cib}}) \mid \text{ren}(\mathbf{i}_{\texttt{cib}}, a)$$

où, $\mathbf{i}_{\texttt{cib}}$ est un identifiant et $\text{I}_{\texttt{src}}$ un id-seq. On manipule des listes de mises à jour notées par ω et appelées dans la suite PUL.

À chaque mise à jour atomique correspond une mise à jour primitive qui détermine $\mathbf{i}_{\texttt{cib}}$ le noeud cible de l'opération de mise à jour et éventuellement $\text{I}_{\texttt{src}}$ l'id-seq des racines des éléments sources de la mise à jour, çàd, les éléments devant être insérés par l'opération de mise à jour.

La Table 2.1 anticipe la présentation de la sémantique des mises à jour primitives.

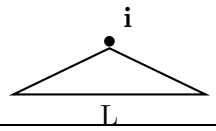
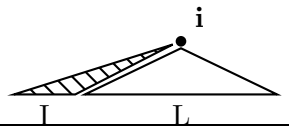
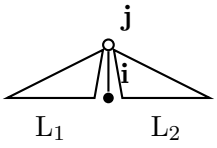
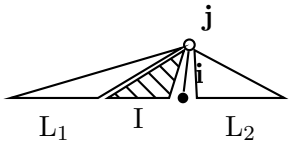
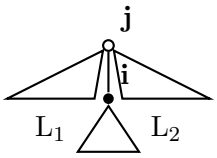
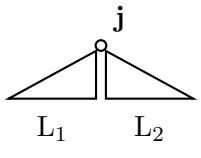
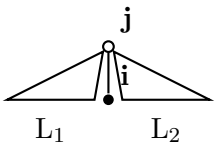
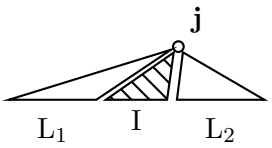
mise à jour μ	**(Avant) sous-arbre de σ**	**(Après) sous-arbre σ_μ**
ins(I, ↙, **i**)		
ins(I, ←, **i**)		
del(**i**)		
repl(**i**, I)		

TABLE 2.1 – Illustration de l'effet de mises à jour atomiques

Notation
Nous introduisons deux fonctions $Source(\omega)$ et $Cible(\omega)$ permettant d'extraire pour une PUL ω les identifiants sources respectivement identifiants cibles de ses mises à jour primitives. Ces fonctions retournent une id-seq, elles sont définies à partir de leurs cas de base comme suit :

- $Source(\mu)$=$\mathrm{I}_{\mathtt{src}}$ si $\mu \in \{\mathrm{ins}(\mathrm{I}_{\mathtt{src}}, \delta, \mathbf{i}_{\mathtt{cib}}), \mathrm{repl}(\mathrm{I}_{\mathtt{src}}, \mathbf{i}_{\mathtt{cib}})\}$ et $Source(\mu)$=() sinon ;
- $Cible(\mu)$=$\{\mathbf{i}_{\mathtt{cib}}\}$ pour toute mise à jour primitive μ.

Le domaine d'une PUL ω, noté *dom*(ω) est défini par $Cible(\omega) \cup Source(\omega)$.

Nous sommes maintenant prêts à donner la définition de la sémantique du langage de mises à jour considéré.

Les notions ci-dessous, introduites pour les requêtes, sont naturellement étendues aux mises à jour :

- variables libres, liées d'une mise à jour
- mise à jour bien formée, et
- environnement fermé pour (les varaibles libres d') une mise à jour.

Pour alléger la présentation, nous ne procédons pas à la définition formelle de ces notions pour lesquelles nous gardons les mêmes notations.

2.5.2 Sémantique du langage de mises à jour

La sémantique du langage de mises à jour a été formalisée à l'aide de règles utilisant des jugements semblables à ceux des requêtes.

Nous introduisons d'abord le jugement principal capturant l'effet global d'une mise à jour u sur un store σ.

$$\sigma \models u \rightsquigarrow \sigma_u \quad \textbf{(Jug-Up)}$$

qui se lit l'évaluation de la mise à jour u sur le document σ produit le document σ_u.

La sémantique intermédiaire des mises à jour est spécifiée par le jugement suivant capturant la pré-évaluation d'une mise à jour u sur un store σ.

$$\sigma, \sigma_\epsilon, d, \gamma \models u \Rightarrow \sigma_\omega, \omega \quad \textbf{(Jug-Gen-PUL)}$$

où, les composants en entrée à la pré-évaluation de u sont :

- σ le store initial sur lequel u est pré-évaluée,
- σ_ϵ le store auxiliaire contenant la description des éléments construits et susceptibles d'être utilisés lors de la pré-évaluation de u,
- d le domaine des identifiants interdits (que la pré-évaluation de u s'empêche d'utiliser lors d'éventuelles créations d'éléments et noeuds texte), et
- γ l'environnement servant à garder traces des liaisons des variables vers les id-seqs.

Les composants retournés par la pré-évaluation de u sont :

- σ_ω le store des éléments construits lors de la pré-évaluation de u. Il est important de souligner que σ_ω contient uniquement les descriptions des éléments sources devant être insérés par u. La description des éléments construits comme effet de bord de l'évaluation de requêtes exprimées dans u n'est pas retournée dans σ_ω.
- ω la liste des mises à jour primitives générées pour u.

Comme pour les requêtes, nous appellerons le tuple $(\sigma_\epsilon, d, \gamma)$ contexte dynamique de la pré-évaluation des mises à jour et nous le noterons Υ.

Donc le jugement **(Jug-Gen-PUL)** se lit : la pré-évaluation de u sur le store σ relativement au contexte dynamique Υ produit le store auxiliaire σ_ω et la PUL ω.

Rappelons que $t{=}(\sigma, r_t)$. Nous introduisons le jugement suivant pour capturer la pré-évaluation des mises à jour de la forme **for** x `in` q_0 **`return`** u_1. Ce jugement permet d'itérer la pré-évaluation de u autant de fois qu'il y a d'éléments dans l'id-seq I :

$$\sigma, \sigma_\epsilon, d, \gamma, x{\in}\mathrm{I} \models^\star u \Rightarrow \sigma_c, \omega \quad \textbf{(Jug-Iter-PUL)}$$

Notation
Nous introduisons les fonctions suivantes comme notations alternatives aux jugements définissant la sémantique des mises à jour.

$PUL_\Upsilon(t,u)=(\sigma_\omega,\omega)$	ssi	$\sigma,\sigma_\epsilon,d,\gamma\models u \Rightarrow \sigma_c,\omega$
$PULStore_\Upsilon(t,u)=\sigma_\omega$	ssi	$\sigma,\sigma_\epsilon,d,\gamma\models u \Rightarrow \sigma_\omega,\omega$
$PULAtomic_\Upsilon(t,u)=\omega$	ssi	$\sigma,\sigma_\epsilon,d,\gamma\models u \Rightarrow \sigma_\omega,\omega$
$\text{PULIter}_{\Upsilon,x}(\mathrm{I},t,u)=(\sigma_c,\omega)$	ssi	$\sigma,\sigma_\epsilon,d,\gamma,x{\in}\mathrm{I}\models^\star u \Rightarrow \sigma_c,\omega$
$\text{PUL}^\star_{\Upsilon,x}(\mathbf{i},t,u)=(\sigma_c,\omega)$	ssi	$\sigma,\sigma_\epsilon,d,ext(\gamma,x,\mathbf{i})\models u \Rightarrow \sigma_c,\omega$
où $\Upsilon=(\sigma_\epsilon,d,\gamma)$		

$$PUL(t,u)=PUL_\Upsilon(t,u)$$
$$PULStore(t,u)=PULStore_\Upsilon(t,u)$$
$$PULAtomic(t,u)=PULAtomic_\Upsilon(t,u)$$
où $\Upsilon=(\varnothing,\emptyset,())$

Nous sommes maintenant prêts à donner les règles spécifiant la pré-évaluation des mises à jour. Ces règles sont guidées par la syntaxe donnée en Figure 2.16 et regroupées en fonction de leur catégorie.

2.5.2.1 Pré-évaluation des mises à jour atomiques

Les règles de la Figure 2.17 capturent la sémantique des mises à jour atomiques.

Avant de les commenter, il est important de revenir sur les hypothèses du W3C [XUP] que nous adoptons. Ces hypothèses concernent l'évaluation de la requête cible $q_{\texttt{cib}}$ qui est supposée retourner au plus un seul noeud, d'où l'hypothèse $|\mathrm{I}_{\texttt{cib}}| \leq 1$ dans les règles ci-dessus. De plus, ce noeud doit être un noeud du document initial σ.

et ce noeud doit appartenir au document à mettre à jour (les noeuds construits ne peuvent être cibles d'une mise à jour).

Insert La règle (**Insert**) traite l'insertion des noeuds spécifiés par la requête source $q_{\texttt{src}}$ dans un emplacement spécifié par la requête cible $q_{\texttt{cib}}$ suivant une direction δ. La pré-évaluation de cette mise à jour nécéssite de copier les réponses de $q_{\texttt{src}}$ dans de nouveaux éléments identifiés par $\mathrm{I}_{\texttt{src}}$ et décrits dans $\sigma_{\texttt{src}}$, et d'évaluer $q_{\texttt{cib}}$ produisant l'id-seq $\mathrm{I}_{\texttt{cib}}$. Le store $\sigma_{\texttt{cib}}$ produit lors de l'évaluation de $q_{\texttt{cib}}$ n'est pas inclus dans le résultat final parce que $q_{\texttt{cib}}$ sert à spécifier un emplacement de mise à jour (l'élément construit par $q_{\texttt{cib}}$ n'est pas utile pour la mise à jour). La pré-évaluation de cette mise à jour retourne le store $\sigma_{\texttt{src}}$ et la mise à jour primitive $\text{ins}(\mathrm{I}_{\texttt{src}},\delta,\mathrm{I}_{\texttt{cib}})$.

Replace La règle (**Replace**) traite le cas où le noeud spécifié par la requête cible $q_{\texttt{cib}}$ est remplacé par les noeuds spécifiés par la requête source $q_{\texttt{src}}$. Cette règle est similaire à la règle (**Insert**) : les réponses de $q_{\texttt{src}}$ sont copiées dans de nouveaux éléments et $q_{\texttt{cib}}$ est évaluée pour capturer l'emplacement du remplacement. La pré-évaluation de de cette mise à jour retourne $\sigma_{\texttt{src}}$ et la mise à jour primitive $\text{repl}(\mathrm{I}_{\texttt{cib}},\mathrm{I}_{\texttt{src}})$

Pour les règles ci-dessous, $\mathbf{Cond}_{\mathtt{cib}}$ est un booléen dont la valeur est vrai si $\mathrm{I}_{\mathtt{cib}} = \{\mathbf{i}_{\mathtt{cib}}\}$ et $\mathbf{i}_{\mathtt{cib}} \in \mathit{dom}(\sigma)$ ou $\mathrm{I}_{\mathtt{cib}} = \emptyset$

$$(\mathbf{Up{:}insert}) \quad \frac{\sigma, \sigma_\epsilon, d, \gamma \models q_{\mathtt{src}} \overset{\text{copie}}{\Longrightarrow} \sigma_{\mathtt{src}}, \mathrm{I}_{\mathtt{src}} \quad \sigma, \sigma_\epsilon, d, \gamma \models q_{\mathtt{cib}} \Rightarrow \sigma_{\mathtt{cib}}, \mathrm{I}_{\mathtt{cib}} \quad \mathbf{Cond}_{\mathtt{cib}}}{\sigma, \sigma_\epsilon, d, \gamma \models \mathbf{insert}\ q_{\mathtt{src}}\ \delta\ q_{\mathtt{cib}} \Rightarrow \sigma_{\mathtt{src}}, \mathrm{ins}(\mathrm{I}_{\mathtt{src}}, \delta, \mathrm{I}_{\mathtt{cib}})}$$

$$(\mathbf{Up{:}replace}) \quad \frac{\sigma, \sigma_\epsilon, d, \gamma \models q_{\mathtt{src}} \overset{\text{copie}}{\Longrightarrow} \sigma_{\mathtt{src}}, \mathrm{I}_{\mathtt{src}} \quad \sigma, \sigma_\epsilon, d, \gamma \models q_{\mathtt{cib}} \Rightarrow \sigma_{\mathtt{cib}}, \mathrm{I}_{\mathtt{cib}} \quad \mathbf{Cond}_{\mathtt{cib}}}{\sigma, \sigma_\epsilon, d, \gamma \models \mathbf{replace}\ q_{\mathtt{cib}}\ \mathtt{with}\ q_{\mathtt{src}} \Rightarrow \sigma_{\mathtt{src}}, \mathrm{repl}(\mathrm{I}_{\mathtt{cib}}, \mathrm{I}_{\mathtt{src}})}$$

$$(\mathbf{Up{:}delete}) \quad \frac{\sigma, \sigma_\epsilon, d, \gamma \models q_{\mathtt{cib}} \Rightarrow \sigma_{\mathtt{cib}}, \mathrm{I}_{\mathtt{cib}} \quad \mathbf{Cond}_{\mathtt{cib}}}{\sigma, \sigma_\epsilon, d, \gamma \models \mathbf{delete}\ q_{\mathtt{cib}} \Rightarrow \varnothing, \mathrm{del}(\mathrm{I}_{\mathtt{cib}})}$$

$$(\mathbf{Up{:}rename}) \quad \frac{\sigma, \sigma_\epsilon, d, \gamma \models q_{\mathtt{cib}} \Rightarrow \sigma_{\mathtt{cib}}, \mathrm{I}_{\mathtt{cib}} \quad \mathbf{Cond}_{\mathtt{cib}}}{\sigma, \sigma_\epsilon, d, \gamma \models \mathbf{rename}\ q_{\mathtt{cib}}\ \mathtt{as}\ a \Rightarrow \varnothing, \mathrm{ren}(a, \mathrm{I}_{\mathtt{cib}})}$$

FIGURE 2.17 – Sémantique des mises à jour atomiques

Delete La règle (**Delete**) traite la suppression de l'élément spécifié par la requête cible $q_{\mathtt{cib}}$ avec les mêmes hypothèses que précédemment, et donc la pré-évaluation de cette mise à jour atomique retourne la mise à jour primitive $\mathrm{del}(\mathrm{I}_{\mathtt{cib}})$ et le store vide $\varnothing$.

Rename La sémantique de cette mise à jour suit le même principe que la sémantique du $\mathbf{delete}\ q_{\mathtt{cib}}$.

2.5.2.2 Pré-évaluation des mises à jour composées

Les règles spécifiant la pré-évaluation des mises à jour composées sont données dans la Figure 2.18. Il n'est pas nécessaire de commenter ces règles étant donné qu'elles s'inspirent de celles de l'évaluation des requêtes qui ont été expliquées dans le détail. En effet, l'évaluation de la partie contextuelle des mises à jour est identique à l'évaluation de celle des requêtes composées vues dans la sous-section 2.4.2. La seule différence réside dans le résultat retourné pas ces règles qui pour le cas des mises à jour consiste en une liste de mises à jour primitives ainsi qu'un store décrivant les éléments sources.

Remarquer que la construction des PUL se fait en utilisant la règle (**Up:Comp**) qui capture la sémantique de la composition séquentielle de mises à jour, et la règle (**Up:For:iter**) qui spécifie le résultat des itérations produites par la pré-évaluation de la mise à jour itérative $\mathbf{for}\ x\ \mathtt{in}\ q_0\ \mathbf{return}\ u_1$.

Exemple 17. Nous allons maintenant illustrer la pré-évaluation des mises à jour en utilisant l'exemple de la Figure 2.19 qui montre un document t, une mise à jour u_1 devant être pré-évaluée sur t et $(\sigma_{\omega_1}, \omega_1)$ le résultat de cette pré-évaluation.

(**Up**:**Empty**) $$\frac{}{\sigma, \sigma_\epsilon, d, \gamma \models () \Rightarrow \emptyset, \epsilon}$$

(**Up**:**Comp**) $$\frac{\sigma, \sigma_\epsilon, d, \gamma \models u_1 \Rightarrow \sigma_1, \omega_1 \quad \sigma, \sigma_\epsilon, d \cup \mathit{dom}(\sigma_1), \gamma \models u_2 \Rightarrow \sigma_2, \omega_2}{\sigma, \sigma_\epsilon, d, \gamma \models u_1, u_2 \Rightarrow \sigma_1 \cdot \sigma_2, \omega_1 \cdot \omega_2}$$

(**Up**:**IfThen**) $$\frac{\sigma, \sigma_\epsilon, d, \gamma \models q_0 \Rightarrow \sigma_0, \mathrm{I}_0 \quad \mathrm{I}_0 \neq () \quad \sigma, \sigma_\epsilon, d \cup \mathit{dom}(\sigma_0), \gamma \models u_1 \Rightarrow \sigma_1, \omega_1}{\sigma, \sigma_\epsilon, d, \gamma \models \textbf{if } q_0 \textbf{ then } u_1 \textbf{ else } u_2 \Rightarrow \sigma_1, \omega_1}$$

(**Up**:**IfElse**) $$\frac{\sigma, \sigma_\epsilon, d, \gamma \models q_0 \Rightarrow \sigma_0, () \quad \sigma, \sigma_\epsilon, d \cup \mathit{dom}(\sigma_0), \gamma \models u_2 \Rightarrow \sigma_2, \omega_2}{\sigma, \sigma_\epsilon, d, \gamma \models \textbf{if } q_0 \textbf{ then } u_1 \textbf{ else } u_2 \Rightarrow \sigma_2, \omega_2}$$

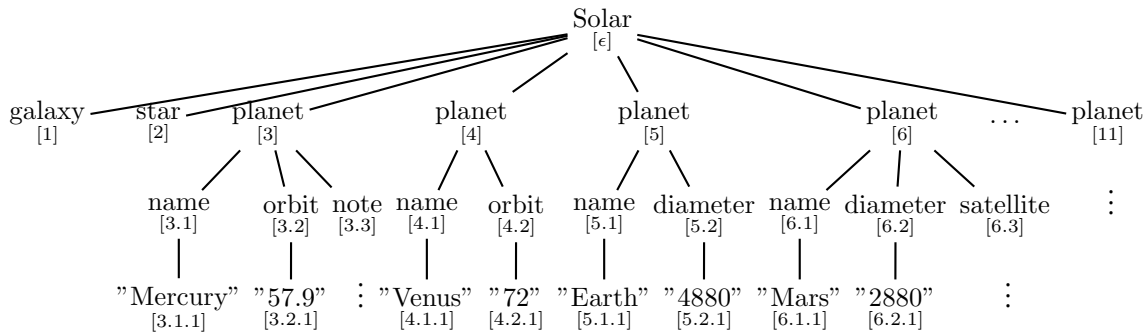
(**Up**:**Let**) $$\frac{\sigma, \sigma_\epsilon, d, \gamma \models q_0 \Rightarrow \sigma_0, \mathrm{I}_0 \quad \sigma, \sigma_\epsilon \cdot \sigma_0, d, ext(\gamma, x, \mathrm{I}_0) \models u_1 \Rightarrow \sigma_1, \omega_1}{\sigma, \sigma_\epsilon, d, \gamma \models \textbf{let } x \texttt{ := } q_0 \textbf{ return } u_1 \Rightarrow \sigma_1, \omega_1}$$

(**Up**:**For**) $$\frac{\sigma, \sigma_\epsilon, d, \gamma \models q_0 \Rightarrow \sigma_0, \mathrm{I}_0 \quad \sigma, \sigma_\epsilon \cdot \sigma_0, d, \gamma, x \in \mathrm{I}_0 \models^\star u_1 \Rightarrow \sigma_1, \omega_1}{\sigma, \sigma_\epsilon, d, \gamma \models \textbf{for } x \texttt{ in } q_0 \textbf{ return } u_1 \Rightarrow \sigma_1, \omega_1}$$

(**Up**:**For**:**stop**) $$\frac{}{\sigma, \sigma_\epsilon, d, \gamma, x \in () \models^\star u \Rightarrow \emptyset, \epsilon}$$

(**Up**:**For**:**iter**) $$\frac{\sigma, \sigma_\epsilon, d, ext(\gamma, x, \mathbf{i}_0) \models u \Rightarrow \sigma_1, \omega_1 \quad \sigma, \sigma_\epsilon, d \cup \mathit{dom}(\sigma_1), \gamma, x \in \mathrm{I}_0 \models^\star u \Rightarrow \sigma_2, \omega_2}{\sigma, \sigma_\epsilon, d, \gamma, x \in \mathbf{i}_0 \cdot \mathrm{I}_0 \models^\star u \Rightarrow \sigma_1 \cdot \sigma_2, \omega_1 \cdot \omega_2}$$

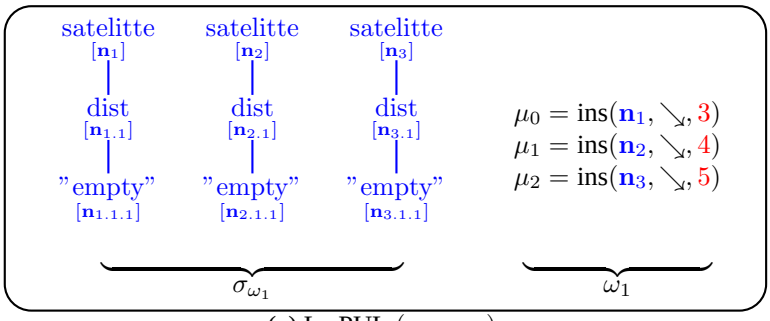
FIGURE 2.18 – Sémantique des mises à jour composées.



(a)- Le document t

for $\$x$ in self : :Solar/planet[not(satellite)]
return insert <satellite><dist>empty</dist></satellite>
as-last into $\$x$

(b)- La mise à jour u_1



(c) La PUL $(\sigma_{\omega_1}, \omega_1)$.

FIGURE 2.19 – Pré-évaluation de la mise à jour u_1 sur le document t

La pré-évaluation de u_1 utilise la règle ($\mathbf{Up{:}For}$) donnée dans la Figure 2.18. D'après cette règle, la pré-évaluation de u_1 commence par sa partie contextuelle. A partir de ce résultat, la partie mise à jour sera pré-évaluée itérativement produisant à chaque itération une mise à jour primitive qui sera concaténée à la PUL.

- L'évaluation de $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}[\mathrm{not}(\mathrm{satellite})]$ sur t retourne l'id-seq $3, 4, 5$ (les identifiants des noeuds planet n'ayant pas de fils $\mathrm{satellite}$). L'environnement est enrichit par la liaison de $x = id - seq$.
- Pour chacun de ces noeuds, la mise à jour
  $\mathbf{insert}$ <satellite><dist>empty<dist/></satellite>$\mathbf{as{-}last\ into}$ $\mathrm{x}
  est pré-évaluée en application de la règle ($\mathbf{Up{:}insert}$) de la Figure 2.17.
- A chaque pas de l'itération un nouvel élément $\mathrm{satellite}$ est créé et une mise à jour primitive de la forme ins($\mathbf{n}, \searrow, \mathbf{i}$) est ajoutée à la PUL.

■

Nous allons maintenant énoncer les propriétés vérifiées par les règles de la pré-évaluation des mises à jour.

Propriété 4 (Pré-évaluation des mises à jour)**.**
Soit t un document, u une mise à jour et (σ_ω, ω)=$PUL(t, u)$ le résultat de la pré-évaluation de u sur t. On a :

$$\forall \mathbf{i} \in Cible(\omega),\ \mathbf{i} \in dom(t) \tag{2.1a}$$

$$Source(\omega) = roots(\sigma_\omega) \tag{2.1b}$$

Le point (2.1a) rappelle l'hypothèse concernant les noeuds cibles des mises à jour primitives : ces noeuds appartiennent à t. Le point (2.1b) énonce que les règles de pré-évaluation des mises à jour ne retournent que les éléments devant être insérés par les mises à jour primitives générées. Autrement dit, les éléments intermédiaires construits lors de cette pré-évaluation ne sont pas retournés.

Application des mises à jour primitives Nous abordons maintenant la deuxième étape de la sémantique des mises à jour qui consiste à appliquer les mises à jour primitives générées lors de l'étape de pré-évaluation sur le store initial.

L'application d'une PUL ω à partir d'un store σ et produisant un store mis à jour σ_u est capturée par le jugement suivant :

$$\sigma, \sigma_\omega \models \omega \rightsquigarrow \sigma_u \quad \textbf{(Jug-Apli-PUL)}$$

Bien évidemment, lorsque le store initial σ est un arbre, l'application de (σ_ω, ω) retourne aussi un arbre σ_u puisque, la spécification du W3C [XUP] fait hypothèse que la racine de σ ne peut ni être cible d'une suppression ni d'une insertion avec la direction est soit $\leftarrow$ ou $\rightarrow$.

La notation alternative pour ce jugement est donnée par :

$$\boxed{ApplyPUL(\sigma, \sigma_\omega, \omega)=\sigma_u \text{ ssi } \sigma, \sigma_\omega \models \omega \rightsquigarrow \sigma_u}$$

$$(\textbf{ins-before}) \quad \frac{\sigma(\mathbf{j}) = a[\mathrm{I}_1 \cdot \mathbf{i} \cdot \mathrm{I}_2]}{\sigma, \sigma_\omega \models \mathrm{ins}(\mathrm{I}, \leftarrow, \mathbf{i}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{j}, a[\mathrm{I}_1 \cdot \mathrm{I} \cdot \mathbf{i} \cdot \mathrm{I}_2], \sigma_\omega)}$$

$$(\textbf{ins-after}) \quad \frac{\sigma(\mathbf{i}) = a[\mathrm{I}_1 \cdot \mathbf{i} \cdot \mathrm{I}_2]}{\sigma, \sigma_\omega \models \mathrm{ins}(\mathrm{I}, \rightarrow, \mathbf{i}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{j}, a[\mathrm{I}_1 \cdot \mathbf{i} \cdot \mathrm{I} \cdot \mathrm{I}_2], \sigma_\omega)}$$

$$(\textbf{ins-as-first}) \quad \frac{\sigma(\mathbf{i}) = a[\mathrm{I}_0]}{\sigma, \sigma_\omega \models \mathrm{ins}(\mathrm{I}, \swarrow, \mathbf{i}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{i}, a[\mathrm{I} \cdot \mathrm{I}_0], \sigma_\omega)}$$

$$(\textbf{ins-as-last}) \quad \frac{\sigma(\mathbf{i}) = a[\mathrm{I}_0]}{\sigma, \sigma_\omega \models \mathrm{ins}(\mathrm{I}, \searrow, \mathbf{i}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{i}, a[\mathrm{I}_0 \cdot \mathrm{I}], \sigma_\omega)}$$

$$(\textbf{ins-into}) \quad \frac{\sigma(\mathbf{j}) = a[\mathrm{I}_1 \cdot \mathrm{I}_2]}{\sigma, \sigma_\omega \models \mathrm{ins}(\mathrm{I}, \downarrow, \mathbf{i}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{j}, a[\mathrm{I}_1 \cdot \mathrm{I} \cdot \mathrm{I}_2], \sigma_\omega)}$$

$$(\textbf{replace}) \quad \frac{\sigma(\mathbf{j}) = a[\mathrm{I}_1 \cdot \mathbf{i} \cdot \mathrm{I}_2]}{\sigma, \sigma_\omega \models \mathrm{repl}(\mathbf{i}, \mathrm{I}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{j}, a[\mathrm{I}_1 \cdot \mathrm{I} \cdot \mathrm{I}_2], \sigma_\omega)}$$

$$(\textbf{delete}) \quad \frac{\sigma(\mathbf{j}) = a[\mathrm{I}_1 \cdot \mathbf{i} \cdot \mathrm{I}_2]}{\sigma, \sigma_\omega \models \mathrm{del}(\mathbf{i}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{j}, a[\mathrm{I}_1 \cdot \mathrm{I}_2], \sigma_\omega)}$$

$$(\textbf{rename}) \quad \frac{\sigma(\mathbf{i}) = a[\mathrm{I}]}{\sigma, \sigma_\omega \models \mathrm{ren}(b, \mathbf{i}) \rightsquigarrow \mathit{SubStore}(\sigma, \mathbf{i}, b[\mathrm{I}], \sigma_\omega)}$$

FIGURE 2.20 – Effets des mises à jour primitives

Les règles d'application des mises à jour primitives sont données dans la Figure 2.20 et ont déjà été illustrées dans la Table 2.1.

Nous regroupons les règles pour les différentes variantes de $\text{ins}(\text{I}, \delta, \mathbf{i})$ selon le cas où l'insertion de la forêt identifiée par I s'effectue :

- à l'intérieur du noeud **i**, càd, au début $\swarrow$, à la fin $\searrow$ où à un emplacement arbitraire $\downarrow$;
- en périphérie du noeud **i**, càd, comme voisin (successif) droit $\rightarrow$ ou voisin (successif) gauche $\leftarrow$.

L'effet de la mise à jour $\text{repl}(\mathbf{i}, \text{I})$ est le même que celui de l'application d'une suppression de **i** suivie directement de l'insertion de I à l'emplacement initial de **i**.

L'effet de la mise à jour $\text{del}(\mathbf{i})$ est celui qui consiste à détacher le noeud **i** du store.

L'effet de $\text{ren}(\mathbf{i}, b)$ consiste à modifier l'étiquette de **i** en b.

Exemple 18. Considérons le document t et la PUL $(\sigma_{\omega_1}, \omega_1)$ de la Figure 2.19. L'application de $(\sigma_{\omega_1}, \omega_1)$ sur t produit le document $u_1(t)$ donné en Figure 2.23-(a). ■

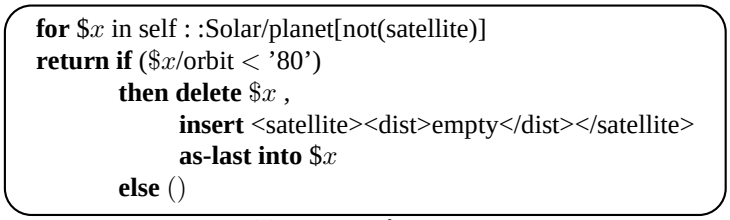
Il est important de souligner les deux points suivants concernant l'application des mises à jour primitives :

- Cette étape suppose une étape intermédiaire ayant pour but la vérification de certaines propriétés sur la PUL. Une des propriétés devant être vérifiées assure que la racine du document n'est pas la cible d'une suppression ou d'une insertion pour laquelle la direction est soit $\leftarrow$ ou $\rightarrow$. Dans ce travail, nous ne détaillons pas cette étape que nous nous contentons de modéliser par un prédicat $\text{CheckPUL}(\omega)$.
- Du fait de la sémantique en deux phases, les mises à jour primitives ne sont pas appliquées dans l'ordre de leur génération. En fonction de leur types (insertion, suppression,...), l'application de certaines mises à jour primitives est "différée" afin de garantir la validité des mises à jour primitives générées lors de l'étape de pré-évaluation.

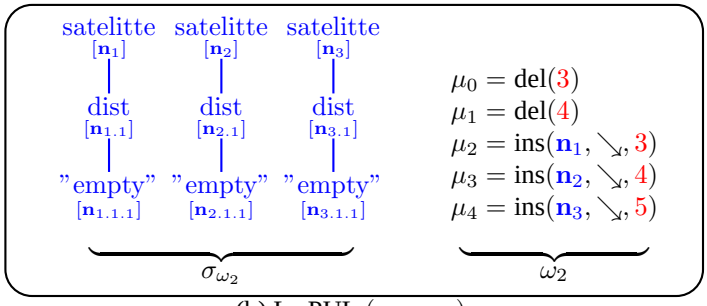
Exemple 19. Afin de mieux comprendre l'intérêt du réordonnancement des mises à jour primitives nous allons considérer la mise à jour u_2 donnée en Figure 2.21 et la PUL ω_2 générée par sa pré-évaluation à partir du document t de la Figure 2.19-(a). Si les mises à jour primitives de ω_2 sont appliquées suivant leur ordre dans la liste, c'est à dire μ_0 suivie de μ_1 suivie de etc, alors du fait de la suppression du noeud identifié par 3 et du noeud identifié par 4 par les primitives μ_0 et μ_1 respectivement, les mises à jour primitives μ_2 et μ_3 ne seraient pas valides car leur noeud cible n'existe pas dans le document au moment de leur application. ■

L'ordre qu'introduit le W3C [XUP] pour les mises à jour primitives consiste à évaluer d'abord les primitives d'insertions avec leur différentes variantes et la primitive de renommage ensuite la primitive de remplacement et enfin la primitive de suppression.

Nous définissons la fonction $\text{OrderPUL}(\omega)$ qui permet de trier les mises à jour d'une PUL ω en fonction de cet ordre. Cette fonction s'appuie sur $\text{Stage}(\mu)$ qui spécifie pour chaque type de mise à jour son ordre d'application relativement à celui des autres types de primitives.



(a)- La mise à jour u_2



(b) La PUL $(\sigma_{\omega_2}, \omega_2)$.

FIGURE 2.21 – Pré-évaluation de la mise à jour u_2 sur le document t

$\text{Stage}(\text{ins}(-,\downarrow,-))$	$=1$	
$\text{Stage}(\text{ren}(-,-))$	$=1$	
$\text{Stage}(\text{ins}(-,\delta,-))$	$=2$	pour $\delta \in \{\leftarrow,\rightarrow,\swarrow,\searrow\}$
$\text{Stage}(\text{repl}(-,-))$	$=3$	
$\text{Stage}(\text{del}(-))$	$=4$	

Définition 14 ($\text{OrderPUL}()$)**.**
Soit ω une PUL. $\text{OrderPUL}(\omega)$ est défini par :
$\text{OrderPUL}(\omega)=\text{PULSel}_1(\omega)\cdot\text{PULSel}_2(\omega)\cdot\text{PULSel}_3(\omega)\cdot\text{PULSel}_4(\omega)$, où :

$$
\begin{array}{lcll}
\text{PULSel}_i(\epsilon) & = & \epsilon & \\
\text{PULSel}_i(\mu\cdot\omega) & = & \mu\cdot\text{PULSel}_i(\omega) & \text{si } \text{Stage}(\mu)=i \\
\text{PULSel}_i(\mu\cdot\omega) & = & \text{PULSel}_i(\omega) & \text{sinon}
\end{array}
$$

Exemple 20. Soit ω_2 la PUL de la Figure 2.21. Alors on a :

- $\text{PULSel}_1(\omega_2) = ()$,
- $\text{PULSel}_2(\omega_2) = \mu_2 \cdot \mu_3 \cdot \mu_4$,
- $\text{PULSel}_3(\omega_2) = ()$, et
- $\text{PULSel}_4(\omega_2) = \mu_0 \cdot \mu_1$.

Donc, $\text{OrderPUL}(\omega_2) = \mu_2 \cdot \mu_3 \cdot \mu_4 \cdot \mu_0 \cdot \mu_1$. ■

Dans la littérature qu'il existe un autre langage de mise à jour XQuery ! [GRS06] qui diffère de XQuery Update du point de la sémantique. Le choix de l'utilisation de XQuery Update est dicté par le fait que c'est le langage standard pour la mise à jour des documents XML.

Les règles spécifiant l'évaluation d'une mise à jour u sur un document σ sont données par la Figure 2.22. La règle (**Up**:**Main**) spécifie les deux étapes principales de l'évaluation ainsi que les étapes intermédiaires. Noter que nous supposons le contexte dynamique de départ est vide.

Les deux règles restantes spécifient l'application des mises à jour primitives obtenues par l'étape de la pré-évaluation après avoir été vérifiées par $\mathrm{CheckPUL}()$ et triées par $\mathrm{OrderPUL}()$.

$$(\mathbf{Up}{:}\mathbf{Main}) \quad \frac{\sigma, \varnothing, \emptyset, ()\models u \Rightarrow \sigma_\omega, \omega_1 \quad \mathrm{CheckPUL}(\omega_1) \quad \mathrm{OrderPUL}(\omega_1){=}\omega \quad \sigma, \sigma_\omega\models \omega \rightsquigarrow \sigma_u}{\sigma\models u \rightsquigarrow \sigma_u}$$

$$(\mathbf{ApplyPUL}{:}\mathbf{empty}) \quad \frac{}{\sigma, \sigma_\omega\models \epsilon \rightsquigarrow \sigma}$$

$$(\mathbf{ApplyPUL}{:}\mathbf{iter}) \quad \frac{\omega{=}\mu_1{\cdot}\omega_1 \quad \sigma, \sigma_\omega\models \mu_1 \rightsquigarrow \sigma_{\mu_1} \quad \sigma_{\mu_1}, \sigma_\omega\models \omega_1 \rightsquigarrow \sigma_u}{\sigma, \sigma_\omega\models \omega \rightsquigarrow \sigma_u}$$

FIGURE 2.22 – Sémantique des mises à jour : pré-évaluation et applications

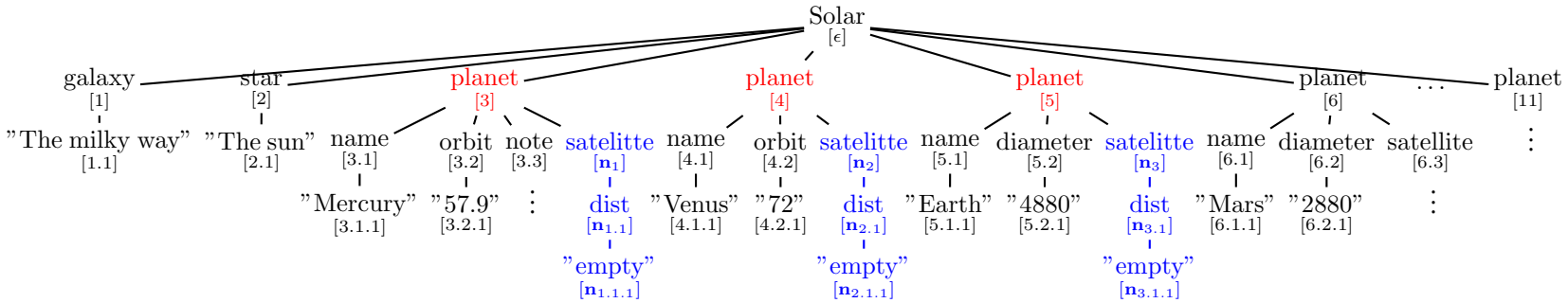
Exemple 21. Le résultat de l'application des mises à jour primitives de ω_2 sur t est le document $u_2(t)$ donné en Figure 2.23-(b). Le document intermédiaire obtenu en appliquant les mises à jour $\mu_2{\cdot}\mu_3{\cdot}\mu_4$ est le document $u_1(t)$ donné dans la partie (a) de la même figure. ■

2.5.3 Autres langages de mises à jour

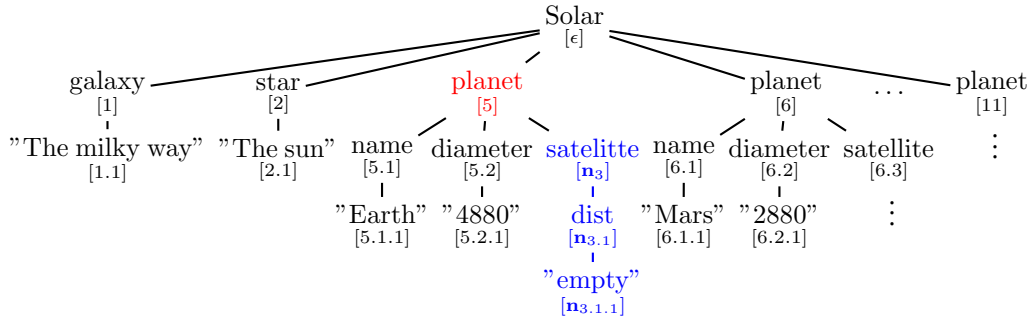
D'autres langages de mises à jour XML ont été proposé dans [TIHW01a], [BBFV05] et [GRS06]. Les deux premiers langages proposés utilisent la sémantique à deux phases permettant de pré-évaluer une mise à jour puis d'appliquer les mises à jour primitives. Dans [TIHW01a] les auteurs utilisent des contraintes lors de la génération des mises à jour afin d'éviter des mises à jour incohérentes. Dans [BBFV05] les auteurs permettent le choix entre différentes ordonnancements possibles des mises à jour primitives. Ils expliquent les avantages et les limites de chaque ordonnancement. Par exemple, un ordonnancement consistant à appliquer les suppressions en premier permet d'optimiser le stockage des documents. Cependant, cet ordonnancement nécessite de vérifier certaines propriétés de correction de la mise à jour. Le langage XQuery ! proposé par [GRS06] diffère des langages précédents et de XQuery Update puisqu'il permet d'appliquer les mises à jour primitives au fur et à mesure de leur génération. L'objectif est d'éviter la matérialisation, parfois inutile, de résultats intermédiaires nécessaire pour les langages utilisant la sémantique à deux phases. Pour notre exemple, l'utilisation de la sémantique XQuery ! dans le cas de la mise à jour u_2 de la Figure 2.21-(a) évite de stocker une partie du store σ_{ω_2} généré lors de la phase de pré-évaluation de u_2 puisque les noeuds identifiés par 3 et 4 sont supprimés par l'application de la mise à jour primitive générée à partir de $\mathbf{delete}\ \$x$.

2.6 Conclusion

Dans cette section nous avons introduit et formalisé le modèle de données XML ainsi que les différents langages de manipulations des documents XML. Concernant le langage de chemins XPath nous avons repris la sémantique standard telle que définie dans plusieurs travaux. Concernant les langages d'interrogation XQuery et de mises à jour XQuery Update, nous avons adapté la sémantique introduite dans [BC09] à nos besoins.



(a) $u_1(t)$



(b) $u_2(t)$

FIGURE 2.23 – Evaluation de u_1 et de u_2 sur t

Première partie

Optimisation des mises à jour XML par projection

Chapitre 3

Projection pour les requêtes XML : état de l'art

3.1 Introduction

XQuery [XQu] est à l'origine un langage destiné à l'interrogation des données XML. Récemment, il est devenu un langage à multi-usage servant aussi bien pour poser des requêtes sur des collections de documents XML que pour effectuer différents types de traitements tels que la recherche textuelle, l'intégration de données, etc. La plupart de ces traitements ne requièrent pas les fonctionnalités complexes des SGBD relationnels telles que la gestion des transactions ou le stockage d'index en mémoire secondaire. Ceci a encouragé le développement de moteurs mémoire-centrale tels que [gal], [exi],[saxb] ou [qizb] qui manipulent les données directement en mémoire centrale. Les avantages qu'offrent ces moteurs comparés aux systèmes utilisant la mémoire secondaire sont la facilité de déploiement et d'utilisation. Cependant, ils restent limités par la taille de la mémoire centrale disponible et sont en particulier incapables de traiter des documents volumineux.

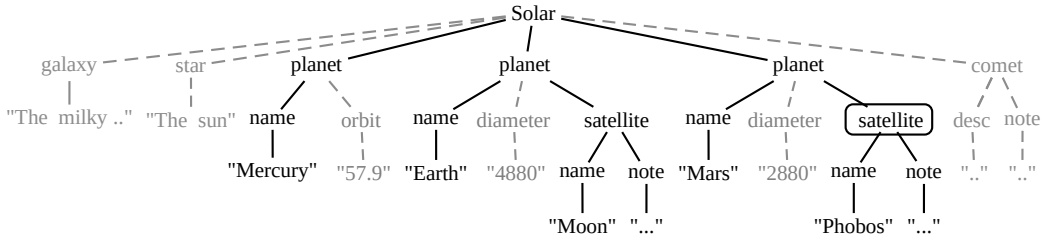
La *projection* XML est une technique d'optimisation introduite par [MS03] pour résoudre ce problème. Son principe est très simple et consiste à charger en mémoire uniquement les parties du document qui sont nécessaires à son traitement. Le scénario de la projection pour le cas de l'évaluation d'une requête Q sur un document t suit deux étapes. La première étape vise à déterminer de manière statique, c'est à dire en analysant la requête Q et éventuellement le schéma du document interrogé, s'il est disponible, la partie t' du document t qui est nécessaire à l'évaluation de Q. L'information inférée pendant cette étape est utilisée lors du chargement du document t pour élaguer les parties qui ne sont pas accédées lors de l'évaluation de Q. L'évaluation de Q s'effectue sur la projection t' qui permet d'obtenir le même résultat que si l'évaluation était effectuée sur le document original t. L'efficacité de la technique de projection tient du fait qu'en général, les requêtes sont assez sélectives, c'est dire qu'elles nécessitent uniquement une partie du document interrogé.

Exemple 22. Afin d'illustrer la projection, considérons la requête Q et le document t présentés dans la Figure 3.1. Cette requête extrait les éléments $\mathrm{satellite}$ de t qui sont fils des noeuds planet satisfaisant une condition particulière (le fils name de planet doit contenir le texte "Mars"). La projection de t pour cette requête doit alors contenir les noeuds Solar, planet et name qui sont utilisés par la requête Q soit pour la navigation soit pour exprimer une condition, ainsi que le

sous-élément de satellite qui est retourné par la requête. Dans le but de faciliter la compréhension de l'effet de la projection sur le document t, nous utilisons un seul document représentant à la fois le document original t et sa projection. Les noeuds grisés sont ceux élagués par la projection. L'évaluation de Q sur le document original retourne l'élément satellite dont la racine est encadrée. Il aisé de constater que l'évaluation de Q sur la projection retourne le même résultat.

let $x := self :: Solar/planet
where $x/name/text()='Mars'
return $x/satellite

(a) La requête Q



(b) Le document t

FIGURE 3.1 – Illustration de la projection

■

Déterminer la meilleure projection pour une requête est l'objectif principal des différentes approches proposées dans la littérature [MS03, BCL$^+$05, BCCN06] dont l'objectif est clairement de minimiser la quantité de mémoire nécessaire pour l'évaluation des requêtes. L'analyse des requêtes constitue le point central de chacune de ces approches puisqu'elle permet d'inférer données nécessaires à leur évaluation. Dans [MS03], les auteurs proposent d'extraire les chemins exprimés dans la requête Q et de les utiliser pour élaguer le document interrogé t. Lors du chargement de t, les chemins extraits de Q sont évalués afin d'identifier et d'élaguer les noeuds de t qui ne sont pas accédés par Q. Cette approche possède deux avantages : (i) elle ne nécessite aucune information sur la structure du document interrogé et (ii) elle peut être utilisée dans le cas où un ensemble de requêtes est appliqué sur un seul document. Cependant, elle souffre de plusieurs limitations. La première limitation est liée à la faisabilité de la technique et concerne le type de chemins pris en compte. Les chemins contenant des axes arrière (`parent` et `ancs`) ou exprimant des prédicats (conditions) ne peuvent pas être évalués au vol et donc ne peuvent pas être considérés par cette technique. La deuxième limitation, liée à l'efficacité de la technique, découle du point (i) et survient lorsque les chemins contenant l'axe `desc-or-self` (//) sont utilisés pour l'élagage. L'évaluation de tels chemins requiert de tester tous les descendants d'un noeud donné et peut ralentir considérablement l'étape l'élagage.

La technique développée dans [BCCN06] permet de résoudre ces deux problèmes en exploitant l'information sur le type (schéma) des documents interrogés. Elle considère que le document interrogé est valide relativement à une DTD et utilise cette dernière lors de l'analyse de la requête Q pour extraire les étiquettes des noeuds accédés par Q. Cet ensemble d'étiquettes constitue le projecteur. L'élagage se trouve alors simplifié puisque il suffit de charger uniquement les noeuds du document

dont l'étiquette appartient au projecteur. Cette approche possède plusieurs avantages comparée à [MS03]. Premièrement, comme les chemins sont utilisés uniquement au niveau de l'analyse statique (ils ne servent pas effectuer l'élagage du document) tous les types de chemins.Deuxièmement, l'élagage du document interrogé s'effectue de manière efficace puisqu'il repose sur un processus simple : la décision de projeter un noeud ou non repose sur le test d'appartenance de son étiquette au projecteur. La technique de [BCL$^+$05] possède un point en commun avec la technique de [BCCN06] qui concerne l'utilisation de l'information sur la structure du document. Elle se base sur l'utilisation d'un DataGuide construit à partir du document interrogé et sur la transformation de la requête Q en une représentation arborescente. L'élagage consiste à établir la correspondance entre les noeuds de t et ceux de cette représentation. Des indexes construits à partir du data guide sont utilisés pour accélérer cette phase d'élagage. L'avantage de cette technique est qu'elle est précise du fait de l'utilisation du data guide. Ses inconvénients sont liés d'abord à la nécessité de transformer la requête Q en une représentation intermédiaire rendant l'approche inutilisable dans le cas où plusieurs requêtes sont utilisées. Cette technique se limite aussi aux chemins qui utilisent des axes en avant seulement et qui n'expriment pas de condition. Enfin, cette technique nécessite de construire des index dans le but d'accélérer l'étape d'élagage.

Dans de ce chapitre nous présentons les techniques de projection développées dans [MS03] et [BCCN06] sur lesquelles se base notre travail. Nous présentons d'abord l'approche proposée dans [MS03] basée sur les chemins qui s'applique dans le cas général en l'absence de schéma pour les documents interrogés. Nous présentons ensuite la technique de projection basée sur les schémas proposée dans [BCCN06] qui suppose que les documents respectent un schéma donné.

3.2 Projection basée sur les chemins

Le scénario de l'approche développée dans [MS03] consiste, pour un document t et une requête Q, en deux étapes :

1. Analyse statique de la requête Q qui consiste à extraire les chemins qui sont exprimés dans Q et qui permettent de déterminer la partie du document t nécessaire à son évaluation ;
2. Elagage du document t, lors de sont chargement, en utilisant les chemins extraits de Q.
3. Evaluation de la requête Q sur la projection t' obtenue à l'issue de l'étape 2.

L'analyse statique proposée dans [MS03] est *correcte* et garantit que l'évaluation de la requête Q sur le document projeté retourne le même résultat que l'évaluation sur le document original, i.e. $Q(t)$=$Q(t')$.

Le but de cette section est de présenter d'abord la technique d'extraction des chemins développée dans [MS03]. Nous discutons des limitations de cette approche ensuite.

3.2.1 L'extraction des chemins à partir des requêtes

L'extraction des chemins pour les requêtes tient compte de la forme générale de celles-ci présentée dans le chapitre 2. Les requêtes sont principalement construites à partir de chemins qui assurent la navigation dans le document interrogé. Ces chemins sont utilisés pour exprimer des conditions et bien plus important encore, pour spécifier les résultats retournés par les requêtes. Afin de tenir

compte des différents rôles joués par les chemins au sein des requêtes et afin de permettre de déterminer avec précision les noeuds devant être gardés dans la projection, la technique d'extraction des chemins développée par [MS03] distingue deux types de chemins : les chemins *returned* correspondant aux chemins qui spécifient les résultats retournés par les requêtes et les chemins *used* correspondant aux autres chemins exprimés dans les requêtes et qui permettent soit la navigation, soit la construction de résultats intermédiaires. Le but de cette distinction est de permettre, lors de l'étape d'élagage, de différencier les noeuds devant être projetés seuls parce qu'utilisés seulement pour naviguer dans le document des noeuds devant être projetés avec leurs descendants parce qu'ils sont les racines des réponses aux requêtes.

Exemple 23. Considérons l'exemple de la Figure 3.1(a) pour illustrer les deux catégories de chemins. A partir de la la requête Q sont génèrés deux chemins *used* :

- $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}$ qui est utilisé pour naviguer dans les éléments
- $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}/\mathrm{name}/\mathrm{text}()$ qui sert à exprimer une condition.

et un chemin *returned* qui est

- $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}/\mathrm{satellite}$

La projection à partir de ces chemins, illustrée dans la Figure 3.1(b), montre bien que les noeuds qui sont projetés sont ceux traversés par l'un des chemins extraits de Q, c'est à dire, les noeuds étiquetés $\mathrm{Solar}, \mathrm{planet}, \mathrm{name}, \mathrm{satellite}$ ou les noeuds descendants du chemin *returned* de Q qui sont les noeuds étiquetés name et note. Dans cet exemple name est à la fois utilisé et retourné par Q. ■

La technique d'extraction développée dans [MS03] permet d'analyser des requêtes complexes construites par imbrication ou composition d'autres requêtes. Les aspects les plus importants de cette extraction sont présentés à l'aide de la Table 3.1. Les fonctions $Use(\Gamma, q)$ et $Ret(\Gamma, q)$ permettent de capturer l'extraction des chemins *used*, *returned* à partir de la requête q en présence de l'environnement statique Γ. Le rôle de l'environnement statique dans l'extraction des chemins est analogue au rôle que joue l'environnement dynamique dans l'évaluation des expressions **for** x `in` $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$ ou **let** x = $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$ (cf. Chapitre 2) et consiste à garder trace des liaisons entre la variable x et le résultat de l'évaluation de la requête contextuelle $q_{\texttt{ctxt}}$. Dans le cas de l'analyse statique, l'environnement Γ utilisé pour garder trace des liaisons impliquant la variable x est l'ensemble des chemins *returned* extraits à partir de $q_{\texttt{ctxt}}$. L'enrichissement d'un environnement statique Γ par une liaison entre une variable x et un ensemble de chemins $\boldsymbol{P}$ est noté $\Gamma \cdot x \mapsto \boldsymbol{P}$.

Nous commentons le principe d'extraction de chaque requête rapportée dans la Table 3.1.

Les variables La première ligne de la Table 3.1 correspond au traitement d'une variable x en tant que chemin ou composant d'un chemin (par exemple x/P), l'extraction utilise l'environnement statique construit jusqu'alors. Cet environnement contient normalement une liaison $x \mapsto \boldsymbol{P}$ pour la variable x où $\boldsymbol{P}$ est un ensemble de chemins $\{P_1, \ldots, P_n\}$ signifiant, évidemment, que x peut "valoir" P_1, ou P_2, ... ou P_n.

Les chemins La deuxième ligne de la Table 3.1 correspond au cas d'une requête réduite a un chemin. Le traitement de ce cas consiste à considérer le chemin comme étant *returned*.

Variables	$Use(\Gamma, x) = \emptyset$ $Ret(\Gamma, x) = \{P \in \boldsymbol{P} \mid x \mapsto \boldsymbol{P} \in \Gamma\}$
Chemins	$Use(\Gamma, P) = \emptyset$ $Ret(\Gamma, P) = \{P\}$
Composition	$Use(\Gamma, (q_1, q_2)) = Use(\Gamma, q_1) \cup Use(\Gamma, q_2)$ $Ret(\Gamma, (q_1, q_2)) = Ret(\Gamma, q_1) \cup Ret(\Gamma, q_2)$
FLWR	$Use(\Gamma, \textbf{for } x \texttt{ in } q_{\texttt{ctxt}} \textbf{ return } q_{\texttt{res}}) =$ $Use(\Gamma, q_{\texttt{ctxt}}) \cup Ret(\Gamma, q_{\texttt{ctxt}}) \cup Use(\Gamma_{\texttt{ctxt}}, q_{\texttt{res}})$ $Ret(\Gamma, \textbf{for } x \texttt{ in } q_{\texttt{ctxt}} \textbf{ return } q_{\texttt{res}}) = Ret(\Gamma_{\texttt{ctxt}}, q_{\texttt{res}})$ où $\Gamma_{\texttt{ctxt}} = \Gamma \cdot x \mapsto Ret(\Gamma, q_{\texttt{ctxt}})$

TABLE 3.1 – Extraction des chemins à partir des requêtes [MS03].

La composition séquentielle L'extraction des chemins pour la requête q_1, q_2 est simple. Elle consiste à cumuler les chemins extraits séparément à partir de q_1 et de q_2.

Les requêtes FLWR Nous donnons uniquement la règle de l'extraction des chemins pour l'itération **for** x `in` $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$. Le traitement des requêtes de binding **let** x = $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$ et des requêtes conditionnelles **if** $q_{\texttt{ctxt}}$ **then** q_{res0} **else** q_{res1} suit le même principe.

La requête d'itération est composée d'une partie contextuelle $q_{\texttt{ctxt}}$ et d'une partie résultat $q_{\texttt{res}}$. La partie contextuelle va générer exclusivement des chemins *used* puisqu'elle sert à la navigation dans le document et à l'expression de conditions avec éventuellement le calcul de résultats intermédiaires. La requête résultat va générer des chemins *used* et des chemins *returned* pour la requête itérative et ceci de manière directe.

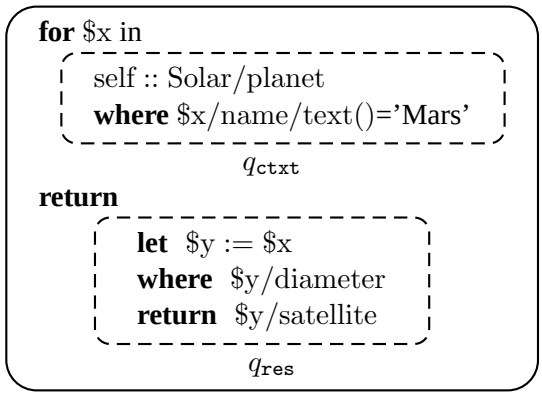
Exemple 24. Le but de cet exemple est d'illustrer l'extraction des chemins pour la requête Q de la Figure 3.2. Cette requête est de la forme **for** x `in` $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$ et suit donc la règle donnée dans la Table 3.1.
L'extraction des chemins à partir de la requête contextuelle $q_{\texttt{ctxt}}$ s'effectue à partir de l'environnement statique $\Gamma{=}\varnothing$. Elle retourne le path $\text{self} :: \text{Solar}/\text{planet}/\text{name}/\text{text}()$ comme *used* car il est dans la condition du **where** et le path $\text{self} :: \text{Solar}/\text{planet}$ comme *returned*.
L'extraction des chemins pour la requête résultat $q_{\texttt{res}}$ s'effectue à partir de l'environnement statique Γ_1 contenant la liaison $x \mapsto self :: Solar/planet$ parce que $\text{self} :: \text{Solar}/\text{planet}$ est le seul path *returned* de $q_{\texttt{res}}$. Elle retourne comme chemins *used*

- $\text{self} :: \text{Solar}/\text{planet}$ en raison du "**let** $\$y := \$x$" et de la liaison pour la variable x, et
- $\text{self} :: \text{Solar}/\text{planet}/\text{diameter}$ qui figure dans la condition **where** de $q_{\texttt{res}}$.

Elle retourne le chemin *returned* $\text{self} :: \text{Solar}/\text{planet}/\text{satellite}$ qui va générer le seul chemin *returned* de Q. ■

3.2.2 Limites d'utilisation

L'approche de [MS03] est confrontée aux problèmes de l'évaluation au vol des chemins XPath étudié dans [BYFJ04, GN11, Gau09]. D'après ces travaux seul le fragment exprimant les axes avant (`child` et `desc`) peut être évalué au vol en temps linéaire en la taille du chemin et en espace linéaire en la profondeur du document interrogé. Les chemins exprimant les axes arrière (`parent` et



(a) La requête analysée Q

$\Gamma = \varnothing$	
$Use(\Gamma,Q)$	self :: Solar/planet/name/text() self :: Solar/planet self :: Solar/planet/diameter
$Ret(\Gamma,Q)$	self :: Solar/planet/satellite

$\Gamma = \varnothing$	
$Use(\Gamma,q_{ctxt})$	self :: Solar/planet/name/text()
$Ret(\Gamma,q_{ctxt})$	self :: Solar/planet

$\Gamma_{\mathcal{C}} = x \mapsto$ {self :: Solar/planet}	
$Use(\Gamma_{\mathcal{C}},q_{res})$	self :: Solar/planet self :: Solar/planet/diameter
$Ret(\Gamma_{\mathcal{C}},q_{res})$	self :: Solar/planet/satellite

(b) L'extraction des paths de Q

FIGURE 3.2 – Illustration du mécanisme d'extraction de chemins de [MS03]

`ancs`) nécessitent quant à eux le stockage en mémoire tampon d'une partie du document interrogé et ne peuvent donc pas être évalués au vol. D'autre part, d'après [Olt07] la réécriture des axes arrière en axes avant ne peut être envisagée en pratique : elle peut produire des chemins d'une taille exponentielle en la taille des chemins en entrée. L'évaluation des chemins exprimant des conditions (prédicats) nécessite elle aussi de stocker en mémoire tampon une partie du document interrogé puisqu'une condition peut contenir par exemple un chemin absolu. La technique de projection de [MS03] se limite donc à la fois aux chemins qui ne contiennent pas d'axes arrière et qui n'expriment pas de conditions.

Cette technique présente un autre inconvénient lié au traitement des chemins exprimant l'axe `desc-or-self` (//). L'évaluation de tels chemins nécessite d'examiner un nombre important de noeuds du document interrogé. Considérons le document de la Figure 3.1(b) l'évaluation du chemin self :: Solar//name/text(). Cette évaluation nécessite l'examen de tous les noeuds du

document alors que seuls les éléments planet sont pertinents puisqu'ils sont les seuls à contenir les noeuds name traversés par le chemin. Cet exemple montre le rôle que pourrait jouer l'information concernant la structure du document dans l'accélération de l'étape d'élagage. En effet, si une telle information était disponible, le chemin $\mathrm{self} :: \mathrm{Solar}//\mathrm{name}/\mathrm{text}()$ pourrait être réécrit en $\mathrm{self} :: \mathrm{Solar}/\mathrm{planet}/\mathrm{name}/\mathrm{text}()$ permettant ainsi de guider la navigation dans le document t en lui indiquant qu'elle doit nécessairement passer par le noeud planet pour aboutir au noeud name. Il serait alors possible d'éviter d'examiner des éléments qui ne contiennent pas de noeud name tels que les éléments galaxy, star ou comet de notre exemple.

L'autre limitation de cette technique concerne son imprécision lorsque les conditions sont utuilisées. Cette technique ne permet pas d'analyser les conditions dans les cheminsce qui peut conduire à projeter plus de noeuds que nécessaire. L'élagage du document t de la Figure 3.1(b) en utilisant le chemin $\mathrm{self} :: \mathrm{Solar}//\texttt{node}()[\mathrm{note}]/\mathrm{satellite}$ conduit à retourner tout le document puisque la condition $[\mathrm{note}]$ n'est pas exploitée pour restreindre les noeuds projetés à ceux dont les descendants possèdent un noeud fils note.

3.3 Projection basée sur les schémas

La technique de projection proposée par [MS03] souffre de limitations dues à l'utilisation directe des chemins exprimés dans les requêtes pour l'élagage du document interrogé. Les problèmes posés par une telle approche concernent d'abord sa faisabilité, puisque seuls les chemins exprimant des axes descendants peuvent être traités, mais aussi son efficacité, puisque la présence de certains axes peut ralentir l'étape d'élagage. La technique développée dans [BCCN06] vise à résoudre ces problèmes en proposant une approche basée sur l'utilisation du schéma du document interrogé. L'idée consiste, pour une DTD D, un document t et une requête Q, à analyser statiquement Q et D pour extraire un ensemble d'étiquettes correspondant à une sur-estimation des noeuds traversés ou retournés par Q. Cet ensemble, appelé *projecteur* et noté π, est utilisé pour construire la projection $\pi(t)$ en élaguant le document t de manière efficace : lors du chargement du document t, la décision de projeter ou d'élaguer un noeud, lorsqu'il est visité, dépend d'un simple test consistant à vérifier si son étiquette appartient au projecteur. Le résultat principal de [BCCN06] énonce que l'évaluation de Q sur la projection produit le même résultat que l'évaluation sur le document original t, i.e. $Q(t)$=$Q(\pi(t))$.

L'avantage de l'approche proposée dans [BCCN06] relativement à celle proposée dans [MS03] est double :

- Du point de vue de la faisabilité, l'approche de [BCCN06] prend en compte les axes arrière ainsi que les conditions. Cela découle du fait que la technique [BCCN06] n'utilise pas les chemins pour l'élagage mais plutôt le projecteur inféré pour la requête Q et la DTD D.
- Du point de vue de l'efficacité, la performance de l'élagage de la technique de [BCCN06] n'est en général pas influencée par l'utilisation d'axes `desc-or-self` puisque l'utilisation de la DTD permet de raffiner l'information concernant les noeuds traversés par ces axes. Cette information est utilisée lors de l'élagage de manière directe via le projecteur.

Scénario de l'approche Le scénario pour une DTD D, un document t et une requête Q suit les étapes suivantes :

(1) la requête Q et la DTD D sont analysées dans le but d'inférer le projecteur π ;

(2) le document t est projeté, au vol, lors de son chargement, en utilisant le projecteur π ;

(3) la requête Q est évaluée sur la projection $\pi(t)$.

L'inférence du projecteur pour la requête Q et la DTD D passe d'abord par l'extraction des chemins exprimés dans Q. Cette extraction suit le principe de [MS03] en distinguant entre chemins *used* et chemins *returned*. L'étape suivante consiste à inférer les étiquettes des noeuds traversés ou retournés par chaque chemin extrait de Q. Cette étape utilise la DTD D et constitue la principale contribution de [BCCN06]. Par abus de langage, l'ensemble des étiquettes inférés pour un chemin P est appelé aussi projecteur dans [BCCN06].

Nous donnons un aperçu global de cette technique de [BCCN06] en nous basant sur l'exemple suivant.

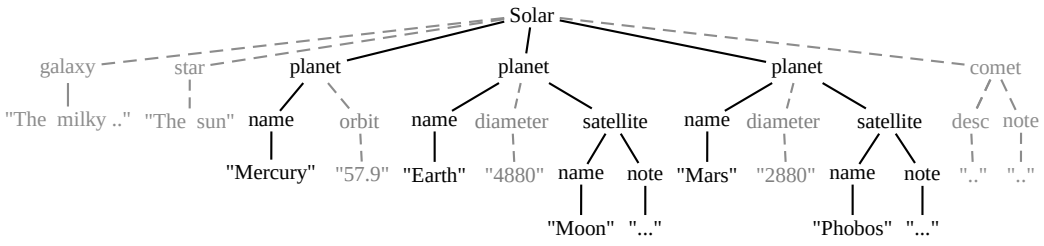
1. Solar	$\rightarrow$	galaxy, star, planet+, comet*
2. planet	$\rightarrow$	name, (orbit | diameter), note ?, satellite*
3. satellite	$\rightarrow$	name, note
4. comet	$\rightarrow$	desc ?, note
5. galaxy, star, orbit, name, note, diameter	$\rightarrow$	*String*

FIGURE 3.3 – La DTD $D_{\texttt{Sol}}$.

Exemple 25. Considérons la DTD de la Figure 3.3 et la requête Q et le document présentés dans la Figure 3.3.

let $x := self :: Solar//node()[note]/satellite
return $x

(a) La requête Q



(b) Le document t et sa projection

FIGURE 3.3 – Illustration de la projection basée sur les schémas

Nous nous focalisons d'abord sur l'étape 1 qui vise à inférer le projecteur π pour la requête Q. L'extraction des chemins à partir de Q suit la technique développée dans [MS03] : deux types de chemins *used* et *returned* sont inférés. Les chemins *used* inférés pour [BCCN06] sont les mêmes que pour [MS03]. Pour l'exemple, le seul chemin *used* est :

$P_1 = \text{self} :: \text{Solar}//\text{node}[\text{note}]/\text{satellite}.$

Le chemin *returned* inféré pour Q suivant [MS03] est :

$\text{self} :: \text{Solar}//\text{node}[\text{note}]/\text{satellite}$.

Ce chemin cible les racines des éléments qui doivent être projetés et dont on doit capturer les étiquettes des descendants. Les chemins *returned* inférés pour [BCCN06] sont obtenus à partir des chemins *returned* inférés pour [MS03] auxquels est rajoutée l'étape `desc-or-self` :: `node`(). Pour notre exemple, le chemin *returned* inféré pour [BCCN06] est

$P_2 =$ $\text{self} :: \text{Solar}//\text{node}()[\text{note}]/\text{satellite}/$`desc-or-self` :: `node`().

L'inférence du projecteur pour Q (sous-étape 1.b) passe par l'inférence des projecteurs des chemins P_1 et P_2 en utilisant la DTD D. Cela consiste à identifier pour chaque chemin les étiquettes des noeuds traversés ou retournés par ce chemin. Pour le chemin P_1 qui retourne les noeuds descendants de Solar ayant un fils name, le projecteur inféré est :

$$\pi_1 = \{\text{Solar}, \text{planet}, \text{satellite}, \text{note}\}$$

Pour le cas de satellite, le noeud lui correspondant est à la fois traversé par P_1 puisque satellite est un descendant de Solar, il est aussi retourné par P_1. Pour P_2 le projecteur inféré contient en plus de π_1 les étiquettes des descendants de satellite, i.e les étiquettes note, name et l'étiquette String qui est utilisée pour désigner les textes :

$$\pi_2 = \pi_1 \cup \{\text{note}, \text{name}, \text{String}\}$$

Le projecteur inféré pour Q est :

$$\pi = \{\text{Solar}, \text{planet}, \text{satellite}, \text{note}, \text{name}, \text{String}\}$$

Focalisons-nous maintenant sur l'étape 2 qui construit la projection $\pi(t)$. Pour notre exemple, cette projection est illustrée dans la Figure 3.3b. Cette projection est obtenue en élaguant le document t à partir du projecteur π. Cet élagage consiste à garder uniquement les noeuds dont l'étiquette appartient à π et à élaguer, avec tous leurs descendants, les noeuds dont l'étiquette n'appartient pas à π. Il est aisé de constater dans notre exemple que l'étiquette de chaque noeud qui est projeté appartient à π. En revanche, la réciproque est fausse puisqu'il peut y avoir des noeuds dont l'étiquette appartient à π mais qui ne soient pas projetés pour autant. Dans notre exemple, le dernier noeud note du document n'est pas projeté puisque son parent comet est élagué par π ($\text{comet} \notin \pi$). ■

Notons que l'utilisation de la technique de [MS03] pour élaguer le document t suivant le chemin $\text{self} :: \text{Solar}//$`node`$()[\text{note}]/\text{satellite}$ conduit à projeter t. Rappelons que cette technique ne permet pas d'analyser les conditions et de ce fait se limite à projeter tous les descendants des noeuds retournés par le chemin $\text{self} :: \text{Solar}//$`node`().

3.3.1 L'inférence du projecteur pour les chemins

La méthode d'inférence du projecteur développée dans [BCCN06] est assez sophistiquée et ceci pour répondre à deux exigences. D'une part, l'objectif est de traiter des chemins "sans restriction" et donc, de prendre en compte toute combinaison d'axes, de filtres et de conditions. D'autre part, l'objectif est de générer un projecteur le plus précis possible afin de ne pas compromettre l'efficacité de la méthode d'évaluation par projection. Ces deux objectifs interfèrent car, comme expliqué par les auteurs dans [BCCN06], la précision est justement rendue difficile par l'utilisation des axes arrière, du filtre `node`() ou des conditions. Les auteurs de [BCCN06] résolvent ces difficultés en introduisant un mécanisme auxiliaire, appelé *inférence des types*, qui permet d'inférer, avec

précision, pour une DTD D et un chemin P les étiquettes des noeuds ciblés par P. L'inférence du projecteur assure l'extraction des étiquettes de tous les noeuds traversés par P et utilise l'inférence des types.

Nous présentons d'abord l'inférence des types pour les chemins et montrons ensuite comment elle se greffe dans le système d'inférence du projecteur. La présentation qui suit est informelle et se base sur des exemples.

3.3.1.1 Inférence des types pour les chemins

Il est d'abord important de présenter la terminologie concernant les types des noeuds et les DTDs que nous utilisons dans la présentation. Le type d'un noeud est donné, sous sa forme la plus simple, par l'étiquette associée à ce noeud. Par exemple, le type de la racine de la DTD $D_{\texttt{Sol}}$ donnée en Figure 3.3, est Solar. Certaines DTDs permettent à une étiquette d'être générée à partir de deux symboles différents. Elles sont appelées ambiguës pour les parents (ou parent ambiguous). C'est le cas de la DTD $D_{\texttt{Sol}}$ où l'étiquette note est générée par le symbole $\mathrm{satellite}$ donné par la ligne 2 et le symbole comet donné par la ligne 4. L'étiquette d'un noeud est donc insuffisante à elle seule pour distinguer les noeuds. On peut alors penser à associer un contexte à chaque noeud en considérant en plus de l'étiquette de ce noeud, l'étiquette de son parent ou plus généralement l'ensemble des étiquettes qui se trouvent sur le chemin allant de la racine vers ce noeud. Dans notre exemple, le contexte du noeud note fils de $\mathrm{satellite}$ est donné par :

$\{\mathrm{satellite}, \mathrm{planet}, \mathrm{Solar}\}$

alors que le contexte du noeud note fils de comet est donné par :

$\{\mathrm{comet}, \mathrm{Solar}\}$.

La notion de type pour un noeud est étendue à un chemin pour lequel elle désigne l'ensemble des étiquettes des noeuds ciblé par ce chemin.

L'inférence des types d'un chemin P donné sous la forme $\mathrm{step_1}/\ldots/\mathrm{step_n}$ et une DTD D, consiste à analyser chaque étape $\mathrm{step_i}$ à partir d'une paire d'ensembles d'étiquettes (T_i, K_i) telle que :

- T_i contient les étiquettes des noeuds ciblés par le chemin $\mathrm{step_1}/\ldots/\mathrm{step_i}$ et
- K_i constitue le contexte des noeuds ciblés par $\mathrm{step_1}/\ldots/\mathrm{step_i}$.

Cette analyse retourne une paire d'ensembles d'étiquettes (T_{i+1}, K_{i+1}) où

- T_{i+1} est obtenu en pré-sélectionnant les étiquettes de D qui sont accédées à partir des étiquettes appartenant à T_i sur la base de l'axe et le filtre spécifiés dans l'étape $\mathrm{step_i}$ et
- K_{i+1} est obtenu en mettant à jour le contexte K_i avec les étiquettes des noeuds traversés pour l'analyse de l'étape $\mathrm{step_i}$.

Utiliser le contexte permet donc d'assurer une certaine précision de l'analyse lorsque les axes `parent` et `ancs` sont exprimés ou lorsque le path analysé contient des conditions.

Afin de simplifier la compréhension des exemples illustrant l'inférence des types, nous représentons la DTD $D_{\texttt{Sol}}$ sous forme d'un graphe dont les noeuds correspondent aux étiquettes de $D_{\texttt{Sol}}$

et dont l'arc de n_1 vers n_2 capture le fait qu'une étiquette n_2 est générée par le symbole n_1 de $D_{\texttt{Sol}}$. Cette représentation est donnée par la Figure 3.4 ci-dessous.

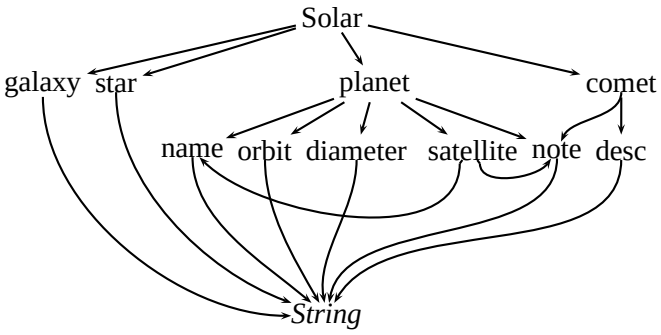


FIGURE 3.4 – Graphe représentant la DTD $D_{\texttt{Sol}}$

La fonction $\mathcal{A}xe_D(L, Axis)$, où L est un ensemble d'étiquettes, utilisée dans la présentation de la méthode d'inférence, permet de retourner les étiquettes de D qui sont liées aux étiquettes de L par la relation $Axis$. Cette fonction est illustrée au moyen de l'exemple suivant.

Exemple 26 ($\mathcal{A}xe_D(L, Axis)$**).**

$\mathcal{A}xe_{D_{\texttt{Sol}}}(\{\text{Solar}\}, \texttt{child})$	=	{galaxy, star, planet,comet}
$\mathcal{A}xe_{D_{\texttt{Sol}}}(\{\text{satellite}\}, \texttt{desc})$	=	{name, note}
$\mathcal{A}xe_{D_{\texttt{Sol}}}(\{\text{note}\}, \texttt{parent})$	=	{planet,satellite, comet}
$\mathcal{A}xe_{D_{\texttt{Sol}}}(\{\text{name}\}, \texttt{ancs})$	=	{planet, comet, Solar}

■

Nous illustrons maintenant l'inférence avec l'exemple suivant.

Exemple 27. Considérons le chemin P donné par :

self::Solar/`child`::planet/`child`::note/`parent`::`node`()

L'inférence du type de P pour $D_{\texttt{Sol}}$ est initialisée avec la paire $(\{Solar\}, \{Solar\})$ où l'étiquette S est utilisée dans cet exemple pour indiquer le symbole de départ de $D_{\texttt{Sol}}$ qui génère la racine Solar. Nous illustrons l'analyse de chaque étape step_i à l'aide de la Table ci-dessous en indiquant la paire en entrée (T_i, K_i) et la paire (T_{i+1}, K_{i+1}) produite par l'analyse.

(T_i, K_i)	step	(T_{i+1}, K_{i+1})
1- ({Solar}, { Solar})	self::Solar	({Solar}, {Solar})
2- ({Solar}, {Solar})	`child`::planet	({planet }, {Solar, planet})
3- ({planet}, {Solar, planet})	`child`::note	({note}, {Solar, planet, note})
4- ({note}, {Solar, planet, note})	`parent`::`node`()	({planet}, {Solar, planet})

L'analyse de la première étape qui cible le noeud Solar s'effectue comme suit :

– T_1 est obtenu à partir de Solar qui correspond à l'étiquette du seul noeud ciblé par cette étape,

– K_1 est obtenu en étendant K_0 par T_1 pour marquer le chemin suivi par chaque étape.

L'analyse des étapes $\texttt{child}::\mathrm{planet}$ et $\texttt{child}::\mathrm{note}$ se déroule de manière analogue et n'a pas besoin d'être commentée.

L'analyse de l'étape $\texttt{parent}::\texttt{node}()$ permet de montrer le rôle joué par le contexte dans la précision de l'inférence des types lorsque les axes arrière sont employés. Cette analyse se déroule comme suit :

– T_4 est obtenu en pré-sélectionnant les étiquettes des noeuds atteints par P en utilisant la fonction $\mathcal{A}xe$ comme suit : $\mathcal{A}xe_{D_{\texttt{Sol}}}(\{\text{note}\}, \texttt{parent}) = \{\text{planet, satellite, commet}\}$. Le résultat retourné par cette dernière contient les étiquettes de tous les noeuds parent de note tels que donnés par $D_{\texttt{Sol}}$. Ce résultat ne tient pas compte des noeuds traversés par P pour atteindre le noeud note de ce chemin. Le contexte K_3 est alors utilisé pour restreindre les étiquettes obtenues par l'application de $\mathcal{A}xe$ aux étiquettes des noeuds traversés par P, i.e $T_4 = \mathcal{A}xe_{D_{\texttt{Sol}}}(\{note\}, \texttt{parent}) \cap K_3$.

– K_4 est obtenu de manière analogue à T_4 en pré-sélectionnant les étiquettes des parents de chaque étiquette dans K_3 et en restreignant le résultat obtenu à K_3 pour n'avoir que les étiquettes des noeuds traversés par P.

Le résultat de l'inférence du type pour P retourne bien l'étiquette planet du noeud ciblé par P et le contexte $\mathrm{S}, \mathrm{Solar}, \mathrm{planet}$ de ce noeud. ■

3.3.1.1.0.1 Correction de l'inférence des types Afin d'énoncer le résultat de la correction de l'inférence des types donné dans [BCCN06], nous introduisons les notations suivantes où P est un chemin, D est une DTD et t est un document.

- $\texttt{Type}(D, P)$ désigne l'ensemble des étiquettes des noeuds ciblés par P, i.e $\texttt{Type}(D, P) = T$ où (T, K) est le résultat de l'inférence du type de P pour D,
- $\texttt{PLab}(t, P)$ désigne l'ensemble des étiquettes des noeuds retournés par l'évaluation de P sur t, i.e $\texttt{PLab}(t, P) = \{lab_t(\mathbf{i}) \mid \mathbf{i} \in PEval(t, P)\}$ où $PEval(t, P)$ capture l'évaluation du chemin P sur le document t (cf. Chapitre 2).

Théorème 1 (Correction de l'inférence des types pour les chemins [BCCN06])**.**
Soit D une DTD et P un path. Alors pour tout document $t \in D$, on a :

$$\texttt{PLab}(t, P) \subseteq \texttt{Type}(D, P)$$

3.3.1.2 Inférence du projecteur pour les chemins

L'inférence du projecteur pour un chemin P et une DTD D, capturée par la fonction $\texttt{Navig}(D, P)$, consiste à extraire les étiquettes de tous les noeuds traversés par P. Intuitivement, un noeud est traversé par P s'il est ciblé par au moins une étape de P. On pourrait donc penser

que pour avoir un projecteur pour P, il suffit de collecter le type inféré pour chaque étape de P. Le projecteur obtenu suivant cette méthode est correct puisqu'il permet de capturer les étiquettes de tous les noeuds traversés par P. Cependant, il se peut qu'il ne soit pas précis dans le cas ou certaines étapes contiennent le test `node()` qui force l'inférence des types à générer des étiquettes de noeuds qui ne sont pas nécessairement traversés par P.

Exemple 28. Considérons le chemin P_2 =self::Solar/`child`::`node`()/`child`::satellite pour lequel on s'intéresse à inférer un projecteur en utilisant la DTD $D_{\texttt{Sol}}$.

Le projecteur "minimal" pour P_2 et $D_{\texttt{Sol}}$ est donné par $\{\text{Solar}, \text{planet}, \text{satellite}\}$. Ce projecteur contient Solar l'étiquette du symbole de départ de $D_{\texttt{Sol}}$, planet l'étiquette du noeud qui est traversé par l'étape `child`::`node`() et satellite l'étiquette du noeud retourné par ce chemin (type inféré pour $D_{\texttt{Sol}}$ et P_2).

Considérons maintenant la méthode qui consiste à inférer le projecteur pour P_2 en collectant les étiquettes des noeuds ciblés par chacune de ses étapes. L'ensemble des étiquettes des noeuds ciblés par chaque étape de P_2 est donné comme suit :

Etape analysée	**Type inféré**
self::Solar	{Solar}
`child`::`node`()	{galaxy, star, planet, comet}
`child`::satellite	{satellite}

Le projecteur inféré suivant cette méthode est alors donné par :
$\{\text{Solar}, \text{galaxy}, \text{star}, \text{planet}, \text{comet}, \text{satellite}\}$
qui contient en plus des étiquettes du projecteur "minimal" les étiquettes galaxy, star et comet dont les noeuds ne sont jamais traversés par P_2. ■

La technique de [BCCN06] permet d'inférer des projecteurs précis pour les chemins. Elle repose sur un mécanisme qui permet de filtrer les étiquettes des noeuds qui sont ciblés par des étapes intermédiaires du chemin analysé mais pas traversés par ce dernier. Cela consiste à identifier l'ensemble des étiquettes productives et à l'utiliser dans l'analyse de chaque étape du chemin pour lequel on veut inférer le projecteur. Etant donné un chemin P de la forme $\text{step}_1/\ldots/\text{step}_\text{n}$, l'ensemble des étiquettes productives pour le sous-chemin $\text{step}_{\text{i}+1}/\ldots/\text{step}_\text{n}$ consiste en l'ensemble des étiquettes des noeuds ciblés par step_i qui permettent d'atteindre les noeuds ciblés par le sous-chemin $\text{step}_{\text{i}+1}/\ldots/\text{step}_\text{n}$. Cet ensemble est obtenu en restreignant les étiquettes des noeuds ciblés par step_i à celles qui permettent d'inférer un type non vide pour $\text{step}_{\text{i}+1}/\ldots/\text{step}_\text{n}$. Pour le chemin P_2 de l'Exemple 28, l'ensemble des étiquettes productives pour le sous-chemin formé par l'étape `child`::satellite est {planet} puisque planet est le seul noeud pouvant mener au noeud satellite ciblé par l'étape `child`::satellite. Le projecteur pour le chemin P est inféré en collectant les noeuds productifs de chacune de ses étapes step_i.

Exemple 29. Considérons le chemin P_2 de l'exemple 28. L'inférence du projecteur pour P_2 est illustrée pour chaque étape comme suit :

Etape analysée	**Etiquettes productives**	**Projecteur inféré**
self::Solar	{Solar}	{Solar}
`child`::`node`()	{planet}	{Solar, planet}
`child`::satellite	{satellite}	{Solar, planet, satellite}

On a : $\texttt{Navig}(D_{\texttt{Sol}}, P_2) = \{Solar, planet, satellite\}$. ■

Notons que le projecteur inféré pour un chemin contient forcément le type de ce chemin. Ceci est énoncé par la propriété ci-dessous.

Propriété 5 ([BCCN06])**.**
Soit D une DTD et P un chemin. Alors on a :

$$\texttt{Type}(D, P) \subseteq \texttt{Navig}(D, P)$$

3.3.1.2.0.2 Correction de l'inférence du projecteur

Théorème 2 (Correction de l'inférence du projecteur [BCCN06])**.**
Soit D une DTD et P un chemin. Soit $\pi = \texttt{Navig}(D, P)$ le projecteur inféré à partir de D et P. Alors on a :

$$PEval(t, P) = PEval(\pi(t), P)$$

Ce Théorème énonce que l'évaluation de P sur le document original t retourne le même résultat que l'évaluation sur la projection $\pi(t)$.

3.3.2 Précision de la projection basées sur les schémas

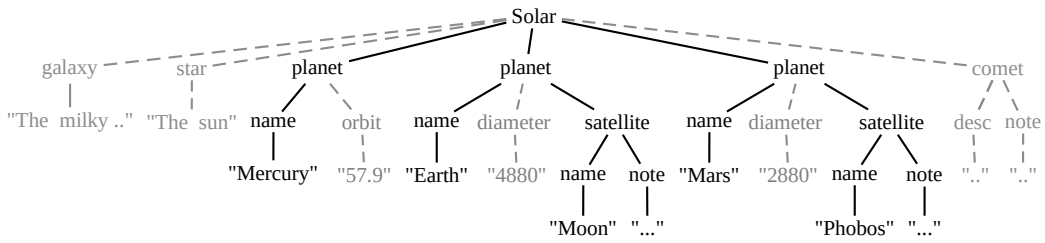
Le but de cette section est de discuter de la précision de la projection en la technique [BCCN06]. Pour ce faire, nous allons réutiliser l'exemple de la Figure 3.3 donnée en page 62 pour lequel nous rappelons la requête Q et le projecteur associé π. Nous reproduisons le document t pour faciliter la compréhension de cette discussion.

let $x := self :: Solar//`node`()[note]/satellite
return $x

La requête Q

$\pi = \{\text{Solar}, \text{planet}, \text{satellite}, \text{note}, \text{name}, \text{String}\}$

Le projecteur π



Le document t

Le premier point que nous abordons concerne les noeuds qui sont projetés comme effet de bord de la projection d'autres noeuds. C'est le cas du noeud name fils de planet qui, bien que jamais accédé par la requête Q, il est projeté puisque l'étiquette name appartient au projecteur. Pour ce cas, l'étiquette name est générée dans le projecteur π puisqu'il s'agit de l'étiquette d'un descendant de $\mathrm{satellite}$. Rappelons que l'extraction des chemins suivant la méthode de [BCCN06] retourne le chemin $\mathrm{self} :: \mathrm{Solar}//\mathtt{node}()[\mathrm{note}]/\mathrm{satellite}//\mathtt{node}()$ pour qui l'inférence du projecteur génère les étiquettes de tous les noeuds descendants de $\mathrm{satellite}$.

Cette imprécision est due au fait que la DTD $D_{\mathtt{Sol}}$ ayant servie à inférer le projecteur est ambiguë pour les parents (l'étiquette name est utilisée dans planet et $\mathrm{satellite}$). Cependant, il subsiste une solution permettant, pour ce cas, d'éviter de se retrouver à projeter le noeud name fils de planet. Cette solution consiste à modifier la méthode de projection de sorte à pouvoir projeter le sous-élément de $\mathrm{satellite}$ directement, et ce, sans avoir besoin de générer les étiquettes de ses descendants dans le projecteur.

Le deuxième point concerne la projection des textes pour laquelle on a le même type de problème que ci-dessus. La projection des textes se fait via l'utilisation du label *String*. Dans le cas où *String* est généré dans le projecteur, il suffit que le parent d'un texte donné soit projeté pour que ce texte le soit aussi. C'est le cas des textes $\mathrm{"Mercury"}$, $\mathrm{"Earth"}$ et $\mathrm{"Mars"}$ qui sont projetés comme effet de bord de la projection de leur parent name. Remédier à cette anomalie est encore plus important que pour le cas des noeuds puisque, en pratique, les textes occupent, le plus souvent, plus d'espace que les noeuds et sont fréquents. Là encore, la solution qui peut être envisagée pour améliorer la précision de la projection consiste à marquer les noeuds parents des textes devant être projetés et à modifier la méthode de projection de sorte à ne projeter les textes que si leur parents sont marqués.

3.4 Conclusion et Bilan

Dans ce chapitre nous avons présenté deux techniques de projection pour les requêtes. La première est proposée par [MS03]. Elle s'appuie sur une analyse statique permettant d'extraire les chemins exprimés dans les requêtes dans le but de s'en servir pour élaguer le document interrogé lors de son chargement. Cette technique est confrontée aux problèmes de l'évaluation de chemins XPath au vol et s'avère inapplicable dans le cas où les chemins utilisent des conditions ou des axes arrière. La seconde est proposée par [BCCN06]. Elle ne souffre pas des limites de la méthode précédente grâce à l'utilisation du schéma du document interrogé. Elle est meilleure que la technique de [MS03] sur différents points. En terme de faisabilité, la technique de [BCCN06] permet de traiter tout type de chemin puisque la projection repose sur un projecteur spécifié par un ensemble de types

(étiquettes), évitant ainsi d'être confrontée aux limitations de l'évaluation des chemins au vol. En terme de précision, la technique de [BCCN06] permet souvent d'obtenir une projection plus "fine" que celle obtenue par [MS03]. C'est pour ces deux raisons que la technique de projection pour les mises à jour que nous présentons dans la Chapitre 4 s'inspire de la méthode de [BCCN06].

La fin de ce chapitre est consacrée à l'identification des outils et résultats développés dans [BCCN06] et que nous utilisons pour notre travail sur les mises à jour et en particulier pour les preuves.

Les expressions de mises à jour sont construites à partir de requêtes, elles mêmes construites à partir de chemins. Inférer un projecteur pour une mise à jour u nécessite donc d'extraire les chemins qui sont exprimés dans u et d'inférer pour chacun de ces chemins son projecteur suivant [BCCN06]. Nous faisons certains choix lors de la conception du projecteur pour les mises à jour qui nous obligent à distinguer, pour chaque chemin extrait d'une mise à jour, les étiquettes des noeuds qu'il cible, càd le type T inféré pour ce chemin, et les étiquettes des noeuds qu'il traverse, ces étiquettes sont incluses dans le projecteur N de ce chemin. Dans [BCCN06] le type et le projecteur pour un chemin sont inférés séparément par deux techniques différentes : pour un chemin P et une DTD D,

- $\texttt{Type}(D, P)$ permet d'inférer le type de P et
- $\texttt{Navig}(D, P)$ permet d'inférer le projecteur pour P.

Afin de faciliter la manipulation, pour les mises à jour, des résultats de ces deux techniques, nous introduisons la notion de path-projecteur pour les chemins. Un path-projecteur pour un chemin P et une DTD D est une paire $\mathbf{pi}$ d'ensembles d'étiquettes (T, N) où :

- T est le type inféré pour le chemin P et la DTD D, et
- N est le projecteur inféré pour le chemin P et la DTD D.

Rappelons d'après la propriété 5 que $T \subseteq N$.

Ces path-projecteurs $\mathbf{pi}=(T, N)$ vont servir comme briques de base dans les preuves. Entre autre, ils vont servir à calculer la trace de la navigation d'un chemin P sur un document t valide pour la DTD D. La fonction $KPath(t, \mathbf{pi})$ définie ci-dessous permet d'extraire, sur la base des étiquettes de $\mathbf{pi}$, les identifiants des noeuds de t potentiellement traversés ou ciblés par P.

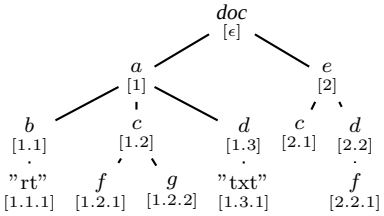
Définition 15 ($KPath(t, \mathbf{pi})$).
Soit un path-projecteur $\mathbf{pi}=(T, N)$. Pour tout document $t=(r_t, \sigma_t)$ tel que *subfor*$(t)=f$, l'id-set $KPath(t, \mathbf{pi})$ est défini récursivement par :

1. si *lab*$(r_t) \notin N$ alors $KPath(t, \mathbf{pi})=\emptyset$,
2. si *lab*$(r_t) \in N$ alors $KPath(t, \mathbf{pi})=\{r_t\} \cup KPath(f, \mathbf{pi})$ avec :
   - $KPath(f, \mathbf{pi}) = \emptyset$ si $f = ()$, sinon supposons que $f = t' \cdot f'$,
   - $KPath(f, \mathbf{pi}) = KPath(t', \mathbf{pi}) \cup KPath(f', \mathbf{pi})$

Cette définition assure (item 1) que les identifiants retournés soient tous connectés à la racine et donc que la projection du document t sur $KPath(t, \mathbf{pi})$ est un arbre. Notons aussi que le calcul de l'id-set $KPath(t, \mathbf{pi})$ est déterminé exclusivement par la composante N du path-projecteur, puisque c'est le projecteur.

doc	→	$(a \mid e)*$
a	→	*b*,c*,d ?*
b	→	*String*
c,d	→	$(f \mid g \mid \textit{String})*$
e	→	*c*,d ?*

La DTD D_5



Le document t_5

Exemple 30. Considérons la DTD D_5 et le document $t_5 \in D_5$ donnés ci-dessous.

Considérons le chemin : $\mathrm{P}_1 = \mathrm{self} :: \mathrm{doc}//\mathrm{d}/\texttt{node}()$.
Le path-projecteur $\mathbf{pi}_1{=}(T_1, N_1)$ pour P_1 et D_5 est donné par $(\{f, g, \textit{String}\}, \{doc, a, e, d, f, g, \textit{String}\})$.

$KPath(t_5, \mathbf{pi}_1) = \{\epsilon, 1, 1.3, 1.3.1, 2, 2.2, 2.2.1\}$.

Remarquez que $1.2.1$ et $1.2.2 \notin KPath(t, \mathbf{pi}_1)$ car $c \notin N_1$. ■

Dans le path-projecteur $\mathbf{pi}$, le type T a été conservé parce qu'il permet d'isoler les identifiants cibles potentiels du chemin P dont nous aurons aussi besoin. Ceci fait d'ailleurs l'objet de la prochaine définition. La fonction $KNode(t, \mathbf{pi})$ permet d'extraire à partir d'un document t, sur la base des étiquettes de $\mathbf{pi}$, les identifiants des noeuds de t potentiellement ciblés par P.

Définition 16 ($KNode(t, \mathbf{pi})$).
Soit un path-projecteur $\mathbf{pi}{=}(T, N)$. Pour tout document $t{=}(r_t, \sigma_t)$ tel que *subfor*$(t){=}f$, l'id-set $KNode(t, \mathbf{pi})$ est défini récursivement par :

1. si *lab*$(r_t){\notin}N$ alors $KNode(t, \mathbf{pi}){=}\emptyset$,
2. si *lab*$(r_t){\in}N{-}T$ alors $KPath(t, \mathbf{pi}){=}KNode(f, \mathbf{pi})$,
3. si *lab*$(r_t){\in}T$ alors $KNode(t, \mathbf{pi}){=}\{r_t\} \cup KNode(f, \mathbf{pi})$,

avec :

- $KNode(f, \mathbf{pi}) = \emptyset$ si $f = ()$, sinon supposons que $f = t' \cdot f'$,
- $KNode(f, \mathbf{pi}) = KNode(t', \mathbf{pi}) \cup KNode(f', \mathbf{pi})$

La fonction $KNode(t, \mathbf{pi})$ est définie de manière très similaire à la fonction $KPath(t, \mathbf{pi})$. La différence est que $KNode(t, \mathbf{pi})$ "parcourt" le document en fonction des types de N (l'item 1 assure que le parcours est effectué de père en fils) mais ne sélectionne que les identifiants correspondants à des noeuds typés par T.

Exemple 31. Considérons l'exemple précédent. Alors $KNode(t, \mathbf{pi}_1)) = \{1.3.1, 2.2.1\}$.
Bien que $f, g \in T_1$, $1.2.1$ et $1.2.2$ n'appartiennent pas à $KNode(t, \mathbf{pi}_1))$. ■

La propriété suivante énonce que pour tout chemin P, l'id-set $KNode(t, (T, \mathbf{pi}))$ est une sur-estimation de l'évaluation du chemin P sur le document t lorsque $\mathbf{pi}$ est le path-projecteur inféré pour P.

Lemme 1 (Correction du path-projecteur [BCCN06])**.**
Soit une DTD D, un chemin P et un document t valide pour D. Soit $\mathbf{pi}$ le path-projecteur inféré pour P et D. Alors on a :

$$PEval(t, P) \subseteq KNode(t, \mathbf{pi}), \text{et}$$

$$\mathit{Ancs}(t, PEval(t, P)) \subseteq KPath(t, \mathbf{pi})$$

■

La démonstration de ce résultat est une conséquence immédiate des résultats de [BCCN06]. Nous ne présentons pas cette preuve. L'exemple ci-dessous illustre ce résultat.

Exemple 32. Considérons toujours le même exemple que précédemment mais cette fois avec le chemin P_2 spécifié par $\mathrm{self} :: \mathrm{doc}//\mathrm{f}/\texttt{parent}::\texttt{node}()$.
Le path-projecteur $\mathbf{pi}_2$ pour P_2 et D_5 est donné par : $(\{c, d\}, \{doc, a, e, c, d, f\})$.

On a : $KNode(t, \mathbf{pi}_2) = \{1.2, 2.1, 2.2\}$ alors que $PEval(t, P_2) = \{1.2, 2.2\}$.

Pour cet exemple, on a donc : $PEval(t, P_2) \subseteq KNode(t, \mathbf{pi}_2)$ ■

Chapitre 4

Projection pour les mises à jour XML

Introduction

Les moteurs mémoire-centrale développés à l'origine pour l'interrogation ont été étendus pour permettre la mise à jour des documents XML. L'utilisation de ces moteurs pour les mises à jour se heurte au même problème rencontré pour l'interrogation concernant leur incapacité à traiter des documents volumineux. Des tests que nous avons réalisés sur les principaux moteurs mémoire-centrale eXist [exi], QizX [qizb] et Saxon [saxb] qui ont permis de conclure que la taille maximale des documents pouvant être mis à jour par ces moteurs est 580MB. La technique de projection introduite dans le cadre de l'optimisation des requêtes s'avère utile dans le cadre de l'optimisation des mises à jour XML.

Aucune des deux méthodes d'évaluation de requêtes utilisant la projection présentées dans le chapitre précédent ne peut être utilisée directement et ceci pour deux raisons. La première raison est évidente et concerne l'analyse statique développée par ces approches pour inférer leur projecteur. Cette analyse est destinée aux requêtes pures et ne tient pas compte de la syntaxe des mises à jour ni de leur sémantique. La deuxième raison concerne le scénario même de la projection pour les requêtes. Ce scénario composé d'une phase de génération du projecteur pour la requête Q, d'une phase de projection du document t se termine par l'application de la requête Q sur la projection t' de t pour produire $Q(t)$=$Q(t')$. Ce scénario est incomplet pour les mises à jour, parce que l'application d'une mise à jour u sur la projection t' de t ne peut pas produire $u(t)$ i.e. parce que $u(t') \neq u(t)$. Intuitivement, toutes les parties de t qui ont été élaguées dans t' (parce que non touchées ou non utilisées par la mise à jour u) n'apparaissent pas dans $u(t')$.

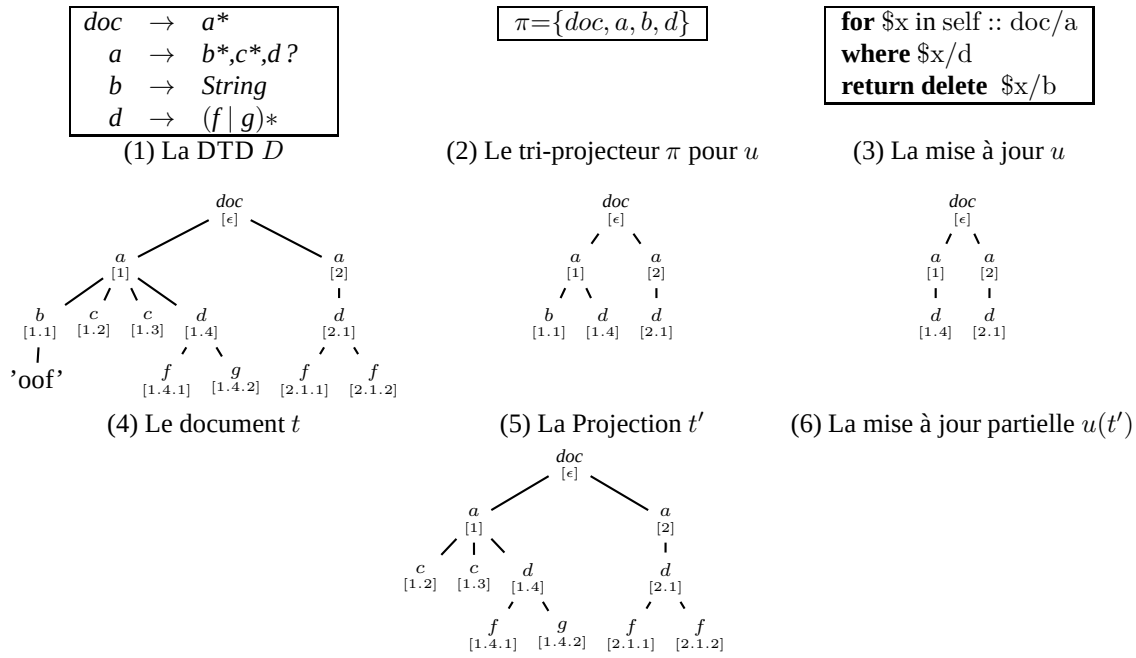
Dans [BBC$^+$11], nous avons proposé une technique d'évaluation de mises à jour utilisant le principe de projection. Cette technique redéfinit le scénario de l'évaluation des requêtes en introduisant une étape supplémentaire dont le but est de construire le résultat final $u(t)$. Cette technique définit un projecteur bien sûr adapté à la sémantique des mises à jour, mais aussi adapaté à la phase finale du scénario que nous allons explicité rapidement.

Ce chapitre présente la technique développée dans [BBC$^+$11]. Le plan de ce chapitre est le suivant. Le scénario de l'évaluation de mises à jour par projection est tout d'abord présenté de manière informelle au travers d'exemples. Chaque étape du scénario est ensuite formellement spécifiée. Pour plus de lisibilité, les preuves de la correction de notre méthode sont développées en annexe.

4.1 Motivation

Cette section est destinée à introduire "la projection pour les mises à jour" à l'aide d'exemples.

Exemple 33. Dans un premier temps, nous allons tenter d'utiliser le scénario développé pour les requêtes afin de comprendre en quoi il est incomplet pour traiter les mises à jour. Nous allons faire ceci en considérant la DTD D, la mise à jour u et le document t de la Figure 4.1. La première étape de ce scénario consiste à inférer le projecteur π à partir de la mise à jour u. Cette inférence extrait et analyse les chemins de u qui sont ici $\mathrm{self :: doc/a}, \mathrm{self :: doc/a/d}, \mathrm{self :: doc/a/b}$. Ces chemins permettent d'inférer le projecteur π donné dans la Figure 4.1-(2) suivant la technique développée dans [BCCN06] et qui consiste à extraire les étiquettes traversées par chacun des chemins ci-dessus. La deuxième étape consiste à utiliser le projecteur π pour élaguer le document t et produit la projection t'.
La dernière étape consiste à appliquer la mise à jour u sur la projection t' produisant la mise à jour de la projection $u(t')$. On voit ici immédiatement que ce document est différent de $u(t)$, en particulier il ne contient pas les noeuds élagués par π. Il est donc nécessaire de définir un mécanisme permettant de rétablir dans leur position d'origine les noeuds élagués par le projecteur π.



(1) La DTD D (2) Le tri-projecteur π pour u (3) La mise à jour u

(4) Le document t (5) La Projection t' (6) La mise à jour partielle $u(t')$

(7) Le résultat final $u(t)$

FIGURE 4.1 – Un exemple de motivation pour les mises à jour

■

4.1.1 Scénario de l'évaluation des mises à jour par projection

Le scénario que nous proposons s'inspire du scénario pour les requêtes. On suppose donnés une DTD D, une mise à jour u et un document t valide relativement à D.

1. Le projecteur π_u est inféré pour u et D, en analysant les chemins exprimés dans u,

2. le document t est projeté, au vol, lors de son chargement, en utilisant le projecteur π_u, produisant la projection $\pi_u(t)$,

3. la mise à jour u est évaluée sur la projection $\pi_u(t)$ et produit le document partiellement mis à jour $u(\pi_u(t))$, appelé par la suite mise à jour partielle

4. le résultat final $u(t)$ est obtenu en fusionnant le document original t avec la mise à jour partielle $u(\pi_u(t))$.

La dernière étape de ce scénario (étape de fusion) permet de récupérer, dans leur position d'origine, les noeuds ayant été élagués lors de la projection de t. Pour des raisons d'efficacité, cette étape est effectuée en streaming et s'appuie uniquement sur la position des noeuds visités et sur le projecteur π_u. Il est important de souligner que :

1. cette étape ne requiert aucune modification de la mise à jour partielle $u(\pi_u(t))$ pour être effectuée,
2. elle ne nécessite pas de réécrire la mise à jour u.

Avant d'illustrer la fusion du document original t avec la mise à jour partielle $u(\pi(t))$ présentés de l'exemple 33, nous introduisons les hypothèses considérées par notre scénario concernant les identifiants des noeuds. Pour cela, rappelons que $t@\mathbf{i}$ désigne le noeud de t identifié par $\mathbf{i}$. Chaque noeud de t est identifié par sa position $(1, 1.1, \ldots)$ donnée entre deux crochets, suivant le *document-order*. Dans la projection $\pi(t)$ de t, l'identifiant d'un noeud projeté est conservé et peut ne plus correspondre à la position de ce noeud dans le document $\pi(t)$. Ceci est le cas du noeud $\pi(t)@1.4$ qui est le deuxième fils de $\pi(t)@1$. Dans la mise à jour partielle $u(\pi(t))$, des identifiants sont affectés aux "nouveaux" noeuds insérés par u soit via un *insert* ou un *replace*. Ces nouveaux identifiants ne véhiculent aucune information sur la position des noeuds.

Nous procédons à l'illustration de la technique de fusion sur les documents t et $u(\pi(t))$ de notre exemple.

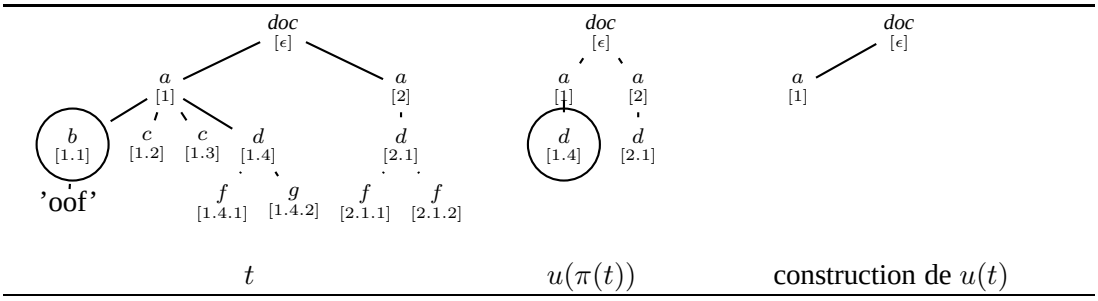


FIGURE 4.2 – Parsing de $t@1.1$ et $u(\pi(t))@1.4$

Exemple 34. La fusion de t et $u(\pi(t))$ se fait sur la base des parcours (on utilisera le mot parsing parfois) simultanés de ces deux documents en profondeur à gauche (parcours suivant le *document-order*). La fusion de la racine $t@\epsilon$ de t avec la racine $u(\pi(t))@\epsilon$ de $u(\pi(t))$ est simple : $t@\epsilon$ a été projeté et correspond à $u(\pi(t))@\epsilon$, donc le résultat de la fusion est le noeud $u(\pi(t))@\epsilon$. Le processus de fusion est identique pour les deux noeuds suivants, i.e $t@1$ et $u(\pi(t))@1$: la fusion retourne le noeud $u(\pi(t))@1$. Le parsing parallèle de t et $u(\pi(t))$ amène donc maintenant à examiner les

noeuds $t@1.1$ et $u(\pi(t))@1.4$. Le processus de fusion pour ce cas est illustré par la Figure 4.2 ci-dessous. Ce processus se base sur les observations suivantes :

- l'étiquette b du noeud $t@1.1$ appartient à π indiquant que ce noeud a été projeté,
- la comparaison des positions 1.1 et 1.4 des deux noeuds permet d'identifier que le noeud $t@1.1$ projeté a été supprimé.

L'examen des noeuds $t@1.1$ et $u(\pi(t))@1.4$ ne produit rien de nouveau dans le résultat et le parcours se poursuit uniquement sur le document t conduisant à l'examen des noeuds $t@1.2$ et $u(\pi(t))@1.4$. Dans ce cas, le processus de fusion se base sur le fait que l'étiquette c n'appartient pas à π indiquant que le noeud $t@1.2$ a été élagué et doit être introduit dans le résultat de la fusion. L'examen de $t@1.2$ et $u(\pi(t))@1.4$ produit donc le noeud $t@1.2$ et le parsing de t uniquement est poursuivi.

Le traitement des noeuds $t@1.3$ et $u(\pi(t))@1.4$ est identique à celui du cas précédent : le noeud $t@1.3$ qui n'a pas été projeté est retourné, le parsing de t se poursuit.

L'examen des noeuds $t@1.4$ et $u(\pi(t))@1.4$ permet d'observer qu'ils ont la même position 4 et le même label d. Dans ce cas, le processus de fusion rend $u(\pi(t))@1.4$ et le parsing est alors poursuivi à la fois sur t et $u(\pi(t))$ (avec les fils de $t@1.4$ et les fils de $u(\pi(t))@1.4$).

Le processus de fusion se poursuit jusqu'à épuisement du parcours et examen des deux documents t et $u(\pi(t))$. Le résultat de la fusion est le document $u(t)$ présentés dans la Figure 4.1-(7). ■

Il est important de remarquer que la fusion s'appuie uniquement sur la position des noeuds et sur le projecteur π. Le projecteur de cet exemple est dérivé de la mise à jour u suivant [BCCN06]. Le but de la discussion qui suit est de montrer que cette méthode d'inférence du projecteur n'est pas appropriée pour les mises à jour, et en particulier qu'elle ne permet pas d'assurer que une étape de fusion réussie.

4.1.2 Traitement des insertions

Afin de comprendre en quoi la présence d'insertions oblige à modifier le projecteur de [BCCN06], considérons l'exemple suivant qui se base sur la DTD de la Figure 4.1.

Exemple 35. Soit u_1 la mise à jour donnée par :
for $\$x$ in $\mathrm{self} :: \mathrm{doc/a}$
return insert <e>'new'</e> **as last into** $\$x$

Le projecteur π_1 extrait pour u_1 est obtenu en analysant $\mathrm{self} :: \mathrm{doc/a}$ le (seul) path extrait de u_1 ce qui donne $\pi_1 = \{doc, a\}$. La projection $\pi_1(t)$ et la mise à jour partielle $u_1(\pi_1(t))$ sont données dans la Figure 4.3. Rappelons que les identifiants des noeuds de t sont conservés dans la projection $\pi_1(t)$ et que de nouveaux identifiants sont attribués aux noeuds insérés par u_1. Dans la Figure 4.3 ci-dessous, **i** et **i′** correspondent à de nouveaux identifiants.

Focalisons-nous sur la fusion de t avec $u_1(\pi_1(t))$ qui a pour objectif de produire le résultat final $u_1(t)$. Après examen des noeuds racines des deux documents et des noeuds $t@1$ et $u_1(\pi_1(t))@1$, les

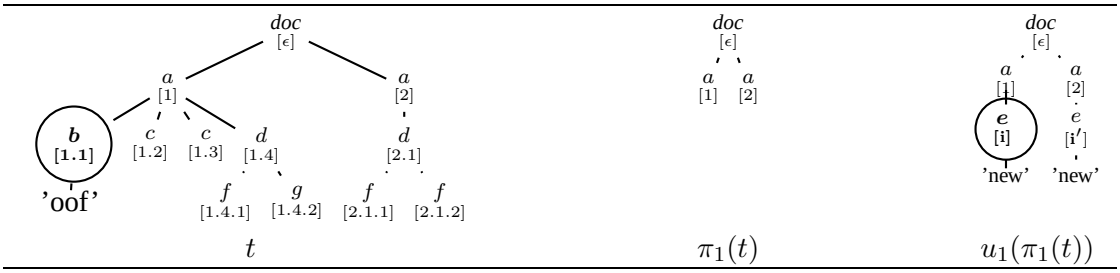


FIGURE 4.3 – Traitement des insertions : projection insuffisante

prochains noeuds examinés sont $t@1.1$ et $u_1(\pi_1(t))@\mathbf{i}$ entourés par un cercle dans la figure. L'identifiant **i** ne véhicule aucune information sur la position du nouveau noeud inséré $u_1(\pi_1(t))@\mathbf{i}$ relativement à ses voisins. Il devient alors impossible de décider quel noeud parmi $t@1.1$ et $u_1(\pi_1(t))@\mathbf{i}$ doit être retourné en premier dans le résultat de la fusion. Il est important de rappeler l'hypothèse selon laquelle aucune information sur la mise à jour u_1 n'est disponible pendant la fusion. ■

Afin de résoudre le problème relatif aux insertions, nous proposons de modifier le projecteur en considérant le fait que le chemin $\mathrm{self} :: \mathrm{doc/a}$ sélectionne des noeuds en dessous desquels les éléments vont être insérés. L'idée clé ici est de projeter ces noeuds mais également **tous leurs fils** afin de résoudre le problème de placement des éléments insérés lors de la fusion.

Ainsi, dans la suite le projecteur va être subdivisé en deux composants $\pi_{\mathbf{no}}$ et $\pi_{\mathbf{olb}}$. Lors de la projection l'un des cas suivants peut survenir :

- cas où un noeud est étiqueté par un type de $\pi_{\mathbf{no}}$: ce noeud est projeté seul, ses fils seront projetés en fonction du projecteur,
- cas où un noeud est étiqueté par un type de $\pi_{\mathbf{olb}}$: ce noeud est projeté bien sûr et tous ses fils sont également projetés même si leur étiquette n'appartient ni à $\pi_{\mathbf{no}}$ ni à $\pi_{\mathbf{olb}}$.

Exemple 36. Pour notre exemple, la projection de t en utilisant π_{u_1} défini par $\pi_{\mathbf{no}} = \{doc\}$ et $\pi_{\mathbf{olb}} = \{a\}$ est illustrée dans la Figure 4.4. Maintenant, notons que le nouveau noeud est inséré dans une projection contenant tous ses noeuds frères. Il en résulte qu'il devient possible pour la fusion de s'exécuter de manière correcte.

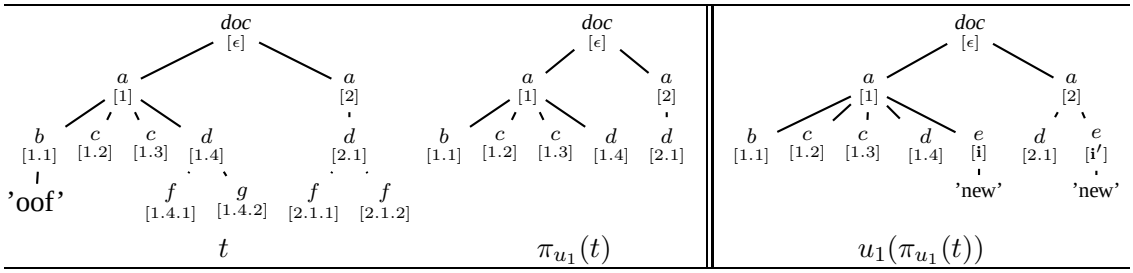


FIGURE 4.4 – Traitement des insertions : nouvelle projection

■

4.1.3 Précision du projecteur : discussion

Le projecteur π_{u_1} peut amener à projeter plus de noeuds qu'il n'est nécessaire à la réalisation de la mise à jour partielle et de l'étape de fusion (une discussion plus élaborée sera proposée en conclusion de cette partie). Toutefois, il est important de souligner que la stratégie utilisée reste raisonnable et qu'en particulier elle permet de projeter, pour l'exemple, les noeuds étiquetés b, c ou d sans inclure ces étiquettes dans le projecteur.
En fait, une stratégie alternative purement syntaxique aurait pu être choisie. Cette alternative consiste à transformer le chemin $\mathrm{self} :: \mathrm{doc/a}$ en $\mathrm{self} :: \mathrm{doc/a/}\texttt{parent::node()/child::node()}$ pour inférer le projecteur :

$$\pi'_{u_1} = \{doc, a, b, c, d\}.$$

Pour l'exemple, utiliser π'_{u_1} ou π_{u_1} pour projeter t produit le même résultat puisqu'avec les deux projecteurs, tous les fils des noeuds $t@1$ et $t@2$ sont projetés.

En revanche, si on modifie la DTD D en remplaçant la règle

$$d \to (f \mid g)* \qquad \text{par} \qquad d \to (b \mid c)*,$$

le projecteur π_{u_1} projète moins de noeuds que le projecteur π'_{u_1}. Dans la Figure 4.5, on voit que les noeuds étiquetés b et c et qui sont fils du noeud d ne sont pas projetés par π_{u_1} alors qu'ils le sont par π'_{u_1}. D'ailleurs, π'_{u_1} n'élague aucun noeud du document t, on a $\pi'_{u_1}(t)$=t.

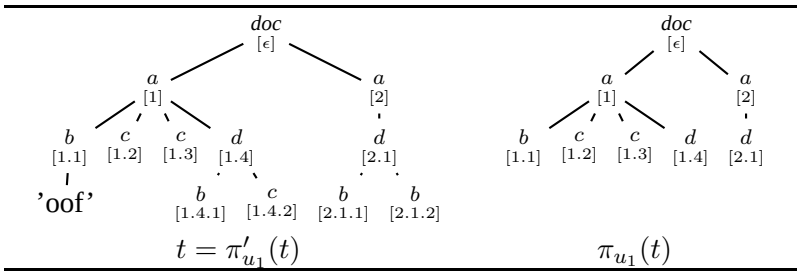


FIGURE 4.5 – Traitement des insertions : approche syntaxique vs projection à deux niveaux.

4.1.4 Traitement des noeuds mixed-content

Le problème traité ici concerne la projection de textes et en particulier le traitement de noeuds dits *mixed-content* dans la jargon XML. Un noeud *mixed-content* peut avoir des fils textes et des fils élements. Le problème pouvant survenir lors de la projection des fils textes d'un noeud mixed-content est la concaténation de ces derniers si les noeuds qui les séparent ne sont pas projetés.

Exemple 37. Afin d'illustrer ce problème sur l'exemple en cours, nous allons modifier la DTD D en remplaçant la règle pour b par $b \to (\mathit{String} \mid c)*$ et considérer la mise à jour u_2 suivante :

for $\$x$ in $\mathrm{self} :: \mathrm{doc/a}$
where $\$\mathrm{x/b/text()}$= 'foot'
return delete $\$\mathrm{x/d}$

Les chemins spécifiant les noeuds nécessaires à l'évaluation de u_2 sont :

self :: doc/a/b/text() et self :: doc/a/d.

Les étiquettes inférées à partir de ces paths sont données par :

$\pi_2 = \{doc, a, b, \textit{String}, d\}$.

Considérons le document t_2 de la Figure 4.6. La projection de t_2 relativement à π_2 a pour effet de bord la concaténation des noeuds textes 'fo' et 'ot' ce qui va permettre à la mise à jour u_2 de supprimer le noeud $\pi_2(t_2)@1.4$ (puisque la condition exprimée dans le **where** est vérifiée). Il faut rappeler que lors de l'étape de fusion, on s'interdit de modifier les noeuds des documents t_2 et $u_2(\pi_2(t))$. Il est donc impossible de produire le résultat final $u_2(t_2)$.

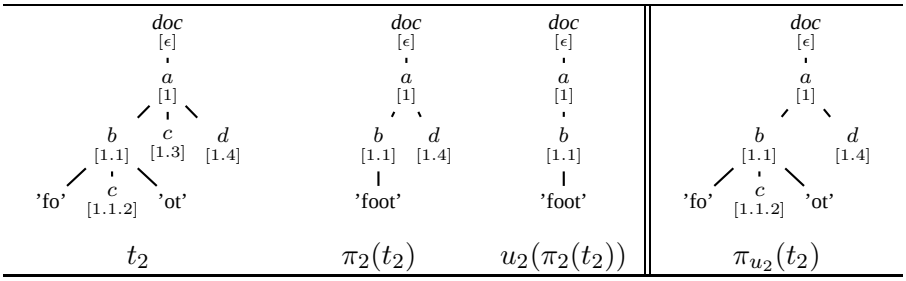


FIGURE 4.6 – Projection des textes : problème lié au mixed-content

■

Pour résoudre ce problème, lié à la projection des noeuds mixed-content, tous les noeuds fils d'un noeud mixed-content sont projetés en ayant recours, comme précédemment, à la différentiation des étiquettes du projecteur en deux niveaux $\pi_{\mathbf{no}}$ et $\pi_{\mathbf{olb}}$.

Dans l'exemple 37, b est l'étiquette d'un mixed-content, elle sera donc insérée dans $\pi_{\mathbf{olb}}$. Par conséquent, un noeud étiqueté b sera projeté ainsi que tous ses fils. Pour cet exemple, le projecteur π_{u_2} est spécifié par :

$\pi_{\mathbf{no}} = \{doc, a, d\}$ et $\pi_{\mathbf{olb}} = \{b\}$.

La projection de t_2 suivant π_{u_2} est présentée dans la Figure 4.6.

Nous entamons une discussion relative à la précision du projecteur similaire à celle que nous avons eu pour le cas des insertions. Le projecteur π_{u_2} peut conduire à projeter davantage que nécessaire. Cependant, la stratégie utilisée reste raisonnable pour les mêmes raisons évoquées précédemment : π_{u_2} permet de projeter le noeud étiqueté c sans inclure l'étiquette c dans le projecteur.

Là encore, on aurait pu envisager une stratégie alternative purement syntaxique consistant à transformer le chemin self :: doc/a/b/text() en self :: doc/a/b/text()/`parent`::`node`()/`child`::`node`() pour inférer le projecteur $\pi'_2 = \{doc, a, b, \textit{String}, c, d\}$. Le projecteur π'_2 permet de traiter correctement la mise à jour partielle et la fusion. Cependant, il est moins précis que π_{u_2} et cela se voit ici car le noeud $t@1.3$ est projeté par π'_2 alors qu'il ne l'est pas par π_{u_2}.

4.1.5 Cas particulier des noeuds textes

La stratégie que nous proposons pour projeter les textes d'un mixed-content est particulièrement intéressante pour restreindre la projection des textes dans le cas général. Notons que le type *String* n'appartient ni à $\pi_{\mathbf{no}}$ ni à $\pi_{\mathbf{olb}}$. Le seul moyen pour projeter un texte consiste donc maintenant à mettre le type de son parent dans $\pi_{\mathbf{olb}}$. Nous illustrons cette remarque au moyen de l'exemple suivant.

Exemple 38. Considérons la DTD D pour laquelle la règle pour d est $d \rightarrow$ *String*. Considérons aussi la même mise à jour u_2 et le document t_3 de la Figure 4.7. Le projecteur π''_2 inféré pour u_2 et la nouvelle DTD à partir des chemins $\mathrm{self :: doc/a/b/text()/}$`parent::node()/child::node()` et $\mathrm{self :: doc/a/d}$ est donné par :

$\pi''_2 = \{doc, a, b, d, \textit{String}\}$.

Ce projecteur permet de générer la mise à jour partielle et de réaliser la fusion de manière correcte. Cependant, ce projecteur n'est pas précis puisqu'appliqué sur un document dont les noeuds étiquetés d ont des fils de type *String* va avoir pour effet de projeter inutilement ces derniers. C'est le cas du document t_3 de la Figure 4.7 pour lequel $\pi''_2(t_3) = t_3$.

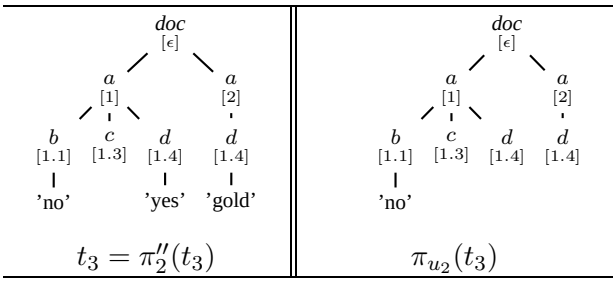


FIGURE 4.7 – Projection des textes seuls en utilisant $\pi_{\mathbf{olb}}$.

■

La stratégie que nous introduisons pour la projection des textes peut être utilisée dans le contexte des requêtes pures pour lequel elle apporte un gain significatif relativement à la stratégie proposée dans [BCCN06]. En effet, les noeuds textes constituent, en général, une part importante des documents XML. Pouvoir restreindre la projection des textes à ceux qui sont réellement utilisés par les requête est donc crucial pour la réduction de la taille des documents projetés.

La discussion qui suit concerne un autre cas pour lequel notre méthode de projection permet un traitement plus efficace que la méthode de [BCCN06]. Il s'agit de l'extraction d'éléments. Cette opération est omniprésente dans les requêtes puisque l'évaluation de celles-ci consiste à retourner tous les descendants des noeuds ciblés par les chemins. Elle est aussi utile pour les mises à jour, en particulier les insertions et remplacements qui permettent de copier des éléments à des emplacements spécifiques.

Pour comprendre en quoi la méthode de [BCCN06] est insuffisante pour inférer un projecteur précis dans le cas de l'extraction d'éléments, considérons l'exemple suivant.

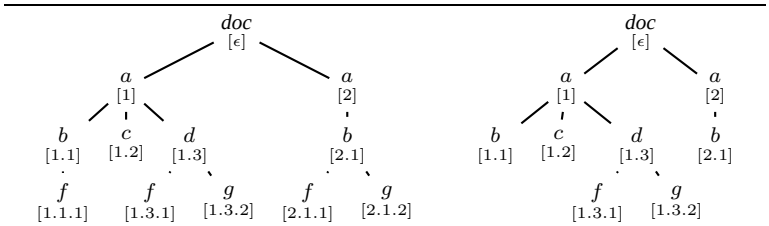
Exemple 39. Soit la DTD D_2, le document t_4 et la mise à jour u_3 donnés dans la Figure 4.8.

doc	→	*a**	*a*	→	*b*,c*,d ?*
b	→	$(f \mid g)*$	*d*	→	$(f \mid g)*$

(1) La DTD D_2.

for $\$x$ in self : :doc/a
return replace $\$x$/b with $\$x$/d

(2) La mise à jour u_3.



(3) $t_4 = \pi_3(t_4)$ (4) $\pi_{u_3}(t_4)$

FIGURE 4.8 – Traitement des insertions : approche syntaxique vs projection à deux niveaux.

Intuitivement, la mise à jour de remplacement se traite comme une suppression suivie d'une insertion. Donc, le chemin self :: doc/a/b spécifiant la cible de la mise à jour doit être considéré comme pour le cas de l'insertion, un chemin ciblant des noeuds devant être projetés avec tous leurs fils. Par conséquent, le composant $\pi_{\mathbf{olb}}$ du projecteur π_3 pour u_3 et D doit contenir l'étiquette a du parent du noeud ciblé par ce chemin i.e $\pi_{\mathbf{olb}}$=$\{a\}$. Le chemin self :: doc/a/d retourne les éléments copiés par la mise à jour u_3 à l'emplacement indiqué par le chemin self :: doc/a/b. Par conséquent, les éléments retournés par ce chemin doivent être projetés.

La première solution qui s'offre à nous pour inférer un projecteur pour la mise à jour u_3 consiste à suivre l'approche de [BCCN06] pour la projection des chemins *returned* (cf. discussion du Chapitre 3 concernant la précision de la projection basée sur les schémas). Cette approche consiste à considérer self :: doc/a/d comme chemin returned, et de ce fait, d'étendre ce dernier pour donner self :: doc/a/d/`desc-or-self`::`node`(). Le but de l'ajout de l'étape `desc-or-self`::`node`() au chemin self :: doc/a/d est de projeter tous les sous-arbres enracinés aux noeuds étiquetés par d en générant les étiquettes f, g dans le projecteur. Le projecteur π_3 obtenu pour u_3 et la DTD D' serait alors :

$$\pi_{\mathbf{no}} = \{doc, b, d, f, g\}, \pi_{\mathbf{olb}} = \{a\}.$$

La projection de t_4 suivant π_3 conduit à projeter tous les noeuds de ce dernier et la figure 4.8 montre que $\pi_3(t_4)$=t_4.

Par souci de générer des projecteurs les plus précis possible, nous avons opté pour une stratégie différente de celle présentée ci-dessus. Cette nouvelle stratégie consiste à introduire, en plus des composants $\pi_{\mathbf{no}}$ et $\pi_{\mathbf{olb}}$, un troisième composant au projecteur, le composant $\pi_{\mathbf{eb}}$ 'everything below' (qu'on abrège en $\forall$ below). Ce composant $\pi_{\mathbf{eb}}$ contient des étiquettes qui vont déclencher la projection de tout le sous-arbre enraciné par un noeud de type $\forall$ below.

Le projecteur π_{u_3} que nous proposons pour la mise à jour u_3 est spécifié par :

$\pi_{\mathbf{no}} = \{doc, b\},\ \ \pi_{\mathbf{olb}} = \{a\},\ \ \pi_{\mathbf{eb}} = \{d\}.$

Il permet de projeter tous les descendants du noeud $t_4@1.3$ sans entraîner la projection des descendants du noeud b qui sont etiquetés f et g. La projection de t_4 suivant π_{u_3} est présentée dans la Figure 4.8-(4). Notons ici que π_{u_3} est un sous-arbre strict de t_4. ■

Les différents exemples et la discussion ci-dessus nous amène à introduire un projecteur pour les mises à jour que nous appelons dans la suite tri-projecteur [4] constitué de trois composants, chaque composant étant un ensemble de types (étiquettes). Nous récapitulons la fonction de chacun de ces trois composants :

- $\pi_{\mathbf{no}}$ le composant 'node only' contient les types des noeuds devant être projetés et dont les fils doivent être examinés et projetés en fonction du tri-projecteur,
- $\pi_{\mathbf{olb}}$ le composant 'one level below' contient les types des noeuds devant être projetés avec tous leurs noeuds fils,
- $\pi_{\mathbf{eb}}$ le composant '$\forall$ below' contient les types des noeuds devant être projetés avec tous leurs descendants.

4.2 Le tri-projecteur

Cette section est destinée à définir formellement le tri-projecteur, la projection engendrée par un tri-projecteur et à introduire les définitions utilisées dans le développement formel de notre méthode.

Nous commençons par définir le tri-projecteur qui s'appuie sur une DTD.

Définition 17 (Tri-projecteur)**.**
Soit une DTD (D, s_D) portant sur l'alphabet Σ. Un tri-projecteur π est un triplet $(\pi_{\mathbf{no}}, \pi_{\mathbf{olb}}, \pi_{\mathbf{eb}})$ tel que (π est aussi utilisé pour noter $\pi_{\mathbf{no}} \cup \pi_{\mathbf{olb}} \cup \pi_{\mathbf{eb}}$) :

i) $\pi \subseteq \Sigma$,

ii) $\pi_{\mathbf{no}}$, $\pi_{\mathbf{olb}}$ et $\pi_{\mathbf{eb}}$ sont deux à deux disjoints, et

iii) $s_D \in \pi$ et pour chaque $b \in \pi$ il existe $a \in \pi$ tel que $D(a)$=r et b apparaît dans r.

La condition i) de cette définition est triviale : elle énonce que les étiquettes du tri-projecteur appartiennent à l'alphabet de la DTD pour laquelle il est défini. La condition ii) est introduite pour le besoin de l'étape de fusion. Dans notre scénario, cette étape repose sur l'examen des noeuds du document original t et de la mise à jour partielle $u(\pi(t))$ et fait naturellement usage du tri-projecteur inféré pour u. La fusion spécifie un traitement spécifique en fonction de chaque composant de tri-projecteur étant donné que chaque composant déclenche une façon de projeter particulière. La disjonction des composants du tri-projecteur permet de simplifier le choix du le traitement à faire par la fusion. La condition iii) permet d'assurer une sorte de propriété de fermeture relative à la DTD D : une étiquette a appartenant à π doit être connectée à la racine s_D de la DTD suivant au moins un chemin.

4. Tri est utilisé ici pour faire référence aux trois composants du projecteur et non pour faire allusion à une quelconque opération de tri.

Notons que *String* n'appartient jamais à aucun composant d'un tri-projecteur. Un noeud texte ne pourra être projeté que si le type de son parent appartient au composant $\pi_{\mathbf{olb}}$ ou encore si le type d'un de ses ancêtres appartient au composant $\pi_{\mathbf{eb}}$.

L'opération de projection engendrée par un tri-projecteur est définie comme suit :

Définition 18 (Tri-projection)**.**
Soit (D, s_D) une DTD , $\pi{=}(\pi_{\mathbf{no}}, \pi_{\mathbf{olb}}, \pi_{\mathbf{eb}})$ un tri-projecteur et $t{\in}D$ un document valide pour D. On pose *roots*$(t){=}\{r_t\}$ et *subfor*$(t){=}F$.

La *projection de* t *suivant* π, notée $\pi(t)$, est le document $\Pi_{K(t,\pi)}(t)$ où $K(t, \pi)$ est l'ensemble des identifants défini récursivement par :

a. si *lab*$(r_t){\notin}\pi$ alors $K(t, \pi){=}\emptyset$,

b. si *lab*$(r_t){\in}\pi_\alpha$, où $\alpha{\in}\{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$, alors $K(t, \pi){=}\{r_t\}{\cup}K_\alpha(F, \pi)$ avec :

1. $K_\alpha(F, \pi){=}\emptyset$ si $F{=}()$ et sinon, en supposant que $F{=}t' \cdot F'$ et *roots*$(t'){=}\{r_{t'}\}$,
2. $K_{\mathbf{no}}(F, \pi){=}K(t', \pi){\cup}K_{\mathbf{no}}(F', \pi)$,
3. $K_{\mathbf{olb}}(F, \pi){=}K(t', \pi){\cup}K_{\mathbf{olb}}(F', \pi)$ si *lab*$(r_{t'}){\in}\pi$,
4. $K_{\mathbf{olb}}(F, \pi){=}\{r_{t'}\}{\cup}K_{\mathbf{olb}}(F', \pi)$ si *lab*$(r_{t'}){\notin}\pi$,
5. $K_{\mathbf{eb}}(F, \pi){=}$*dom*$(F)$.

La fonction $K(t, \pi)$ examine chaque noeud n de t pour décider s'il est projeté, projeté avec ses fils ou projeté avec ses sous-arbres. Lorsque l'étiquette de n appartient à π_α, la fonction $K_\alpha(F, \pi)$ est appelée pour traiter la sous-forêt F de n. Ce traitement diffère selon la valeur de α comme déjà annoncé :

– dans le cas où α est 'node only' (ligne 2), le traitement consiste à examiner la racine $r_{t'}$ de chaque sous-arbre t' de F. Lorsque l'étiquette de $r_{t'}$ appartient à π_α, $r_{t'}$ est retourné et sa sous-forêt f' est examinée de manière récursive.

– Les lignes 3 et 4 capturent le traitement pour le composant 'one level below' qui diffère du cas précédent par le fait que chaque racine de F doit être projetée même si son étiquette n'appartient pas à π. La ligne 4 est importante, elle montre que la projection de toutes les racines de F n'implique pas la projection de tous ses arbres : les arbres de F dont l'étiquette de la racine n'appartient pas à π sont simplement ignorés après la projection de la racine bien sûr.

– La dernière ligne de cette définition entend spécifier le traitement des noeuds '$\forall$ below'. Ce traitement est simple : il consiste à projeter tout F.

Notation. Dans la suite de ce manuscrit, nous appellerons id-projecteur tout id-set utilisé pour projeter un document. Le plus souvent, nous noterons les id-projecteurs par K.

Exemple 40. Considérons la DTD D_4 et le tri-projecteur π_4 de la Figure 4.9. Ce tri-projecteur est bien-défini puisque le type c est connecté à la racine a à travers b. L'id-projecteur K obtenu pour ce document est $\{\epsilon, 1, 1.1\}$. Remarquez que, bien que $c{\in}\pi_{\mathbf{no}}$, on a $2.1{\notin}K(t_4, \pi_4)$ parce que $e{\notin}\pi$.

■

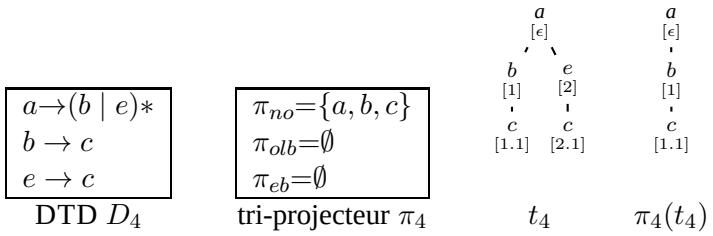


FIGURE 4.9 – Un exemple de tri-projection

4.3 Inférence du tri-projecteur associé à une mise à jour

L'inférence du tri-projecteur pour une mise à jour u et une DTD D est constituée de deux étapes :

1. extraction des chemins exprimés dans u,
2. inférence des étiquettes des noeuds traversés par les chemins extraits à partir de u, et

L'extraction des chemins à partir de u constitue le point de départ de l'inférence du tri-projecteur. Cette extraction va devoir anticiper l'objectif final qui est de construire les trois composants du tri-projecteur. Les chemins sont les ingrédients utilisés dans les mises à jour pour spécifier les noeuds "source" , respectivement "cible" ou pour encore pour spécifier la navigation requise par une mise à jour. Il est donc crucial de distinguer, lors de leur extraction, parmi les chemins ceux qui ciblent des noeuds 'node only' de ceux qui ciblent des noeuds 'one level below' ou encore des noeuds '∀ below'.

La seconde étape consiste à inférer, en utilisant la technique de [BCCN06], les étiquettes des noeuds qui sont soit traversés soit retournés par les chemins extraits de u. Ces étiquettes sont différenciées en fonction de la catégorie (node only, one level below, ...) du chemin à partir duquel elles sont inférées. Il ne reste donc qu'à collecter dans chaque composant du tri-projecteur les étiquettes des noeuds destinés à être projetés soit seuls, soit avec leurs fils, etc.

4.3.1 Extraction des chemins pour les mises à jour

Le mécanisme d'extraction des chemins pour les mises à jour repose, bien évidemment, sur la décomposition générale des expressions de mises à jour introduite dans le Chapitre 2. Cette décomposition est illustrée par la Figure 4.10. Brièvement, rappelons qu'une mise à jour u est composée d'une partie *contextuelle* utilisée pour la navigation sur le document t, et d'une partie action spécifiant la modification devant être effectuée sur le document t. La partie *action* peut être à son tour composée d'une requête *source* chargée de construire les éléments devant être insérés à une position donnée dans le document et d'une requête *cible* permettant de localiser les noeuds où les mises à jour primitives (insertion, suppression, remplacement et renommage) sont appliquées. Il résulte de cette structure des mises à jour que l'extraction des chemins pour les mises à jour a pour premier constituant l'extraction des chemins pour les requêtes.

L'extraction des chemins pour les requêtes a déjà été étudiée dans le cadre de l'optimisation de l'évaluation par projection. [MS03] propose une technique qui extrait deux catégories de chemins :

– les chemins *used* qui capturent la navigation ou les conditions,

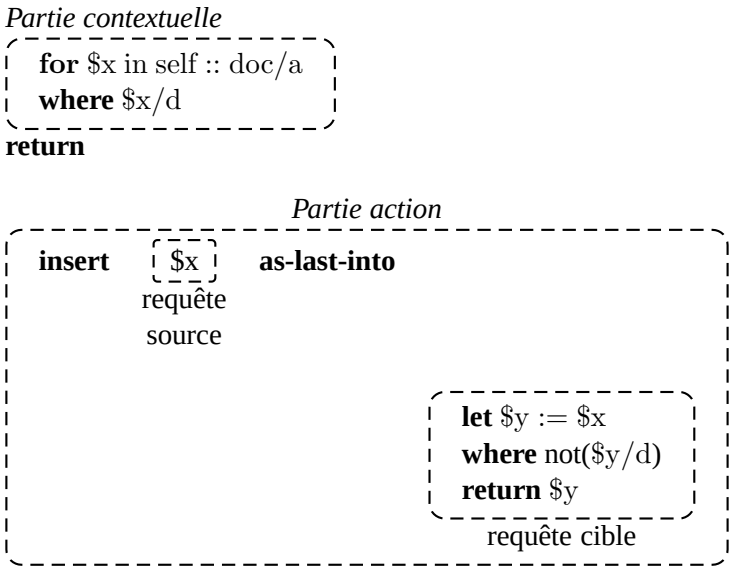


FIGURE 4.10 – Décomposition d'une mise à jour u

– les chemins *returned* qui spécifient les racines des éléments retournés par la requête.

L'analyse des mises à jour nécessite une classification plus fine des chemins extraits que celle proposée dans [MS03]. En effet, les exemples de motivation fournis en Section 4.1 montrent différentes situations pour lesquelles il est nécessaire d'identifier si un chemin cible des textes ou des noeuds, ou est utilisé dans une condition nécessitant l'extraction d'éléments du document interrogé, des informations qui ne sont pas fournies par la technique de [MS03]. Le raffinement de la classification des chemins extraits à partir d'une requête (d'une mise à jour) que nous proposons s'appuie sur la technique de [MS03] de la manière suivante :

– Concernant les chemins *used* qui servent principalement à la navigation ou pour exprimer des conditions, nous distinguons ceux qui ciblent des noeuds éléments de ceux qui ciblent des noeuds textes. Ces derniers sont appelés chemins *string used*.

  Les chemins qui ciblent des noeuds éléments sont divisés en deux catégories :
  – ceux qui ciblent des noeuds seulement utilisés par la requête (ils expriment une condition booléenne ou assurent la navigation), appelés chemins *node used* et
  – ceux qui ciblent les racines d'éléments devant être extraits parce qu'ils sont utilisés dans des conditions **where** et `if`, appelés chemins $\forall$ *below used*.

– Concernant les chemins *returned*, nous distinguons :
  – ceux qui ciblent les racines d'éléments réponses à la requête, appelés chemins *node returned*, de
  – ceux qui ciblent des noeuds texte réponses à la requête, appelés chemins *string returned*.

Exemple 41. Considérons la requête q_1 suivante.

for $\$x$ in self :: doc/a
where $x/b/text()="some" and $x/c=<c>...</c>
return ($x/d, $x/c/text())

Les chemins extraits à partir de q_1 sont indiqués dans la Figure 4.11 ci-dessous pour la technique

de [MS03] et pour notre technique.

Extraction pour [MS03]	Notre technique	Chemins
used	Node used	$\mathrm{self :: doc/a}$
	String used	$\mathrm{self :: doc/a/b/text()}$
	$\forall$ below used	$\mathrm{self :: doc/a/c}$
returned	Node returned	$\mathrm{self :: doc/a/d}$
	String returned	$\mathrm{self :: doc/a/c/text()}$

FIGURE 4.11 – Extraction des chemins pour de q_1.

Dans cet exemple, il est aisé de constater que $\mathrm{self :: doc/a}$ cible un noeud alors que $\mathrm{self :: doc/a/b/text()}$ cible un texte. Par ailleurs, on peut facilement voir que $\mathrm{self :: doc/a/c}$ est utilisé au sein d'une comparaison structurelle nécessitant d'extraire l'élément enraciné en c. ■

4.3.1.1 Analyse statique des requêtes d'une mise à jour.

Afin de déterminer la catégorie des chemins extraits pour les requêtes d'une mise à jour, notre technique d'analyse repose sur deux mécanismes. Le premier sert à distinguer entre les chemins *node used* et $\forall$ *below used*. Il consiste simplement à examiner l'emplacement de la requête contenant le chemin extrait, et en particulier à analyser le type de condition si ce chemin est utilisé dans une condition. Le second mécanisme sert principalement à distinguer les chemins ciblant des noeuds de ceux ciblant des textes. Il est plus sophistiqué que le premier du fait du besoin d'analyser les chemins avant même de décider s'ils ciblent des textes ou des noeuds.

Exemple 42. Soit la DTD D_5 de la Figure 4.12 et le chemin $\mathrm{P_1 = self :: doc/a/d/node()}$ posé sur un document valide par rapport à D_5. Il est clair que décider si P_1 cible des noeuds éléments ou des noeuds textes n'est pas possible en examinant exclusivement la syntaxe du chemin P_1 à cause de la dernière étape de ce chemin qui est `node()`. Cette étape permet de sélectionner des noeuds éléments comme des noeuds texte. ■

La solution au problème illustré dans l'exemple ci-dessus est simple. Il suffit d'utiliser l'inférence des types développé dans [BCCN06]. Cette analyse permet d'inférer avec précision les étiquettes des noeuds ciblés par un chemin en se basant sur la DTD et peut être utilisée pour déterminer si un chemin est susceptible de retourner des textes ou non.

Exemple 43. En considérant la DTD et le chemin de l'exemple 42, l'inférence des types pour P_1 retourne les étiquettes $\{f, g, \mathit{String}\}$ parmi lesquelles il y a *String*. Ceci nous permet de conclure que P_1 cible à la fois des textes et des noeuds. ■

doc	$\rightarrow$	$(a \mid e)*$
a	$\rightarrow$	*b*,c*,d ?*
b	$\rightarrow$	*String*
c,d	$\rightarrow$	$(f \mid g \mid \mathit{String})*$
e	$\rightarrow$	*c*,d ?*

FIGURE 4.12 – La DTD D_5

Formellement, nous introduisons le test $\texttt{STR}_D(P)$ qui retourne vrai lorsque P cible potentiellement des textes et faux dans le cas où P cible exclusivement des noeuds élément.

$\texttt{STR}_D(P)$	=	true	si $P = P'/Axis$::text(), ou $P = P'/Axis$::node() avec $String \in \texttt{Type}(D, P)$
$\texttt{STR}_D(P)$	=	false	sinon

Exemple 44. Considérons la DTD D_5. La table ci-dessous illustre $\texttt{STR}_D()$ pour différents chemins.

P	$\texttt{Type}(D, P)$	$\texttt{STR}_D(P)$
self :: doc/a/b/text()	pas besoin	true
self :: doc/a/d/node()	$\{f, g, \textit{String}\}$	true
self :: doc/e/node()	$\{c, d\}$	false

■

Pour rappel, une fois encore, la projection des textes s'effectue via le parent du noued texte à projeter, noeud parent qui doit être étiqueté par un type *one level below*. Ceci va entraîner, par effet de bord, la projection de tous les fils du noeud parent en plus du noeud texte.
Afin de pouvoir projeter les textes, il est nécessaire d'inférer les étiquettes de leur parent. Etant donné que les chemins peuvent accéder à des textes à partir d'étapes intermédiaires ou à l'intérieur des conditions, il n'est pas suffisant d'analyser la dernière étape des chemins pour identifier tous les accès d'un chemin à des textes. Il est donc nécessaire d'analyser les chemins étape par étape afin de générer tous les sous-chemins susceptibles de cibler des textes.

Exemple 45. Avec la même DTD que l'exemple précédent, considérons les chemins suivants :
$\mathrm{P}_2 = \mathrm{self} :: \mathrm{doc}/\texttt{desc}::\texttt{node}()/\texttt{ancs}::\mathrm{e}$, et
$\mathrm{P}_3 = \mathrm{self} :: \mathrm{doc/a[b/text()} = \text{"val"}]/\mathrm{c}$

En examinant ces deux chemins il est aisé de constater qu'ils accèdent à des textes à travers des étapes intermédiaires :

– pour P_2, il s'agit de l'étape desc::node() qui sélectionne tous les descendants de doc, et cible donc les textes générés par les symboles b, c et d de la DTD, il est donc nécessaire de générer le (sous-)chemin $\mathrm{self} :: \mathrm{doc}/\texttt{desc}::\texttt{node}()$ de P_2 comme chemin *string used*.

– pour P_3, la condition contenue dans ce chemin exprime une comparaison : le texte retourné par le chemin relatif $\mathrm{b/text}()$ est comparée à la valeur "val". Dans ce cas, il est nécessaire de concaténer le chemin formé par le préfixe $\mathrm{self} :: \mathrm{doc/a}$ et le chemin relatif $\mathrm{b/text}()$ pour générer $\mathrm{self} :: \mathrm{doc/a/b/text}()$ comme chemin *string used*. ■

Avant de présenter l'extraction des chemins à partir des requêtes d'une mise à jour, nous mettons en place quelques définitions et terminologies préliminaires.

Comme nous allons être amenés à définir les mêmes règles d'extraction pour des catégories différentes de chemins, nous introduisons les symboles **use**, **ret** et **all** pour regrouper les catégories de chemins comme suit :

- **use** désigne une des catégories parmi *node used* abrégée $\mathbf{nu}$, *string used* abrégée $\mathbf{su}$ et $\forall$ *belowused* abrégée $\mathbf{ebu}$,
- **ret** désigne une des catégories parmi *node returned* abrégée $\mathbf{nr}$, *string returned* abrégée $\mathbf{sr}$, et
- **all** désigne n'importe quelle catégorie parmi celles citées ci-dessus.

L'extraction des chemins à partir des requêtes d'une mise à jour est formalisée par des règles utilisant les jugements suivants :

$$(\Gamma, q) \rightsquigarrow_{\mathbf{all}} \boldsymbol{P} \qquad (\mathbf{QPathExt})$$

qui devra être lu comme suit : étant donné l'environnement statique Γ, l'ensemble des chemins $\boldsymbol{P}$ de la catégorie $\mathbf{all}$ sont extraits à partir de la requête q.

Afin de faciliter la lecture des règles pour l'extraction des chemins, nous utilisons la convention donnée dans la Table ci-dessous concernant les cinq catégories de chemins extraits pour les requêtes. Nous noterons les chemins de la catégorie *node returned* par $\boldsymbol{N}^{\mathtt{R}}$, les chemins de la catégorie *string returned* par $\boldsymbol{S}^{\mathtt{R}}$ etc.

catégorie	**jugement**
node returned	$(\Gamma, q) \rightsquigarrow_{\mathbf{nr}} \boldsymbol{N}^{\mathtt{R}}$
string returned	$(\Gamma, q) \rightsquigarrow_{\mathbf{sr}} \boldsymbol{S}^{\mathtt{R}}$

catégorie	**jugement**
node used	$(\Gamma, q) \rightsquigarrow_{\mathbf{nu}} \boldsymbol{N}^{\mathtt{U}}$
string used	$(\Gamma, q) \rightsquigarrow_{\mathbf{su}} \boldsymbol{S}^{\mathtt{U}}$
$\forall$ *below used*	$(\Gamma, q) \rightsquigarrow_{\mathbf{ebu}} \boldsymbol{E}$

Comme nous l'avons fait pour les jugements précédents, nous introduisons la fonction $QExt(\Gamma, q)$ comme notation alternative au jugement capturant l'extraction des chemins pour les requêtes. On a :

$$\boxed{QExt_{\mathbf{all}}(\Gamma, q) = \boldsymbol{P} \quad ssi \quad (\Gamma, q) \rightsquigarrow_{\mathbf{all}} \boldsymbol{P}}$$

L'extraction des chemins pour les requêtes est guidée par la syntaxe des requêtes donnée en Section 2.4. Les règles d'extraction des requêtes sont présentées dans le même ordre que celui de la définition de la sémantique des requêtes. Pour rappel, cet ordre est comme suit :

- les requêtes atomiques, c'est à dire :
  - ◇ les littéraux (textes) dénotés par st,
  - ◇ les variables dénotées par x,
  - ◇ les chemins $Path$ et les chemins précédés d'une variable $x/Path$, et
  - ◇ les fonctions pré-définies.
- les expressions générales qui comprennent les expressions arithmétiques, les expressions de comparaison et les expressions booléennes,
- les requêtes composées qui sont :
  - ◇ l'itération $\mathbf{for}\ x\ \mathtt{in}\ q_0\ \mathbf{return}\ q_1$,
  - ◇ le binding $\mathbf{let}\ x = q_0\ \mathbf{return}\ q_1$, et

◇ le branchement conditionnel $\mathbf{if}\ q_0\ \mathbf{then}\ q_1\ \mathbf{else}\ q_2$.

Pour alléger la lecture de ce manuscrit, nous avons décidé de placer, dans l'annexe A, les règles d'extraction pour les conditions, les étapes intermédiaires des chemins pour les fonctions pré-définies et les expressions générales. Nous donnons ici les règles d'extraction des chemins uniquement pour les requêtes atomiques et les requêtes composées.

L'extension d'un environnement statique Γ par une liaison $x{\mapsto}\boldsymbol{P}$ où $\boldsymbol{P}$ est un ensemble de chemins est notée $\Gamma \cdot x{\mapsto}\boldsymbol{P}$. Afin de faciliter la manipulation des environnements statiques dans les règles d'extraction, nous définissons ci-dessous deux fonctions $\texttt{Extend}(\Gamma, x, q_0)$ et $\texttt{Extract}(\Gamma, x)$. La première retourne un environnement statique Γ_0 étendant l'environnement Γ par une liaison pour x avec l'ensemble des chemins *string returned* et *node returned* de q. La seconde sélectionne l'ensemble des chemins liés à x dans l'environnement statique Γ. Formellement :

$\texttt{Extend}(\Gamma, x, q)$	$=$	$\Gamma \cdot x{\mapsto}\boldsymbol{N}^{\mathrm{R}} \cup \boldsymbol{S}^{\mathrm{R}}$ où $\boldsymbol{N}^{\mathrm{R}} = \mathit{QExt}_{\mathbf{nr}}(\Gamma, q)$ et $\boldsymbol{S}^{\mathrm{R}} = \mathit{QExt}_{\mathbf{sr}}(\Gamma, q)$
$\texttt{Extract}(\Gamma, x)$	$=$	$\boldsymbol{P}$ où $x{\mapsto}\boldsymbol{P} \in \Gamma$

Nous sommes maintenant prêts à présenter les règles d'extraction.

4.3.1.1.1 Requêtes atomiques : Les règles d'extraction des chemins à partir des requêtes atomiques sont regroupées dans la Figure 4.13.

Littéral Intuitivement, aucun chemin n'est extrait d'un littéral, c'est ce que spécifie la règle ($\mathbf{QPE}$:$\mathbf{literal}$).

Variable L'extraction des chemins pour une variable x est capturée par la règle ($\mathbf{QPE}$:$\mathbf{var}$). Elle consiste à récupérer dans un premier temps les chemins liés à x en utilisant la fonction $\texttt{Extract}(\Gamma, x)$. Chaque chemin P_i ainsi obtenu est analysé suivant la catégorie $\mathbf{all}$. Voici un exemple pour illustrer cette règle.

Exemple 46. Considérons la DTD D_5 donnée en Figure 4.12 page 86. Supposons que Γ contienne la liaison $x{\mapsto}\{P_1, P_2\}$ où :

- $\mathrm{P}_1 = \mathrm{self} :: \mathrm{doc/a/b[c/text()]/e}$, et
- $\mathrm{P}_2 = \mathrm{self} :: \mathrm{doc//a[e/f]}$.

L'extraction des chemins pour x produit :

- $\mathrm{self} :: \mathrm{doc/a/b/c/text()}$ comme chemin *string used* puisque P_1 accède, à travers la condition $\mathrm{[c/text()]}$ au noeud texte fils du noeud c lui-même retourné par le sous-chemin $\mathrm{self} :: \mathrm{doc/a/b}$,

$$(\mathbf{QPE{:}litteral}) \quad \frac{}{(\Gamma, st) \overset{\rightarrow}{\sim}_{\mathtt{all}} \emptyset}$$

$$(\mathbf{QPE{:}var}) \quad \frac{\mathtt{Extract}(\Gamma, x){=}\{P_1, ..., P_n\} \quad (\Gamma, P_i) \overset{\rightarrow}{\sim}_{\mathtt{all}} \boldsymbol{P}_i \; i = 1..n}{(\Gamma, x) \overset{\rightarrow}{\sim}_{\mathtt{all}} \bigcup_{i=1}^{n} \boldsymbol{P}_i}$$

$$(\mathbf{QPE{:}path}) \quad \frac{(\Gamma, \{\epsilon\}, P) \overset{\rightarrow}{\sim}_{\mathtt{all}} \boldsymbol{P}_{\mathbf{all}}}{(\Gamma, P) \overset{\rightarrow}{\sim}_{\mathtt{all}} \boldsymbol{P}_{\mathbf{all}}} \qquad (\epsilon \text{ chemin vide})$$

$$(\mathbf{QPE{:}x/P{:}use}) \quad \frac{(\Gamma, x) \overset{\rightarrow}{\sim}_{\mathtt{nr}} \boldsymbol{N}^{\mathtt{R}}_x \quad (\Gamma, x) \overset{\rightarrow}{\sim}_{\mathtt{sr}} \boldsymbol{S}^{\mathtt{R}}_x \quad (\Gamma, \boldsymbol{N}^{\mathtt{R}}_x \cup \boldsymbol{S}^{\mathtt{R}}_x, P) \overset{\rightarrow}{\sim}_{\mathtt{use}} \boldsymbol{P}^{\mathtt{U}}_P \quad (\Gamma, x) \overset{\rightarrow}{\sim}_{\mathtt{use}} \boldsymbol{P}^{\mathtt{U}}_x}{(\Gamma, x/P) \overset{\rightarrow}{\sim}_{\mathtt{use}} \boldsymbol{P}^{\mathtt{U}}_x \cup \boldsymbol{P}^{\mathtt{U}}_P}$$

$$(\mathbf{QPE{:}x/P{:}ret}) \quad \frac{(\Gamma, x) \overset{\rightarrow}{\sim}_{\mathtt{nr}} \boldsymbol{N}^{\mathtt{R}}_x \quad (\Gamma, x) \overset{\rightarrow}{\sim}_{\mathtt{sr}} \boldsymbol{S}^{\mathtt{R}}_x \quad (\Gamma, \boldsymbol{N}^{\mathtt{R}}_x \cup \boldsymbol{S}^{\mathtt{R}}_x, P) \overset{\rightarrow}{\sim}_{\mathtt{ret}} \boldsymbol{P}^{\mathtt{R}}_P}{(\Gamma, x/P) \overset{\rightarrow}{\sim}_{\mathtt{ret}} \boldsymbol{P}^{\mathtt{R}}_P}$$

FIGURE 4.13 – Règles d'extraction pour les requêtes atomiques.

- self :: doc//a/e/f comme chemin *node used* puisque ce chemin cible un noeud dont P_2 veut tester l'existence à travers la condition [e/f],
- self :: doc/a/b/e et self :: doc//a comme chemins *node returned*, ces chemins sont obtenus en éliminant les conditions des chemins initiaux P_1 et P_2.

■

Chemins – Chemins préfixés par une variable La règle (**QPE:path**) constitue le point d'entrée vers les règles d'analyse des chemins avec condition. L'analyse complète des chemins peut être consultée dans l'annexe A.

Notons ici que cette règle surcharge le jugement $(\Gamma, P) \overset{\rightarrow}{\sim}_{\mathtt{all}} \boldsymbol{P}_{\mathbf{all}}$ par le jugement $(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, P) \overset{\rightarrow}{\sim}_{\mathtt{all}} \boldsymbol{P}_{\mathbf{all}}$ où $\boldsymbol{P}_{\mathtt{ctx}}$ est un ensemble de chemins. Dans la règle (**QPE:path**), l'ensemble $\boldsymbol{P}_{\mathtt{ctx}}$ est un singleton contenant le chemin vide. Intuitivement, étant donné un chemin P, par exemple de la forme $step_1[cond_1]/.../step_i[cond_i]/.../step_n[cond_n]$, l'extraction pour P au moment de l'analyse de la ième étape $step_i[cond_i]$ nécessite de récupérer le chemin extrait $step_1/.../step_{i-1}$: ceci est fait grâce à l'ensemble $\boldsymbol{P}_{\mathtt{ctx}}$. Ceci explique qu'au départ de l'analyse de P, l'ensemble $\boldsymbol{P}_{\mathtt{ctx}}$ est initialisé avec $\{\epsilon\}$.

Cette surcharge d'écriture pour le jugement $(\Gamma, P) \overset{\rightarrow}{\sim}_{\mathtt{all}} \boldsymbol{P}_{\mathbf{all}}$ est également utilisée pour les chemins préfixés par une variable. Intuitivement, étant donné un chemin de la forme x/P, l'extraction pour x/P, au moment de l'analyse de P, nécessite de récupérer l'ensemble des chemins *returned* extraits pour x pour les concaténer (intuitivement) aux chemins extraits de P : ceci est fait grâce à

l'ensemble $\boldsymbol{P}_{\texttt{ctx}}$.

L'extraction des chemins *returned* pour x/P est capturée par la règle ($\mathbf{QPE{:}x/P{:}ret}$) qui est construite comme suit :

- extraction des chemins *node returned* de x,
- extraction des chemins *string returned* de x,
- passage de l'union des chemins *returned* extrait pour x à l'analyse du chemin P pour s'en servir comme de préfixes.

L'extraction des chemins *used* pour x/P, capturée par la règle ($\mathbf{QPE{:}x/P{:}use}$) est légèrement différente parce qu'elle extrait de x les chemins de la catégorie *use* pour les ajouter au résultat de l'analyse du chemin P qui elle, comme précédemment, se sert des chemins *returned* extraits de x comme préfixes.

Exemple 47. Considérons que Γ contienne la liaison $x{\mapsto}\{P_3, P_4\}$ où :

- $\mathrm{P}_3 = \mathrm{self} :: \mathrm{doc/a/b}$ et
- $\mathrm{P}_4 = \mathrm{self} :: \mathrm{doc//c/text()}$

D'après la règle ($\mathbf{QPE{:}var}$), les chemins extraits pour x sont : P_3 chemin *node returned* et P_4 chemin *string returned*. Considérons P un chemin (relatif) donné par $\texttt{ancs}{::}\mathrm{c/e}$. Alors, les chemins retournés pour x/P sont :

$\mathrm{self} :: \mathrm{doc/a/b/}\texttt{ancs}{::}\mathrm{c/e}$ et $\mathrm{self} :: \mathrm{doc//c/text()/}\texttt{ancs}{::}\mathrm{c/e}$ comme chemins *node returned*. ■

4.3.1.1.2 Requêtes composées L'extraction des chemins pour les requêtes composées suit le même principe que celui de la technique de [MS03] présentée dans le Chapitre 3. Ce principe est rappelé ci-dessous.

- Les chemins *used* spécifiant les noeuds exclusivement utilisés par la requête sont obtenus à la fois à partir des chemins *used* extraits de la requête contextuelle $q_{\texttt{ctxt}}$ et des chemins *used* extraits de la requête résultat $q_{\texttt{res}}$. Pour la requête d'itération (**for** x `in` $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$) et la requête de liaison (**let** x = $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$), il est aussi nécessaire de considérer les chemins *returned* extraits de la requête contextuelle.
- Les chemins *returned* spécifiant le résultat de la requête sont quant à eux obtenus exclusivement à partir des chemins *returned* de la requête résultat $q_{\texttt{res}}$.

Nous illustrons ce principe à l'aide de l'exemple suivant.

Exemple 48. Considérons la requête q donnée par :

for $x

in self : :doc/a/b
where $x/c/`text()`='txt'

$q_{\texttt{ctxt}}$

return

let $y= self : :doc//`node()`
return $y/f

$q_{\texttt{res}}$

Les chemins *used* de q sont :

– self :: $\mathrm{doc/a/b/c/}\texttt{text}()$ qui est un chemin *used* de $q_{\texttt{ctxt}}$,

– self :: $\mathrm{doc/a/b}$ qui est un chemin *returned* de $q_{\texttt{ctxt}}$, et

– self :: $\mathrm{doc//}\texttt{node}()$ qui est un chemin *returned* de $q_{\texttt{res}}$.

■

Comme expliqué au début de cette section, en plus de déterminer si les chemins extraits sont *used* ou *returned*, notre analyse va mettre en place les moyens de distinguer entre les chemins *node used*, *string used*, $\forall$ *below used* , *node returned* et *string returned*.

Nous présentons d'abord l'extraction des chemins pour la construction d'élément <a>q</a>. Intuitivement, les noeuds nécessaires à cette construction sont ceux nécessaires à l'évaluation de q. Les chemins extraits pour <a>q</a> sont exactement ceux extraits pour q comme indiqué par la règle ci-dessous.

$$(\mathbf{QPE{:}query{-}const}) \quad \frac{(\Gamma, q) \rightsquigarrow^{\rightarrow}_{\alpha} \boldsymbol{P}_{\alpha}}{(\Gamma, \text{<}a\text{>}q\text{</}a\text{>}) \rightsquigarrow^{\rightarrow}_{\alpha} \boldsymbol{P}_{\alpha}}$$

Les règles d'extraction des chemins pour les requêtes d'itération sont données par la Figure 4.14. Ces règles ne présentent pas de difficulté particulière et ne sont pas commentées. Notons que pour ces règles, nous considérons que $\Gamma_{\texttt{ctxt}}$ est obtenu en enrichissant l'environnement statique Γ en liant x aux chemins extraits de $q_{\texttt{ctxt}}$, i.e

$\Gamma_{\texttt{ctxt}} = \texttt{Extend}(\Gamma, x, q_{\texttt{ctxt}})$.

L'extraction des chemins pour les requêtes de binding $\mathbf{let}\ x = q_{\texttt{ctxt}}\ \mathbf{return}\ q_{\texttt{res}}$ suit le même principe que l'extraction des chemins pour les requêtes d'itération. Ces règles ne sont pas présentées ici.

L'extraction des chemins pour les requêtes de branchement conditionnel diffère du cas des requêtes d'itération et de binding principalement par l'utilisation faite de l'environnement statique. Comme les requêtes sont bien formées, les variables définies dans $q_{\texttt{ctxt}}$ ne sont ni utilisées ni définies dans $q_{\texttt{res1}}$ ou $q_{\texttt{res2}}$. C'est pourquoi l'environnement statique sous lequel est réalisée l'extraction des chemins pour ces deux requêtes n'est pas modifié.

Les règles d'extraction pour le branchement conditionnel sont données dans la Figure 4.15. On omet la règle pour les chemins *string used* dont le principe est exactement le même que celui de l'extraction des chemins *node used* spécifiée par la règle ($\mathbf{QPE{:}if{:}nu}$).

4.3.1.2 Correction de l'inférence des chemins pour les requêtes

La correction de l'inférence des chemins pour les requêtes s'appuie sur la notion de correction des id-projecteurs pour l'évaluation des requêtes. Nous considérons la correction aux sens suivants :

Pour les règles ci-dessous, on pose :
$\Gamma_{\texttt{ctxt}} = \texttt{Extend}(\Gamma, x, q_{\texttt{ctxt}})$

$$(\textbf{QPE:for:nu}) \quad \frac{(\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}^{\texttt{U}}_{\texttt{ctxt}} \quad (\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{nr}} \boldsymbol{N}^{\texttt{R}}_{\texttt{ctxt}} \quad (\Gamma_{\texttt{ctxt}}, q_{\texttt{res}}) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}^{\texttt{U}}_{\texttt{res}}}{(\Gamma, \textbf{for } x \texttt{ in } q_{\texttt{ctxt}} \textbf{ return } q_{\texttt{res}}) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}^{\texttt{U}}_{\texttt{ctxt}} \cup \boldsymbol{N}^{\texttt{R}}_{\texttt{ctxt}} \cup \boldsymbol{N}^{\texttt{U}}_{\texttt{res}}}$$

$$(\textbf{QPE:for:su}) \quad \frac{(\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{su}} \boldsymbol{S}^{\texttt{U}}_{\texttt{ctxt}} \quad (\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{sr}} \boldsymbol{S}^{\texttt{R}}_{\texttt{ctxt}} \quad (\Gamma_{\texttt{ctxt}}, q_{\texttt{res}}) \rightsquigarrow_{\texttt{su}} \boldsymbol{S}^{\texttt{U}}_{\texttt{res}}}{(\Gamma, \textbf{for } x \texttt{ in } q_{\texttt{ctxt}} \textbf{ return } q_{\texttt{res}}) \rightsquigarrow_{\texttt{su}} \boldsymbol{S}^{\texttt{U}}_{\texttt{ctxt}} \cup \boldsymbol{S}^{\texttt{R}}_{\texttt{ctxt}} \cup \boldsymbol{S}^{\texttt{U}}_{\texttt{res}}}$$

$$(\textbf{QPE:for:ebu}) \quad \frac{(\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_{\texttt{ctxt}} \quad (\Gamma_{\texttt{ctxt}}, q_{\texttt{res}}) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_{\texttt{res}}}{(\Gamma, \textbf{for } x \texttt{ in } q_{\texttt{ctxt}} \textbf{ return } q_{\texttt{res}}) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_{\texttt{ctxt}} \cup \boldsymbol{E}_{\texttt{res}}}$$

$$(\textbf{QPE:for:ret}) \quad \frac{(\Gamma_{\texttt{ctxt}}, q_{\texttt{res}}) \rightsquigarrow_{\texttt{ret}} \boldsymbol{P}^{\texttt{R}}_{\texttt{res}}}{(\Gamma, \textbf{for } x \texttt{ in } q_{\texttt{ctxt}} \textbf{ return } q_{\texttt{res}}) \rightsquigarrow_{\texttt{ret}} \boldsymbol{P}^{\texttt{R}}_{\texttt{res}}}$$

FIGURE 4.14 – Règles d'extraction : itération.

$$(\textbf{QPE:if:nu}) \quad \frac{(\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}^{\texttt{U}}_{\texttt{ctxt}} \quad (\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{nr}} \boldsymbol{N}^{\texttt{R}}_{\texttt{ctxt}} \quad (\Gamma, q_{\texttt{res}i}) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}^{\texttt{U}}_{\texttt{res}i} \;\; i = 1,2}{(\Gamma, \textbf{if } q_{\texttt{ctxt}} \textbf{ then } q_{\texttt{res1}} \textbf{ else } q_{\texttt{res2}}) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}^{\texttt{U}}_{\texttt{ctxt}} \cup \boldsymbol{N}^{\texttt{R}}_{\texttt{ctxt}} \cup \boldsymbol{N}^{\texttt{U}}_{\texttt{res1}} \cup \boldsymbol{N}^{\texttt{U}}_{\texttt{res2}}}$$

$$(\textbf{QPE:if:ebu}) \quad \frac{(\Gamma, q_{\texttt{ctxt}}) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_{\texttt{ctxt}} \quad (\Gamma, q_{\texttt{res}i}) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_{\texttt{res}i} \;\; i = 1,2}{(\Gamma, \textbf{if } q_{\texttt{ctxt}} \textbf{ then } q_{\texttt{res1}} \textbf{ else } q_{\texttt{res2}}) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_{\texttt{ctxt}} \cup \boldsymbol{E}_{\texttt{res1}} \cup \boldsymbol{E}_{\texttt{res2}}}$$

$$(\textbf{QPE:if:ret}) \quad \frac{(\Gamma, q_{\texttt{res}i}) \rightsquigarrow_{\texttt{ret}} \boldsymbol{P}^{\texttt{R}}_{\texttt{res}i} \;\; i = 1,2}{(\Gamma, \textbf{if } q_{\texttt{ctxt}} \textbf{ then } q_{\texttt{res1}} \textbf{ else } q_{\texttt{res2}}) \rightsquigarrow_{\texttt{ret}} \boldsymbol{P}^{\texttt{R}}_{\texttt{res1}} \cup \boldsymbol{P}^{\texttt{R}}_{\texttt{res2}}}$$

FIGURE 4.15 – Règles d'extraction : branchement conditionnel

- correction globale (plus simplement, correction)
  Dans ce cas la notion de correction est relative à la réponse obtenue par évaluation de la requête, réponse constituée d'un id-seq I_q et d'un store σ_q décrivant les éléments réponses.
- correction pour la pré-évaluation
  Dans ce cas, la notion de correction est définie relativement aux seules racines I_q des réponses à la requête.
- correction pour la matérialisation
  Dans ce cas, la notion de correction est définie relativement aux copies des réponses à la requête.

Définition 19 (Correction des Id-projecteurs pour les requêtes)**.**
Soit q une requête et $t{=}(r_t, \sigma_t)$ un document. Soit K un id-set. Posons :

$QEval(t,q){=}(\sigma_q, \mathrm{I}_q) \qquad QEval(\Pi_K(t), q){=}(\widehat{\sigma_q}, \widehat{\mathrm{I}_q}), et$

$QCopy(t,q){=}(\sigma_c, \mathrm{I}_c) \qquad QCopy(\Pi_K(t), q){=}(\widehat{\sigma}_c, \widehat{\mathrm{I}}_c).$

– L'id-set K est un id-projecteur *(globalement) correct pour l'évaluation* de q relativement à t

ssi $(\sigma_t \cdot \sigma_q, \mathrm{I}_q) \simeq (\Pi_K(\sigma_t) \cdot \widehat{\sigma_q}, \widehat{\mathrm{I}_q})$.

- Si $\mathrm{I}_q \subseteq \mathit{dom}(\sigma_t)$, l'id-set K est un id-projecteur *correct pour la pré-évaluation* de q relativement à t ssi $\mathrm{I}_q = \widehat{\mathrm{I}_q}$.
- L'id-set K est un id-projecteur *correct pour la matérialisation* des réponses à q relativement à t ssi $(\sigma_c, \mathrm{I}_c) \simeq (\widehat{\sigma_c}, \widehat{\mathrm{I}_c})$.

La notion de correction pour la pré-évaluation d'une requête q est introduite en prévision du traitement des requêtes cibles et des requêtes contextuelles d'une mise à jour. Par exemple, une requête cible localise les noeuds du document où une mise à jour primitive est appliquée. A priori, toutes les réponses à une requête cible sont des éléments du document sur lequel elle est évaluée (il n'y a pas d'éléments réponses construits) d'où la pré-condition $\mathrm{I}_q \subseteq \mathit{dom}(\sigma_t)$ (rappelons que d'après l'hypothèse $\mathbf{Cond}_{\mathtt{cib}}$ donnée dans le Chapitre 2 en page 42 soit $\mathrm{I}_q{=}()$ soit $\mathrm{I}_q = \{\mathbf{i}_{\mathtt{cib}}\}$ avec $\mathbf{i}_{\mathtt{cib}} \in \mathit{dom}(\sigma_t)$). Intuitivement, cette notion énonce donc que cette localisation est identique que la requête soit exécutée sur le document t ou qu'elle le soit sur le document projeté $\Pi_K(\sigma_t)$.

La notion de correction pour l'évaluation d'une requête q est simple. Intuitivement, elle énonce que l'évaluation de q sur le document initial t est équivalente à l'évaluation de q sur le document projeté $\Pi_K(t)$. Ici, q est une requête quelconque et peut construire des éléments réponses dans le store σ_q dont les racines appartiendront à I_q. D'ailleurs, la Propriété 1 (page 27) nous dit que $\mathrm{I}_q \subseteq \mathit{dom}(\sigma_t) \cup \mathit{roots}(\sigma_q)$ d'où le besoin de considérer, dans la condition de la définition, le store initial σ_t (resp. $\Pi_K(\sigma_t)$) et le store auxiliaire σ_q (resp. $\widehat{\sigma_q}$) obtenu par l'évaluation de q sur t (resp. $\Pi_K(t)$).

La notion de correction pour la matérialisation des réponses à une requête q est introduite essentiellement pour des raisons techniques. Ici les réponses à q qu'elles soient dans le document t ou dans le store σ_q sont matérialisées par copie. Ainsi, la propriété 2 (page 28) assure que $\mathrm{I}_c{=}\mathit{roots}(\sigma_c)$ (resp. $\widehat{\mathrm{I}_c}{=}\mathit{roots}(\widehat{\sigma_c})$) et c'est pourquoi dans la définition il n'est pas nécessaire de considérer le store initial σ_t (resp. $\Pi_K(\sigma_t)$).

La propriété suivante établit un lien entre les trois notions de correction.

Propriété 6.
Avec les hypothèses de la Définition 19, si K est un id-projecteur correct pour q relativement à t alors :

(i) si $\mathrm{I}_q \subseteq \mathit{dom}(\sigma_t)$, K est un id-projecteur correct pour la pré-évaluation de q

(ii) K est un id-projecteur correct pour la matérialisation des réponses à q

La réciproque du point (i) est fausse : un id-projecteur K correct pour la pré-évaluation de q n'est pas nécessairement correct pour l'évaluation de q puisque la condition de la Définition 19 ne garantit pas que les identifiants des réponses de q soient inclus dans K.

Exemple 49. Considérons u une mise à jour de la forme $\mathbf{delete}\ q$. Bien que nous n'ayons pas encore présenté les règles d'extraction pour les mises à jour, il est facile de comprendre que la pré-évaluation de la mise à jour de suppression ne requiert les éléments retournés par la requête cible q, les racines des réponses de q sont suffisantes pour effectuer la suppression. De ce fait, un id-projecteur correct pour la pré-évaluation de q est aussi correct pour la pré-évaluation de u.

La Figure 4.16 illustre une projection "correcte" pour la mise à jour u. Cette projection est faite avec K en supposant qu'il ne contient pas les identifiants des descendants de I_q. De ce fait, cet id-projecteur n'est pas correct pour l'évaluation globale de q.

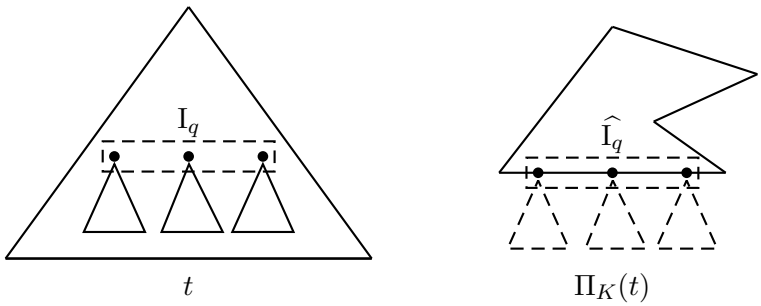


FIGURE 4.16 – Correction globale et correction pour la pré-évaluation des requêtes

■

Le résultat énoncé ci-dessous est important. Il énonce que l'inférence des chemins pour les requêtes est globalement correcte. Ce résultat utilise la notion de *string-free* path-projecteur pour un chemin P et une DTD D. Cette notion se distingue de la notion de path-projecteur introduite dans [BCCN06] et reprise dans la Section 3.4 uniquement par le fait que l'étiquette $String$ est exclue des *string-free* path-projecteurs. Formellement, étant donnée une DTD D et un chemin P, le string-free path-projecteur associé à P suivant D est la paire d'ensembles d'étiquettes $(T, N - \{String\})$ où T et N sont définis dans la Section 3.4 en page 69. L'utilisation des *string-free* path-projecteurs est motivée par le fait que notre technique vise à "optimiser" la projection des noeuds textes dont la projection s'effectue (principalement) en considérant dans le composant *one level below* du tri-projecteur l'étiquette du noeud parent du noeud texte devant être projeté (cf. Section 4.1). L'inclusion du type $String$ dans un path-projecteur entraîne la protection inutile de tous les noeuds textes d'un document lorsque les parents de ceux-ci sont projetés, une des limites de la technique de [BCCN06].

Dans la suite du manuscrit, et pour des raisons de simplicité, on utilisera le terme path-projecteurs pour désigner les *string-free* path-projecteurs.

Lemme 2 (Correction de l'inférence des chemins pour les requêtes)**.**
Soit une DTD D, un document $t \in D$ et une requête q. Alors :
L'id-set $\mathbf{eval\text{-}}\boldsymbol{K}^q$ défini ci-dessous est un id-projecteur correct pour la requête q relativement à t.

$$\mathbf{eval\text{-}}\boldsymbol{K}^q = \bigcup_{\alpha \in \{\mathbf{nu},\mathbf{su},\mathbf{ebu},\mathbf{nr},\mathbf{sr}\}} \mathbf{eval\text{-}}\boldsymbol{K}^q_\alpha$$

où, on pose :

- pour $\alpha \in \{\mathbf{nu}, \mathbf{ebu}, \mathbf{nr}\}$, $QExt_\alpha(/, q) = \{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$
- pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$, $\{P^1_\alpha, .., P^{k_\alpha}_\alpha\} = \{P/\texttt{parent::node}() \mid P \in QExt_\alpha(/, q)\}$

et pour $\alpha \in \{\mathbf{nu}, \mathbf{ebu}, \mathbf{nr}, \mathbf{su}, \mathbf{sr}\}$ et $i = 1..k_\alpha$, on définit :

- $\mathbf{pi}^i_\alpha$ le path-projecteur pour P^i_α et D
- $ev\text{-}T^i_\alpha = PEval(t, P^i_\alpha)$
- $ev\text{-}N^i_\alpha = KPath(t, \mathbf{pi}^i_\alpha)$
- $\mathbf{eval}\text{-}\boldsymbol{K}^q_{\mathbf{nu}} = \bigcup_{i=1}^{k_{\mathbf{nu}}} [ev\text{-}N^i_{\mathbf{nu}}]$
- pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$ $\mathbf{eval}\text{-}\boldsymbol{K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [ev\text{-}N^i_\alpha \cup \mathit{Child}(t, ev\text{-}T^i_\alpha)]$
- pour $\alpha \in \{\mathbf{nr}, \mathbf{ebu}\}$, $\mathbf{eval}\text{-}\boldsymbol{K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [ev\text{-}N^i_\alpha \cup \mathit{Desc}(t, ev\text{-}T^i_\alpha)]$

■

Le Lemme 2 énonce la correction de l'inférence des chemins pour les requêtes d'une mise à jour en construisant, à partir des chemins extraits de q, un id-set $\mathbf{eval}\text{-}\boldsymbol{K}^q$ correct pour l'évaluation de q relativement à t. Cette construction consiste essentiellement à identifier les noeuds de t traversés ou retournés par chaque chemin P^i_α et à considérer en plus de ces noeuds, soit les fils des noeuds retournés par ce chemin P^i_α soit leurs descendants en fonction de la catégorie α de P^i_α.

La Table 4.1 illustre pour chaque catégorie de chemins, les noeuds devant être considérés pour la projection. Pour chaque chemin P_q inféré à partir d'une requête, une cible de P_q, notée **i** pour les noeuds éléments et **st** pour les textes, est indiquée ainsi que les noeuds devant être projetés en fonction de la catégorie du chemin. Un trait discontinu entourant un noeud **i** indique que ce noeud doit être projeté seul. Un rectangle autour du noeud **i** indique que ce noeud doit être projeté avec tous ses frères. Enfin, un triangle en dessous d'un noeud **i** indique que ce noeud doit être avec tous ses descendants. On voit que pour un chemin *node used*, il est suffisant de projeter sa cible **i**. Un

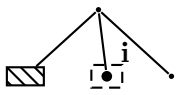
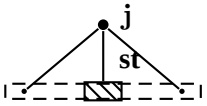
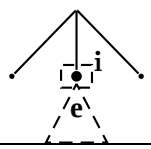
catégorie de P_q	*node used* ($\mathbf{nu}$)	*string used* ($\mathbf{su}$) *string returned* ($\mathbf{sr}$)	$\forall$ *below used* ($\mathbf{ebu}$) *node returned* ($\mathbf{nr}$)
Cible de P_q			

TABLE 4.1 – Extraction des chemins pour les requêtes : extrémités des paths

chemin *string used* ou *string returned* est d'abord étendu avec l'étape `parent::node()` afin de cibler le noeud **j** parent du texte **st**. Il faut ensuite projeter tous les fils du noeud **j** c.a.d. tous les noeuds frères de **st** (cf. Section 4.1 pour comprendre la projection des textes). Pour un chemin $\forall$ *below used* ou *node returned* qui cible les racines **i** d'éléments **e**, il est nécessaire de projeter les noeuds cibles **i** ainsi que tous leurs descendants.

Nous commentons maintenant la construction de chaque $\mathbf{eval}\text{-}\boldsymbol{K}^q_\alpha$ à partir des chemins P^i_α. Les noeuds retournés par les chemins sont obtenus en évaluant ces chemins par la fonction $PEval(t, P^i_\alpha)$. Les noeuds traversés par ces chemins sont obtenus en utilisant les path-projecteurs introduits dans la Section 3.4 (page 69). Pour rappel, un path-projecteur pour un chemin P est donné

par une paire d'ensembles d'étiquettes (T, N) issus respectivement de l'inférence de type et du projecteur pour les chemins. La projection basée sur les path-projecteurs est assurée par $KPath(t, N)$ qui retourne les noeuds de t dont les étiquettes appartiennent à N (et dont les parents appartiennent aussi à $KPath(t, N)$). Une fois les noeuds traversés ou retournés par les chemins P^i_α identifiés, il ne reste plus qu'à rajouter, en fonction de la catégorie α, soit les fils (voir l'utilisation de *Child*(t, KT^i_α) dans la construction de $\mathbf{eval}$-$\boldsymbol{K}^q_\alpha$) soit les descendants (voir l'utilisation de *Desc*(t, KT^i_α) dans la construction de $\mathbf{eval}$-$\boldsymbol{K}^q_\alpha$ pour $\alpha \in \{\mathbf{ebu}, \mathbf{nr}\}$) des noeuds retournés par ces chemins.

La démonstration du Lemme 2 n'est pas faite dans ce document. Elle s'inspire de la preuve de la technique de projection pour les requêtes de [BCCN06] qui est faite dans [BCCN11].

La propriété suivante, abusivement appelée *Monotonie de la correction*, est utile pour les preuves de la correction de l'inférence des chemins pour les mises à jour.

Propriété 7 (Monotonie de la correction des id-projecteurs pour les requêtes)**.**
Soit une DTD D, un document $t{\in}D$ et une requête q. Soit $\mathbf{eval}$-$\boldsymbol{K}^q$ l'id-projecteur spécifié dans le Lemme 2. Alors :

Pour tout id-set $\boldsymbol{L}$ tel que $\mathbf{eval}$-$\boldsymbol{K}^q{\subseteq}\boldsymbol{L}$, $\boldsymbol{L}$ est correct pour q relativement à t.

Notons que la monotonie (au sens habituelle) de la correction pour les requêtes n'est pas vérifiée dans le cas général. Cela découle du fait que la correction des id-projecteurs pour les chemins n'est pas monotone comme l'illustre l'exemple suivant.

Exemple 50. Considérons la DTD D_5 donnée dans la Figure 4.12 en page 86. Considérons le document t valide pour D_5 donné dans la Figure 4.17.
Soit le chemin :
$P = \text{self :: doc/a[not(c/f) } \wedge \text{ d]}$.
Considérons les id-sets $K_0 = \{\epsilon, 1, 1.1, 1.2\}$ et $K_1 = K_0 \cup \{1.3\}$.
La projection de t suivant ces deux id-sets est donnée dans la Figure 4.17.

D'après la Définition 19, K_0 est correct pour l'évaluation de P relativement à t puisque $PEval(t, P){=}PEval(\Pi_{K_0}(t), P){=}\emptyset$. Cependant, bien que $K_0{\subseteq}K_1$, K_1 n'est pas un id-projecteur correct pour P relativement à t puisque $PEval(\Pi_{K_1}(t), P){=}\{1\}$.

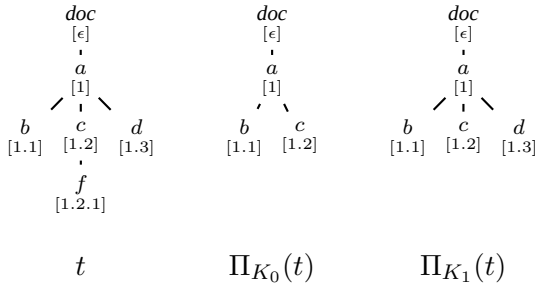


FIGURE 4.17 – Contre-exemple : monotonie.

■

La monotonie de la correction pour les requêtes est néanmoins vérifiée pour les id-projecteurs construits tel que spécifié dans le Lemme 2 puisque ces id-projecteurs sont basés sur les path-

projecteurs. Les path-projecteurs sont obtenus par l'inférence du projecteur pour les chemins. L'utilisation des path-projecteurs permet de construire des id-projecteurs "minimaux" corrects pour les chemins tel que le montre l'exemple suivant.

Exemple 51. Soit le chemin P et la DTD D_5 de l'exemple précédent. Le path-projecteur (T, N) dérivé pour le P et D_5 est donné par :
$(\{a\}, \{doc, d, c, f\})$.
L'id-projecteur $\mathbf{eval}$-$\boldsymbol{K}^q$ dérivé pour P relativement à t suivant le Lemme 2 est :
$\{\epsilon, 1, 1.2, 1.2.1, 1.3\}$.
L'utilisation du path-projecteur (T, N) a permis de récupérer les identifiants de tout noeud traversé ou retourné par P sur t. L'id-projecteur ainsi obtenu représente donc l'id-projecteur "minimal" correct pour P relativement à t. ■

La propriété 6 permet, à partir du Lemme 2 de déduire le résultat suivant :

Corollaire 1.
Sous les mêmes hypothèses que Lemme 2 $\mathbf{eval}$-$\boldsymbol{K}^q$ est correct pour la matérialisation des réponses de q relativement à t.

De la même manière, pour une requête q et un document t telle que $\mathrm{I}_q \subseteq \mathit{dom}(\sigma_t)$, nous pouvons utiliser la propriété 6 pour déduire que $\mathbf{eval}$-$\boldsymbol{K}^q$ est correct pour la pré-évaluation de q relativement à t. Toutefois, l'id-set $\mathbf{eval}$-$\boldsymbol{K}^q$ construit pour la correction globale est potentiellement "trop grand" pour la correction pour la pré-évaluation. Le lemme 3 propose donc une construction plus fine d'un id-set correct pour la pré-évaluation à partir bien sûr des chemins extraits pour la requête q.

Lemme 3 (Correction de l'inférence des chemins pour la pré-évaluation des requêtes)**.**
Soit une DTD D, un document $t \in D$ et une requête q telle que $\mathrm{I}_q \subseteq \mathit{dom}(\sigma_t)$. Alors :
L"id-set $\mathbf{pre}$-$\boldsymbol{K}^q$ défini ci-dessous est un id-projecteur correct pour la pré-évaluation de la requête q relativement à t.

$$\mathbf{pre}\text{-}\boldsymbol{K}^q = \bigcup_{\alpha \in \{\mathbf{nu},\mathbf{su},\mathbf{ebu},\mathbf{nr},\mathbf{sr}\}} \mathbf{pre}\text{-}\boldsymbol{K}^q_\alpha$$

où, on pose :

- pour $\alpha \in \{\mathbf{nu}, \mathbf{ebu}, \mathbf{nr}\}$, $QExt_\alpha(/, q) = \{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$
- pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$, $\{P^1_\alpha, .., P^{k_\alpha}_\alpha\} = \{P/\texttt{parent::node()} \mid P \in QExt_\alpha(/, q)\}$

et pour $\alpha \in \{\mathbf{nu}, \mathbf{su}, \mathbf{ebu}, \mathbf{nr}, \mathbf{sr}\}$ et $i = 1..k_\alpha$ on définit

- $\mathbf{pi}^i_\alpha$ le path-projecteur pour P^i_α et D
- $\mathit{pre}\text{-}T^i_\alpha = PEval(t, P^i_\alpha)$
- $\mathit{pre}\text{-}N^i_\alpha = KPath(t, \mathbf{pi}^i_\alpha)$
- pour $\alpha \in \{\mathbf{nu}, \mathbf{nr}\}$ $\mathbf{pre}\text{-}\boldsymbol{K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [\mathit{pre}\text{-}N^i_\alpha]$
- pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$ $\mathbf{pre}\text{-}\boldsymbol{K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [\mathit{pre}\text{-}N^i_\alpha \cup \mathit{Child}(t, \mathit{pre}\text{-}T^i_\alpha)]$
- $\mathbf{pre}\text{-}\boldsymbol{K}^q_{\mathbf{ebu}} = \bigcup_{i=1}^{k_{\mathbf{ebu}}} [\mathit{pre}\text{-}N^i_{\mathbf{ebu}} \cup \mathit{Desc}(t, \mathit{pre}\text{-}T^i_{\mathbf{ebu}})]$

■

L'id-projecteur $\mathbf{pre}$-$\boldsymbol{K}^q$ défini ci-dessus est construit de manière analogue à celui du lemme 2 sauf pour les chemins de la catégorie *node returned*. Il n'est pas nécessaire, pour ces chemins de considérer les descendants des noeuds qu'ils ciblent parce que la correction pour la pré-évaluation s'intéresse exclusivement aux racines des réponses.

De la même manière que pour la correction au sens évaluation globale, nous énonçons la propriété de la monotonie pour la correction au sens pré-évaluation.

Propriété 8 (Monotonie de la correction de la pré-évaluation des réponses aux requêtes)**.**
Soit t un document, q une requête telle que $\mathrm{I}_q \subseteq dom(\sigma_t)$. Soit $\mathbf{pre}$-$\boldsymbol{K}^q$ l'id-projecteur défini dans le lemme 3. Alors pour tout id-set $\boldsymbol{L}$ tel que $\mathbf{pre}$-$\boldsymbol{K}^q \subseteq \boldsymbol{L}$, on a :

$\boldsymbol{L}$ est correct pour la pré-évaluation de q relativement à t.

4.3.1.3 Analyse statique des mises à jour

Nous abordons maintenant l'extraction des chemins pour les mises à jour. Cette extraction considère les catégories *node only*, *one level below* et $\forall$ *below* introduites dans la Section 4.1. Elle s'appuie sur l'analyse des chemins extraits à partir des requêtes d'une mise à jour et sur l'emplacement ou le rôle (contexte, source ou cible) de ces requêtes au sein des mises à jour.

Nous introduisons le mécanisme d'extraction de manière informelle d'abord en nous appuyant sur la Table 4.2 qui synthètise les différents cas à considérer.

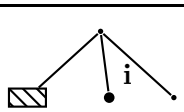
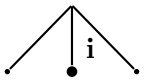
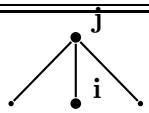
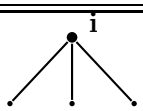
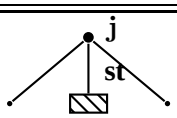
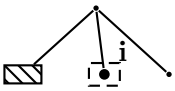
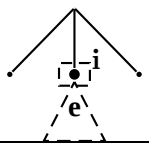
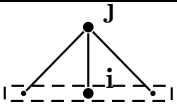
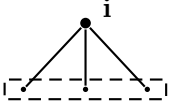
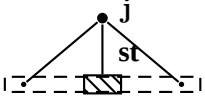
	Cas 1	**Cas 2**	**Cas 3**	**Cas 4**	**Cas 5**
P_{q_u}					
P_u					

TABLE 4.2 – Extraction des chemins pour les mises à jour : analyse.

Etant une requête q_u imbriqué dans la mise à jour u en cours d'analyse, nous notons P_{q_u} tout chemin extrait de q_u. Les différents cas de figure pour P_{q_u} sont représentés dans la première ligne de la Table 4.2. Un chemin P_{q_u} à partir de son analyse peut se trouver transformé en un chemin noté P_u qui capture les noeuds à projeter pour la mise à jour u. Les chemins transformés P_u sont représentés dans la seconde ligne de la Table 4.2 qui utilise les mêmes conventions que la table 4.1.

Afin d'illustrer chacun des cas de la Table 4.2, nous allons nous appuyer sur l'exemple de la Figure 4.18.

Cas 1 Ce cas correspond à la situation où P_{q_u} est extrait en tant que :

let $\$doc$= self : : doc/a ($q_{\texttt{ctxt}}$) **return**

insert **for** $\$x$ in $\$doc$ **where** $\$x$/b/text()="..." **return** ($\$x$/d, $\$x$/c/text()) ($q_{\texttt{src}}$) **as-last into** **let** $\$y$:=self : : doc/e **where** $\$y$/c = <c>...</c> **return** $\$y$ ($q_{\texttt{cib}}$)

(a) La mise à jour u

	$q_{\texttt{ctxt}}$	$q_{\texttt{src}}$	$q_{\texttt{cib}}$
Node used	-	self : : doc/a	self : :doc/e
String used	-	self : : doc/a/b/text()	-
$\forall$ below used	self : :doc/a	-	self : :doc/e/c
Node returned	-	self : : doc/a/d	self : :doc/e
String returned	-	self : :doc/a/c/text()	-

(b) L'extraction des paths à partir des requêtes $q_{\texttt{ctxt}}$, $q_{\texttt{src}}$ et $q_{\texttt{cib}}$

node only	self : :doc/a self : :doc/e
one level below	self : :doc/a/b/text()/parent::node() self : :doc/a/c/text()/parent::node() self : :doc/e/
$\forall$ below	self : :doc/a/d self : :doc/e/c

(c) L'extraction des paths pour u

FIGURE 4.18 – Extraction des chemins pour les mises à jour : exemple

i) chemin *node used* à partir de la requête contextuelle ou de la requête cible ou de la requête source de la mise à jour analysée u, ou en tant que

ii) chemin *node returned* à partir de la requête contextuelle de u, ou en tant que

iii) chemin *node returned* à partir de la requête cible $q_{\texttt{cib}}$ des mises à jour **delete** $q_{\texttt{cib}}$ et **rename** $q_{\texttt{cib}}$ as a.

Dans chacun de ces cas, il est suffisant de projeter le noeud **i** seul puisqu'il est simplement utilisé par la mise à jour et puisque la suppression et le renommage d'un noeud **i** ne nécessitent pas de projeter l'élément enraciné en **i**. En terme d'extraction des chemins pour la mise à jour u analysée, le chemin extrait est P_u=P_{q_u} dans la catégorie *node only*.

Exemple 52. Nous nous référons ici à l'exemple de la Figure 4.18. Les chemins $\text{self} :: \text{doc/a}$ $\text{self} :: \text{doc/e}$ retournés comme chemins 'node only' sont un exemple pour le cas 1. ■

Cas 2 Ce cas correspond à la situation où P_{q_u} est extrait en tant que :

i) chemin $\forall$ *below used* à partir de n'importe quelle requête (contextuelle, cible ou source), ou en tant que

ii) chemin *node returned* à partir de la requête source $q_{\texttt{src}}$ d'une mise à jour atomique de la forme $u_1 = \textbf{insert}\ q_{\texttt{src}}\ \delta\ q_{\texttt{cib}}$ ou de la forme $u_2 = \textbf{replace}\ q_{\texttt{cib}}\ \texttt{with}\ q_{\texttt{src}}$.

Dans ce cas, il est nécessaire de projeter l'élément **e** enraciné en **i**. Le sous-cas i) suggère que cet élément est utilisé pour vérifier une condition de la mise à jour analysée. Le sous-cas ii) suggère que cet élément devra être matérialisé (copié) par la mise à jour u_1 ou u_2. En terme d'extraction des chemins pour la mise à jour u analysée, le chemin extrait est $P_u{=}P_{q_u}$ dans la catégorie $\forall$ *below*.

Exemple 53. Nous nous référons une fois encore à l'exemple de la Figure 4.18. Les chemins $\text{self} :: \text{doc/a/d}$ et $\text{self} :: \text{doc/e/c}$ sont retournés comme des chemins $\forall$ *below* pour u. Le premier parce que son évaluation est matérialsié (copié) par u et le second parce que son évaluation est accédée par $q_{\texttt{cib}}$ pour vérifier la condition $\$y$/c=<c>..</c>. ■

Cas 3 Ce cas correspond à la situation où P_{q_u} est extrait en tant que :

i) chemin *node returned* de la requête cible $q_{\texttt{cib}}$ d'une mise à jour de la forme $\textbf{replace}\ q_{\texttt{cib}}\ \texttt{with}\ q_{\texttt{src}}$ ou de la forme $\textbf{insert}\ q_{\texttt{src}}\ \delta\ q_{\texttt{cib}}$ avec $\delta \in \{\leftarrow, \rightarrow\}$.

Dans ce cas, entre autre pour assurer le bon fonctionnement de l'étape de fusion, il est nécessaire de préparer la projection de tous les noeuds frère de **i**, noeud ciblé par P_{q_u}. Pour ce faire, le chemin P_{q_u} est étendu avec l'étape $\texttt{parent::node}()$ afin de remonter au noeud **j** dont on souhaite projeter tous les fils. En terme d'extraction des chemins pour la mise à jour u analysée, le chemin extrait est $P_u{=}P_{q_u}/\texttt{parent::node}()$ dans la catégorie *one level below*.

Exemple 54. L'exemple de la Figure 4.18. ne permet pas d'illustrer le cas décrit ci-dessus. Nous allons donc considérer la mise à jour $u{=}\textbf{let}\ \$doc\ \texttt{:=}\ q_{\texttt{ctxt}}\ \textbf{return}\ u'$ avec $u'{=}\textbf{replace}\ q_{\texttt{src}}\ \texttt{with}\ q_{\texttt{cib}}$. Le chemin $\text{self} :: \text{doc/e}$ extrait comme *node returned* de $q_{\texttt{cib}}$ sera étendu par l'étape $\texttt{parent::node}()$ et retourné comme chemin *one level below* pour u. ■

Cas 4 Ce cas correspond à la situation où P_{q_u} est extrait en tant que :

i) chemin *node returned* de la requête cible $q_{\texttt{cib}}$ d'une mise à jour $\textbf{insert}\ q_{\texttt{src}}\ \delta\ q_{\texttt{cib}}$ avec $\delta \in \{\swarrow, \downarrow, \searrow\}$.

Dans ce cas, il faut préparer la projection de tous les fils du noeud **i** ciblé par P_{q_u}. En terme d'extraction des chemins pour la mise à jour u analysée, le chemin extrait est $P_u{=}P_{q_u}$ dans la catégorie *one level below*.

Exemple 55. Nous nous référons à nouveau à l'exemple de la Figure 4.18. Le chemin self : :doc/e extrait comme *node returned* de $q_{\texttt{cib}}$ est extrait comme chemin *one level below* pour u. ■

Cas 5 Ce cas correspond à la situation où P_{q_u} est extrait en tant que :

i) chemin *string used* ou *string returned* à partir de n'importe quelle requête (contextuelle, source, cible).

Dans ce cas (qui regroupe les situations où un noeud texte **st** est cible d'un *delete*, d'un *insert* ou d'un *replace* ou simplement accédé par une condition exprimée dans requête cible, source ou contextuelle), afin de préparer la projection de tous les frères du noeud du noeud texte **st** ciblé par P_{q_u}, le chemin P_q est étendu par l'étape `parent::node()`. En terme d'extraction des chemins pour la mise à jour u analysée, le chemin extrait est $P_u{=}P_{q_u}/\texttt{parent::node()}$ dans la catégorie *one level below*.

Exemple 56. Pour l'exemple de la Figure 4.18, le chemin self : :doc/a/b/text(), respectivement self : :doc/a/c/text() extrait comme *string used* , respectivement *string returned*, de $q_{\texttt{src}}$ est étendu avec l'étape `parent::node()` et retourné comme chemin *one level below* de u. ■

Nous formalisons l'extraction des chemins à partir des mises à jour à l'aide de règles utilisant les jugements suivants :

$$(\Gamma, u) \rightsquigarrow_{cat} \boldsymbol{P} \quad (\mathbf{UpPathExt})$$

où cat est l'une des catégories *node only* abrégée **no**, *one level below* abrégée **olb** ou $\forall$ *below* abrégée **eb**.

La table ci-dessous définit la notation des chemins extraits pour chaque catégorie.

catégorie	**jugement**
node only	$(\Gamma, u) \rightsquigarrow_{\mathbf{no}} \boldsymbol{P}_{\mathbf{no}}$
one level below	$(\Gamma, u) \rightsquigarrow_{\mathbf{olb}} \boldsymbol{P}_{\mathbf{olb}}$
$\forall$ *below*	$(\Gamma, u) \rightsquigarrow_{\mathbf{eb}} \boldsymbol{P}_{\mathbf{eb}}$

Comme d'habitude, nous introduisons la fonction $UExt(\Gamma, u)$ comme notation alternative à c.

$$\boxed{UExt_{cat}(\Gamma, u) = \boldsymbol{P} \quad ssi \quad (\Gamma, u) \rightsquigarrow_{cat} \boldsymbol{P}}$$

Les conventions utilisées dans la présentation des règles d'extraction sont :

- `ins-rep`$(q_{\texttt{src}}, q_{\texttt{cib}})$ désigne une des mises à jour **insert** $q_{\texttt{src}}$ δ $q_{\texttt{cib}}$ avec $\delta \in \{\leftarrow, \rightarrow\}$ ou **replace** $q_{\texttt{cib}}$ `with` $q_{\texttt{src}}$,
- Etant donné un ensemble de chemins $\boldsymbol{P}$, $Par(\boldsymbol{P})$ désigne l'ensemble des chemins $\{P/\texttt{parent::node()} \mid P \in \boldsymbol{P}\}$.
- $\delta_{\texttt{sib}}$ désigne une des directions $\leftarrow, \rightarrow$ et $\delta_{\texttt{in}}$ une des directions $\swarrow, \downarrow, \searrow$.

Comme pour les requêtes, la présentation des règles d'extraction pour les mises à jour est guidée par la syntaxe de celles-ci. Nous commençons par donner les règles d'extraction pour les mises à jour atomiques.

L'extraction des chemins pour les insertions (**insert** $q_{\texttt{src}}$ δ $q_{\texttt{cib}}$) est analogue à l'extraction des chemins pour les remplacements (**replace** $q_{\texttt{cib}}$ `with` $q_{\texttt{src}}$) d'où leur présentation dans le même encadré (Figure 4.19). Pour comprendre ces règles, nous rappelons que :

R1 les mises à jour **replace** $q_{\texttt{cib}}$ `with` $q_{\texttt{src}}$ et **insert** $q_{\texttt{src}}$ $\delta_{\texttt{sib}}$ $q_{\texttt{cib}}$ insèrent une sous-forêt au même niveau que le noeud retourné par la requête cible $q_{\texttt{cib}}$ et requièrent d'étendre les chemins extraits de la requête cible $q_{\texttt{cib}}$ avec l'étape `parent::node()`.

R2 la mise à jour **insert** $q_{\texttt{src}}$ $\delta_{\texttt{in}}$ $q_{\texttt{cib}}$ se traite différemment puisque l'insertion se fait sous le noeud retourné par $q_{\texttt{cib}}$. Dans ce cas, les chemins *one level below* sont directement obtenus à partir des chemins *node returned* de $q_{\texttt{cib}}$.

$$(\textbf{UPE:ins:no}) \quad \frac{(\Gamma, q_{\texttt{src}}) \rightsquigarrow_{\vec{\texttt{nu}}} \boldsymbol{N}^{\texttt{U}}_{\texttt{src}} \quad (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{nu}}} \boldsymbol{N}^{\texttt{U}}_{\texttt{cib}}}{(\Gamma, \texttt{ins-rep}(q_{\texttt{src}}, q_{\texttt{cib}})) \rightsquigarrow_{\vec{\texttt{no}}} \boldsymbol{N}^{\texttt{U}}_{\texttt{src}} \cup \boldsymbol{N}^{\texttt{U}}_{\texttt{cib}}}$$

$$(\textbf{UPE:ins:olb}) \quad \frac{\begin{array}{c}(\Gamma, q_{\texttt{src}}) \rightsquigarrow_{\vec{\texttt{su}}} \boldsymbol{S}^{\texttt{U}}_{\texttt{src}} \quad (\Gamma, q_{\texttt{src}}) \rightsquigarrow_{\vec{\texttt{sr}}} \boldsymbol{S}^{\texttt{R}}_{\texttt{src}} \quad (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{nr}}} \boldsymbol{N}^{\texttt{R}}_{\texttt{cib}} \\ (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{su}}} \boldsymbol{S}^{\texttt{U}}_{\texttt{cib}} \quad (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{sr}}} \boldsymbol{S}^{\texttt{R}}_{\texttt{cib}}\end{array}}{(\Gamma, \texttt{ins-rep}(q_{\texttt{src}}, q_{\texttt{cib}})) \rightsquigarrow_{\vec{\texttt{olb}}} Par(\boldsymbol{S}^{\texttt{U}}_{\texttt{src}} \cup \boldsymbol{S}^{\texttt{R}}_{\texttt{src}} \cup \boldsymbol{N}^{\texttt{R}}_{\texttt{cib}} \cup \boldsymbol{S}^{\texttt{U}}_{\texttt{cib}} \cup \boldsymbol{S}^{\texttt{R}}_{\texttt{cib}})}$$

$$(\textbf{UPE:ins:eb}) \quad \frac{(\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{ebu}}} \boldsymbol{E}_{\texttt{cib}} \quad (\Gamma, q_{\texttt{src}}) \rightsquigarrow_{\vec{\texttt{ebu}}} \boldsymbol{E}_{\texttt{src}} \quad (\Gamma, q_{\texttt{src}}) \rightsquigarrow_{\vec{\texttt{nr}}} \boldsymbol{N}^{\texttt{R}}_{\texttt{src}}}{(\Gamma, \texttt{ins-rep}(q_{\texttt{src}}, q_{\texttt{cib}})) \rightsquigarrow_{\vec{\texttt{eb}}} \boldsymbol{E}_{\texttt{cib}} \cup \boldsymbol{E}_{\texttt{src}} \cup \boldsymbol{N}^{\texttt{R}}_{\texttt{src}}}$$

$$(\textbf{UPE:ins:olb-bis}) \quad \frac{\begin{array}{c}(\Gamma, q_{\texttt{src}}) \rightsquigarrow_{\vec{\texttt{su}}} \boldsymbol{S}^{\texttt{U}}_{\texttt{src}} \quad (\Gamma, q_{\texttt{src}}) \rightsquigarrow_{\vec{\texttt{sr}}} \boldsymbol{S}^{\texttt{R}}_{\texttt{src}} \quad (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{nr}}} \boldsymbol{N}^{\texttt{R}}_{\texttt{cib}} \\ (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{su}}} \boldsymbol{S}^{\texttt{U}}_{\texttt{cib}} \quad (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{sr}}} \boldsymbol{S}^{\texttt{R}}_{\texttt{cib}}\end{array}}{(\Gamma, \textbf{insert}\ q_{\texttt{src}}\ \delta_{\texttt{in}}\ q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{olb}}} Par(\boldsymbol{S}^{\texttt{U}}_{\texttt{src}} \cup \boldsymbol{S}^{\texttt{R}}_{\texttt{src}} \cup \boldsymbol{S}^{\texttt{U}}_{\texttt{cib}} \cup \boldsymbol{S}^{\texttt{R}}_{\texttt{cib}}) \cup \boldsymbol{N}^{\texttt{R}}_{\texttt{cib}}}$$

FIGURE 4.19 – Extraction des chemins : `insert` et `replace`

L'extraction des chemins pour **delete** $q_{\texttt{cib}}$ est identique à celle pour **rename** $q_{\texttt{cib}}$ `as` a. Les règles pour **delete** $q_{\texttt{cib}}$ sont données dans la Figure 4.20.

$$(\textbf{UPE:del:no}) \quad \frac{(\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{nu}}} \boldsymbol{N}^{\texttt{U}}_{\texttt{cib}} \quad (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{nr}}} \boldsymbol{N}^{\texttt{R}}_{\texttt{cib}}}{(\Gamma, \textbf{delete}\ q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{no}}} \boldsymbol{N}^{\texttt{U}}_{\texttt{cib}} \cup \boldsymbol{N}^{\texttt{R}}_{\texttt{cib}}}$$

$$(\textbf{UPE:del:olb}) \quad \frac{(\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{su}}} \boldsymbol{S}^{\texttt{U}}_{\texttt{cib}} \quad (\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{sr}}} \boldsymbol{S}^{\texttt{R}}_{\texttt{cib}}}{(\Gamma, \textbf{delete}\ q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{olb}}} Par(\boldsymbol{S}^{\texttt{U}}_{\texttt{cib}} \cup \boldsymbol{S}^{\texttt{R}}_{\texttt{cib}})}$$

$$(\textbf{UPE:del:eb}) \quad \frac{(\Gamma, q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{ebu}}} \boldsymbol{E}_{\texttt{cib}}}{(\Gamma, \textbf{delete}\ q_{\texttt{cib}}) \rightsquigarrow_{\vec{\texttt{eb}}} \boldsymbol{E}_{\texttt{cib}}}$$

FIGURE 4.20 – Extraction des chemins : `delete`

L'extraction des chemins pour les mises à jour composées est formalisée dans les Figures 4.21 et 4.22. Elle utilise évidemment l'extraction des chemins à partir de mises à jour atomiques que nous venons de présenter. Le principe suivi pour l'extraction des chemins à partir des requêtes contextuelles est exactement le même que pour le cas des requêtes composées (**for** x `in` $q_{\texttt{ctxt}}$ **return** $q_{\texttt{res}}$, etc). Nous ne commentons pas ces règles.

Pour les règles ci-dessous, on pose :
$\Gamma_0 = \texttt{Extend}(\Gamma, x, q_0)$

$$(\textbf{UPE:for:no}) \quad \frac{(\Gamma, q_0) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}_0^{\texttt{U}} \quad (\Gamma, q_0) \rightsquigarrow_{\texttt{nr}} \boldsymbol{N}_0^{\texttt{R}} \quad (\Gamma_0, u_1) \rightsquigarrow_{\texttt{no}} \boldsymbol{P}_{\textbf{no}}}{(\Gamma, \textbf{for } x \texttt{ in } q_0 \textbf{ return } u_1) \rightsquigarrow_{\texttt{no}} \boldsymbol{N}_0^{\texttt{U}} \cup \boldsymbol{N}_0^{\texttt{R}} \cup \boldsymbol{P}_{\textbf{no}}}$$

$$(\textbf{UPE:for:olb}) \quad \frac{(\Gamma, q_0) \rightsquigarrow_{\texttt{su}} \boldsymbol{S}_0^{\texttt{U}} \quad (\Gamma, q_0) \rightsquigarrow_{\texttt{sr}} \boldsymbol{S}_0^{\texttt{R}} \quad (\Gamma_0, u_1) \rightsquigarrow_{\texttt{olb}} \boldsymbol{P}_{\textbf{olb}}}{(\Gamma, \textbf{for } x \texttt{ in } q_0 \textbf{ return } u_1) \rightsquigarrow_{\texttt{olb}} Par(\boldsymbol{S}_0^{\texttt{U}} \cup \boldsymbol{S}_0^{\texttt{R}}) \cup \boldsymbol{P}_{\textbf{olb}}}$$

$$(\textbf{UPE:for:eb}) \quad \frac{(\Gamma, q_0) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_0 \quad (\Gamma_0, u_1) \rightsquigarrow_{\texttt{eb}} \boldsymbol{P}_{\textbf{eb}}}{(\Gamma, \textbf{for } x \texttt{ in } q_0 \textbf{ return } u_1) \rightsquigarrow_{\texttt{eb}} \boldsymbol{E}_0 \cup \boldsymbol{P}_{\textbf{eb}}}$$

FIGURE 4.21 – Règles d'extraction : mises à jour itératives

$$(\textbf{UPE:if:no}) \quad \frac{\begin{array}{c}(\Gamma, q_0) \rightsquigarrow_{\texttt{nu}} \boldsymbol{N}_0^{\texttt{U}} \quad (\Gamma, q_0) \rightsquigarrow_{\texttt{nr}} \boldsymbol{N}_0^{\texttt{R}} \\ (\Gamma, u_1) \rightsquigarrow_{\texttt{no}} \boldsymbol{P}_{\textbf{no}}^1 \quad (\Gamma, u_2) \rightsquigarrow_{\texttt{no}} \boldsymbol{P}_{\textbf{no}}^2\end{array}}{(\Gamma, \textbf{if } q_0 \textbf{ then } u_1 \textbf{ else } u_2) \rightsquigarrow_{\texttt{no}} \boldsymbol{N}_0^{\texttt{U}} \cup \boldsymbol{N}_0^{\texttt{R}} \cup \boldsymbol{P}_{\textbf{no}}^1 \cup \boldsymbol{P}_{\textbf{no}}^2}$$

$$(\textbf{UPE:if:olb}) \quad \frac{\begin{array}{c}(\Gamma, q_0) \rightsquigarrow_{\texttt{su}} \boldsymbol{S}_0^{\texttt{U}} \quad (\Gamma, q_0) \rightsquigarrow_{\texttt{sr}} \boldsymbol{S}_0^{\texttt{R}} \\ (\Gamma, u_1) \rightsquigarrow_{\texttt{olb}} \boldsymbol{P}_{\textbf{olb}}^1 \quad (\Gamma, u_2) \rightsquigarrow_{\texttt{olb}} \boldsymbol{P}_{\textbf{olb}}^2\end{array}}{(\Gamma, \textbf{if } q_0 \textbf{ then } u_1 \textbf{ else } u_2) \rightsquigarrow_{\texttt{olb}} Par(\boldsymbol{S}_0^{\texttt{U}} \cup \boldsymbol{S}_0^{\texttt{R}}) \cup \boldsymbol{P}_{\textbf{olb}}^1 \cup \boldsymbol{P}_{\textbf{olb}}^2}$$

$$(\textbf{UPE:if:eb}) \quad \frac{(\Gamma, q_0) \rightsquigarrow_{\texttt{ebu}} \boldsymbol{E}_0 \quad (\Gamma, u_1) \rightsquigarrow_{\texttt{eb}} \boldsymbol{P}_{\textbf{eb}}^1 \quad (\Gamma, u_2) \rightsquigarrow_{\texttt{eb}} \boldsymbol{P}_{\textbf{eb}}^2}{(\Gamma, \textbf{if } q_0 \textbf{ then } u_1 \textbf{ else } u_2) \rightsquigarrow_{\texttt{no}} \boldsymbol{E}_0 \cup \boldsymbol{P}_{\textbf{eb}}^1 \cup \boldsymbol{P}_{\textbf{eb}}^2}$$

FIGURE 4.22 – Règles d'extraction : mises à jour conditionnelles

4.3.1.4 Correction de l'inférence des chemins pour les mises à jour

Afin d'énoncer la correction de l'inférence des chemins pour les mises à jour il est nécessaire de définir la correction des id-projecteurs pour la pré-évaluation des mises à jour (à ne pas confondre avec la pré-évaluation des requêtes).

Rappelons que la pré-évaluation d'une mise à jour u sur un document $t{=}(r_t, \sigma_t)$ produit une PUL (σ_ω, ω) où ω est une liste de primitives de mise à jour et σ_ω un store décrivant les éléments devant être insérés par certaines des primitives de ω. Intuitivement, pour qu'un id-set K soit un id-projecteur correct pour la pré-évaluation de u sur t il faut que la PUL obtenue par pré-évaluation de u sur $\Pi_K(t)$ soit équivalente à la PUL obtenue par pré-évaluation de u sur t.

Pour définir l'équivalence des PUL, nous commençons par l'équivalence des primitives de mise à jour. Intuitivement, deux primitives de mise à jour μ et μ' sont équivalentes si leurs noeuds cibles sont identiques et si leurs éléments sources sont équivalents par les valeurs.

Définition 20 (Equivalence de primitives de mise à jour)**.**

Soient σ et σ' deux stores de domaines I et I′ non nécessairement disjoints. Soient (σ, μ) et (σ', μ') deux primitives de mise à jour (munies des stores décrivant les éléments à insérer).
L'équivalence $(\sigma, \mu) \sim (\sigma', \mu')$ est définie par :

- $(\sigma, \text{ins}(\text{J}, \delta, \mathbf{i})) \sim (\sigma', \text{ins}(\text{J}', \delta, \mathbf{j}))$ ssi $\mathbf{i}=\mathbf{j}$ et $(\sigma, \text{J}) \simeq (\sigma', \text{J}')$,
- $(\sigma, \text{del}(\mathbf{i})) \sim (\sigma', \text{del}(\mathbf{j}))$ ssi $\mathbf{i}=\mathbf{j}$,
- $(\sigma, \text{repl}(\mathbf{i}, \text{J})) \sim (\sigma', \text{repl}(\mathbf{j}, \text{J}'))$ ssi $\mathbf{i}=\mathbf{j}$ et $(\sigma, \text{J}) \simeq (\sigma', \text{J}')$, et
- $(\sigma, \text{ren}(a, \mathbf{i})) \sim (\sigma', \text{ren}(b, \mathbf{j}))$ ssi $a=b$ et $\mathbf{i}=\mathbf{j}$.

L'équivalence des PUL est définie comme suit.

Définition 21 (Equivalence des PUL)**.**
Soient σ and σ' deux stores de domaines I et I′ non nécessairement disjoints.
L'inclusion des deux PULs (σ, ω) et (σ', ω'), notée $(\sigma, \omega) \sqsubseteq (\sigma', \omega')$, est définie par induction sur ω comme suit :

- $(\sigma, ()) \sqsubseteq (\sigma', \omega')$,
- $(\sigma, \mu \cdot \omega) \sqsubseteq (\sigma', \omega')$ ssi $\omega' = \omega'_1 \cdot \mu' \cdot \omega'_2$, $(\sigma, \mu) \sim (\sigma', \mu')$, et $(\sigma, \omega) \sqsubseteq (\sigma', \omega'_2)$.

L'équivalence des deux PULs (σ, ω) et (σ', ω'), notée $(\sigma, \omega) \sim (\sigma', \omega')$ est définie par $(\sigma, \omega) \sqsubseteq (\sigma', \omega')$ et $(\sigma', \omega') \sqsubseteq (\sigma, \omega)$.

Remarquez que cette définition considère l'ordre des primitives dans les PULs comparées. A ce propos, nous supposons que les PULs sont toujours générées suivant le même ordre.

Nous donnons maintenant la définition de la correction des id-projecteurs pour la pré-évaluation des mises à jour.

Définition 22 (Id-projecteur correct pour la pré-évaluation des mises à jour)**.**
Soit u une mise à jour et $t=(r_t, \sigma_t)$ un document. Soit K un id-set. Posons :
$(\sigma_\omega, \omega) = \mathit{PUL}(t, u)$, et $(\widehat{\sigma_\omega}, \widehat{\omega}) = \mathit{PUL}(\Pi_K(t), u)$.
L'id-set K est un id-projecteur correct pour la pré-évaluation de u relativement à t ssi $(\sigma_t \cdot \sigma_\omega, \omega) \sim (\Pi_K(\sigma_t) \cdot \widehat{\sigma_\omega}, \widehat{\omega})$.

Le résultat suivant est important. Il énonce la correction de l'extraction des chemins à partir d'une mise à jour, pour la pré-évaluation, en exhibant un id-set construit à partir de ces chemins.

Lemme 4 (Correction de l'inférence des chemins pour la pré-évaluation des mises à jour)**.**
Soit une DTD D, un document $t \in D$ et une mise à jour u. Alors :
L'id-set $\boldsymbol{K}^u$ défini ci-dessous est un id-projecteur correct pour la pré-évaluation de u relativement à t.

$$\boldsymbol{K}^u = \bigcup_{\alpha \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}} \boldsymbol{K}^u_\alpha$$

où pour $\alpha \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$, on pose $UExt_\alpha((), u) = \{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$ et pour $i = 1..k_\alpha$ on a :

- $\mathbf{pi}^i_\alpha$ le path-projecteur pour P^i_α et D,

- KT^i_α=$PEval(t, P^i_\alpha)$,
- KN^i_α=$KPath(t, \mathbf{pi}^i_\alpha)$,
- $\boldsymbol{K}^u_{\mathbf{no}}$=$\bigcup_{i=1}^{k_{\mathbf{no}}}[KN^i_{\mathbf{no}}]$,
- $\boldsymbol{K}^u_{\mathbf{olb}}$=$\bigcup_{i=1}^{k_{\mathbf{olb}}}[KN^i_{\mathbf{olb}} \cup \mathit{Child}(t, KT^i_{\mathbf{olb}})]$,
- $\boldsymbol{K}^u_{\mathbf{eb}}$=$\bigcup_{i=1}^{k_{\mathbf{eb}}}[KN^i_{\mathbf{eb}} \cup \mathit{Desc}(t, KT^i_{\mathbf{eb}})]$.

■

Dans le lemme 4, la construction de $\boldsymbol{K}^u$ suit le même principe que la construction de $\mathbf{eval}$-$\boldsymbol{K}^q$ dans le lemme 2 : les noeuds traversés ou retournés par les chemins P^i_α sont d'abord identifiés ensuite, en fonction de leur catégorie α, les fils ou les descendants des noeuds retournés par les chemins P^i_α sont considérés. Remarquez que dans le lemme 4 ci-dessus, il n'y a pas besoin de rajouter d'étape `parent::node()` aux chemins extraits de u (comme c'est le cas pour les chemins *string used* et *string returned* dans le lemme 2) car l'extraction des chemins pour les mises à jour s'est chargée de cet ajustement.

La preuve de ce lemme est présentée dans l'annexe B.

La propriété suivante, abusivement appelée *Monotonie de la correction*, est utile pour les preuves de la correction de l'inférence du tri-projecteur pour u.

Propriété 9 (Monotonie de la correction des id-projecteurs pour les mises à jour)**.**
Soit une DTD D, un document $t \in D$ et une mise à jour u. Soit $\boldsymbol{K}^u$ l'id-projecteur spécifié dans le Lemme 4. Alors, pour tout id-set $\boldsymbol{L}$ tel que $\boldsymbol{K}^u \subseteq \boldsymbol{L}$, on a :

$\boldsymbol{L}$ est correct pour la pré-évaluation de u relativement à t.

4.3.2 Dérivation du tri-projecteur

Nous abordons la deuxième étape de l'inférence du tri-projecteur qui consiste à dériver les composants $\pi_{\mathbf{no}}$, $\pi_{\mathbf{olb}}$ et $\pi_{\mathbf{eb}}$ à partir des chemins $\boldsymbol{P}_{\mathbf{no}}$, $\boldsymbol{P}_{\mathbf{olb}}$ et $\boldsymbol{P}_{\mathbf{eb}}$ extraits des mises à jour lors de l'étape précédente. Cette étape se déroule en trois phases :

1. Inférence du path-projecteur (T_α, N_α) pour chaque chemin P_α extrait pour la mise à jour u dans la catégorie α, pour $\alpha \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$. Le résultat de cette étape est une paire d'ensemble d'étiquettes $(\tau_\alpha, \kappa_\alpha)$ telle que :

- τ_α est l'ensemble des étiquettes des noeuds ciblés par les chemins extraits dans la catégorie α,
- κ_α est l'ensemble des étiquettes des noeuds traversés par les chemins extraits dans la catégorie α.

2. Construction "provisoire" des composants $\pi_{\mathbf{no}}$, $\pi_{\mathbf{olb}}$ et $\pi_{\mathbf{eb}}$ du tri-projecteur. Cette construction se fait comme suit :

- Le composant $\pi_{\mathbf{no}}$ devant contenir les types des noeuds à projeter seuls, est peuplé par les étiquettes de $\tau_{\mathbf{no}}$, de $\kappa_{\mathbf{no}}$, de $\kappa_{\mathbf{olb}}$ et de $\kappa_{\mathbf{eb}}$,
- Les composants $\pi_{\mathbf{olb}}$ (resp. de $\pi_{\mathbf{eb}}$) est peuplé par les étiquettes de $\tau_{\mathbf{olb}}$ (resp. par les étiquettes de $\tau_{\mathbf{eb}}$).

3. Traitement des composants $\pi_{\mathbf{no}}$, $\pi_{\mathbf{olb}}$ et $\pi_{\mathbf{eb}}$ pour assurer leur disjonction. Le but de cette étape est d'assurer l'efficacité du tri-projecteur et de la fusion. Du point de vue de la projection, par exemple, lorsqu'une étiquette appartient au composant $\pi_{\mathbf{eb}}$ elle n'a pas besoin d'appartenir au composant $\pi_{\mathbf{olb}}$ ni au composant $\pi_{\mathbf{no}}$ puisque tous les descendants de ces noeuds de ce type (donc a fortiori leurs fils) seront projetés, d'après la Définition 18. De la même manière, lorsqu'un type appartient au composant $\pi_{\mathbf{olb}}$, il n'a pas besoin d'appartenir au composant $\pi_{\mathbf{no}}$.

Formellement le tri-projecteur est spécifié comme suit :

Définition 23 (Tri-projecteur associé à une mise à jour)**.**
Soit D une DTD, $t{\in}D$ un document valide pour D et u une mise à jour.
Pour $\alpha{\in}\{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$, soit $\boldsymbol{P}_\alpha{=}UExt_\alpha((), u)$ les chemins extraits pour la mise à jour u et la DTD D. Posons :

- $\boldsymbol{P}_\alpha{=}\{P_\alpha^1, .., P_\alpha^{k_\alpha}\}$.
- $\tau_\alpha{=}\bigcup_{i=1}^{k_\alpha} T_\alpha^i$ et $\kappa_\alpha{=}\bigcup_{i=1}^{k_\alpha} N_\alpha^i$ où $(T_\alpha^{i_\alpha}, N_\alpha^{i_\alpha})$ est le path-projecteur inféré pour le chemin $P_\alpha^{i_\alpha} \in \boldsymbol{P}_\alpha$.

Le tri-projecteur $\pi{=}(\pi_{\mathbf{no}}, \pi_{\mathbf{olb}}, \pi_{\mathbf{eb}})$ associé à la mise à jour u en présence de la DTD D est alors défini par :

$$\begin{array}{lcl} \pi_{\mathbf{eb}} & = & \tau_{\mathbf{eb}} \\ \pi_{\mathbf{olb}} & = & \tau_{\mathbf{olb}} - \pi_{\mathbf{eb}}, \text{ et} \\ \pi_{\mathbf{no}} & = & \tau_{\mathbf{no}} \cup \kappa_{\mathbf{no}} \cup \kappa_{\mathbf{olb}} \cup \kappa_{\mathbf{eb}} - (\pi_{\mathbf{eb}} \cup \pi_{\mathbf{olb}}). \end{array}$$

Exemple 57. Considérons la mise à jour de la Figure 4.18 et la DTD D_5 de la Figure 4.12 donnée en page 86. La Table 4.3 ci-dessous illustre la dérivation du tri-projecteur pour les chemins inférés de u en utilisant la DTD D_5. On remarque que l'étiquette e n'appartient pas à $\pi_{\mathbf{no}}$ car elle appartient à $\tau_{\mathbf{olb}}$. De même, c n'appartient pas à $\pi_{\mathbf{olb}}$ puisqu'elle appartient à $\tau_{\mathbf{eb}}$.

catégorie α	**paths de la catégorie** α	τ_α	κ_α	π_α
no	self :: doc/a	$\{a\}$	$\{doc, a\}$	$\{doc, a\}$
	self :: doc/e	$\{e\}$	$\{doc, e\}$	
olb	self :: doc/a/b/text()/`parent::node()`	$\{b\}$	$\{doc, a, b, \textit{String}\}$	$\{b, e\}$
	self :: doc/a/c/text()/`parent::node()`	$\{c\}$	$\{doc, a, e, \textit{String}\}$	
	self :: doc/e/	$\{e\}$	$\{doc, e\}$	
eb	self :: doc/a/d	$\{d\}$	$\{doc, a, d\}$	$\{c, d\}$
	self :: doc/e/c	$\{c\}$	$\{doc, e, c\}$	

TABLE 4.3 – Exemple d'extraction du tri-projecteur.

■

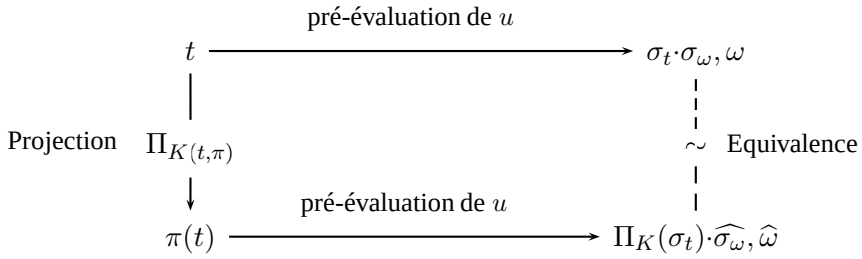


FIGURE 4.23 – Correction de l'inférence du tri-projecteur pour les mises à jour

Le résultat suivant est central : il énonce la correction de la dérivation du tri-projecteur pour les mises à jour.

Théorème 3 (Correction de l'inférence du tri-projecteur)**.**
Soit une DTD D, une mise à jour u. Soit le tri-projecteur π associé à la mise à jour u en présence de la DTD D telle que spécifié dans la Définition 23. Pour tout document $t \in D$, on a :

$K(t, \pi)$ est un id-projecteur correct pour la pré-évaluation de u relativement à t.

La preuve détaillée du Théorème 3 est présentée dans l'annexe B. Cette preuve s'appuie sur la correction de l'inférence des chemins pour le pré-évaluation des mises à jour énoncée par le lemme 4. Etant donné l'id-projecteur $\boldsymbol{K}^u$ défini dans ce lemme, il suffit de démontrer que $\boldsymbol{K}^u \subseteq K(t, \pi)$ et d'utiliser la propriété de monotonie 9 pour déduire que $K(t, \pi)$ est correct pour u relativement à t.

La Figure 4.23 illustre le résultat énoncé dans le Théorème 3.

4.4 Etape de fusion

Le contenu de cette section n'est pas une contribution personnelle. La formalisation de l'étape de fusion a été faite par Marina Sahakyan qui, dans sa thèse [Sah11], s'est focalisée par ailleurs sur l'implémentation et la validation expériementale de l'évaluation des mises à jour XQuery Update en utilisant la projection.

Cette section a pour but de formaliser la dernière étape du scénario de l'évaluation des mises à jour par projection. Cette étape consiste à construire le résultat final $u(t)$ à partir du document initial t et de la mise à jour partielle $u(\pi(t))$. Cette construction est assurée par la fonction *Merge* que nous avons présentée de manière informelle dans la Section 4.1. Nous rappelons que la fusion et donc *Merge* s'appuie sur les deux hypothèses suivantes :

1. Le document t correspond à un p-store dont les identifiants donnent la position des noeuds.

2. L'application de la mise à jour u peut introduire de nouveaux identifiants correspondant aux nouveaux noeuds introduits par l'insertion et le remplacement.

La fusion du document original t et de la mise à jour partielle $u(\pi(t))$ est assurée par deux fonctions *Merge* et *CMerge* qui sont formalisées dans la Figures 4.25. Le tri-projecteur π est un paramètre de ces fonctions. Pour allèger la notation, il restera implicite dans la présentation de ces fonctions.

Chacune de ces deux fonctions spécifie un parcours synchronisé de deux forêts : d'une part, la forêt F_i appartenant au document initial t et d'autre part, la forêt $\widehat{F_u}$ appartenant au résultat de la mise à jour partielle $u(\pi(t))$. Dans la suite, nous notons σ_t le store associé à t suivant la convention habituelle) et $\widehat{\sigma_u}$ le store associé à $u(\pi(t))$. La synchronisation de la fusion est capturée ici par l'hypothèse que les forêts F_i et $\widehat{F_u}$ paramètres des fonctions *Merge* et *CMerge* sont les éléments fils d'un seul noeud n de t, respectivement d'un seul noeud m de $u(\pi(t))$ telle que n et m partagent le même identifiant. En d'autre terme, le noeud m de $u(\pi(t))$ est issu de la projection du noeud n de t dont il a pu ou non conservé l'étiquette suivant que la mise à jour a effectué un renomage.

La fusion du document t avec le document $u(\pi(t))$ utilise les fonctions suivantes :

TreeMerge Cette fonction prend comme paramètre deux sous-arbres t_i et $\widehat{t_u}$ et construit un arbre *tree*$(r_{\widehat{t_u}}, F_r)$ dont la racine est la racine de $\widehat{t_u}$ et dont la sous-forêt F_r est définie à partir de *Merge* ou *CMerge* selon la catégorie de *lab*(r_{t_i}). Cette fonction est illustrée par la Figure 4.24.

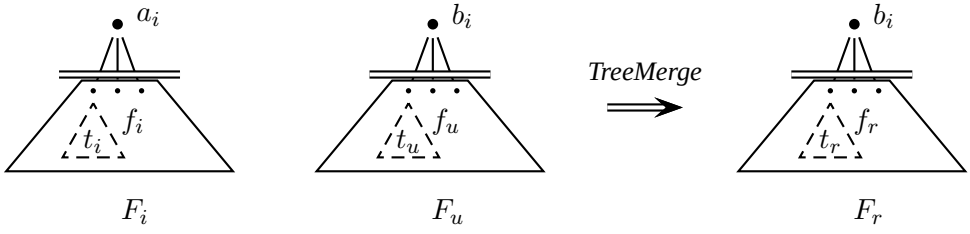


FIGURE 4.24 – Illustration de *TreeMerge*

Formellement,

TreeMerge$(t_i \mid \widehat{t_u}) =$ *tree*$(r_{\widehat{t_u}},$ *lab*$(r_{\widehat{t_u}}), F_r)$				où
$t.1$	F_r	$=$	*Merge*(*subfor*$(t_i) \mid$ *subfor*$(\widehat{t_u}))$	si $lab(r_{t_i}) \in \pi_{\mathbf{no}}$
$t.2$			*CMerge*(*subfor*$(t_i) \mid$ *subfor*$(\widehat{t_u}))$	si $lab(r_{t_i}) \in \pi_{\mathbf{olb}}$
$t.3$			*subfor*$(\widehat{t_u})$	si $lab(r_{t_i}) \in \pi_{\mathbf{eb}}$

Dans le cas où $lab(r_{t_i}) \in \pi_{\mathbf{eb}}$, *TreeMerge*$(t_i \mid \widehat{t_u}) = \widehat{t_u}$.

Merge et *CMerge* Les fonctions *Merge* et *CMerge* sont formalisées dans la Figure 4.25. Elles se distinguent sur la base des deux pré-conditions ci-dessous :

– *Merge* suppose que (†) le type du noeud n parent de la forêt F_i est de la catégorie *node only* ce qui implique que, du fait de la synchronisation,

(i) aucun noeud racine d'un arbre de F_u n'est de type *String*,

(ii) toute racine d'un arbre de F_u appartient à F_i, c'est à dire plus formellement, *roots*$(F_u) \subseteq$ *roots*(F_i). Ceci est garanti parce qu'il n'y a pas d'insertion sous le noeud m.

– *CMerge* suppose que $(\dagger\dagger)$ le type du noeud n est de la catégorie 'one level below', ce qui implique que, du fait de la synchronisation,

(i) tout noeud de *roots*(F_i) a été projeté et *roots*(F_u) contient exactement les noeuds fils de m devant être retournés par *CMerge*.

Nous allons procéder à l'explication de *Merge*.

Ligne 1. Cette ligne traite le cas terminal : F_i a été complètement parcourue. Dans ce cas, $\widehat{F_u}$ est retournée et du fait de l'hypothèse $(\dagger - ii)$, il se trouve que $\widehat{F_u}$ est forcément vide.

Ligne 2. Cette ligne traite le cas où l'élément t_i traité est de type *String*. L'hypothèse $(\dagger)$ nous permet de savoir que ce noeud a été élagué à l'aide de tri-projecteur et doit donc être réintroduit. La situation capturée par cette ligne est illustrée dans l'exemple suivant.

Exemple 58. Soit la DTD D_6 donnée par les règles suivantes :

a	$\rightarrow$	$(b \mid c \mid d)*$
b	$\rightarrow$	$(e \mid c)?$
d	$\rightarrow$	$(e \mid f \mid g)*$

La DTD D_6

Soit la mise à jour u_1 spécifiée par :

$$\textbf{for } \$x \; in \; self :: a \; \textbf{return rename } \$x/c \; \textbf{with } "b"$$

Le tri-projecteur inféré pour u_1 et D_6 est donné pour chacun de ses composants par : $\pi_{\mathbf{no}} = \{a, c\}$ et $\pi_{\mathbf{olb}} = \pi_{\mathbf{eb}} = \emptyset$.
La Figure 4.27 présente la sous-forêt F_i du document initial, la sous-forêt $\widehat{F_u}$ du document projeté et mis à jour et la sous-forêt F_r du résultat de la fusion de F_i avec $\widehat{F_u}$. Comme le parent de F_i est étiqueté avec $a \in \pi_{\mathbf{no}}$, cette fusion est réalisée par la fonction *Merge*. Le parsing des deux forêts commence avec le texte 'uz' du côté de F_i et le noeud $\widehat{F_u}@1.1$ du côté de $\widehat{F_u}$. Le texte 'uz' est immédiatement retourné car il a été élagué par le tri-projecteur. Le parsing se poursuit avec les noeuds $F_i@1.1$ et $\widehat{F_u}@1.1$. Le traitement de ces noeuds est assuré par la ligne 4. ■

Ligne 3. Cette ligne traite le cas où l'étiquette a de la racine r_{t_i} de t_i appartient au tri-projecteur π (et donc, un sous-arbre de t_i à été projeté), et où r_{t_i} n'apparaît pas dans $\widehat{F_u}$ (signifiant que la projection de t_i a été supprimée par u). Lorsque $\widehat{F_u}$ n'est pas vide, cette situation est identifiée en comparant les identifiants (positions) des noeuds examinés. Dans ce cas, $r_{\widehat{t_u}} > r_{t_i}$ indique que l'arbre t_i précède l'arbre $\widehat{t_u}$ dans la forêt F_i. Donc, t_i n'est pas retourné.

Exemple 59. Considérons la DTD D_6 et la mise à jour u_2 spécifiée par :

$$\textbf{for } \$x \; in \; self :: a/b \; \textbf{where not } \$x/c \; \textbf{return delete } \$x/c$$

Le tri-projecteur inféré pour u_2 et D_6 est : $\pi_{\mathbf{no}} = \{a, b, c\}$, $\pi_{\mathbf{olb}} = \pi_{\mathbf{eb}} = \emptyset$.
La Figure 4.27 présente la sous-forêt F_i du document initial, la sous-forêt $\widehat{F_u}$ du document mis

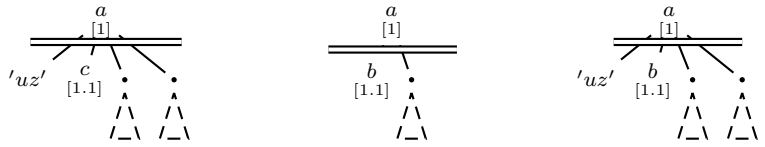
$$
\begin{array}{lll}
1 & \textit{Merge}(F_i \mid \widehat{F_u}) = & \widehat{F_u} \quad \text{si } \textit{roots}(F_i)=\emptyset, \\
 & & \textbf{sinon supposons } F_i=t_i \cdot f_i \\
2 & t_i \cdot \textit{Merge}(f_i \mid \widehat{F_u}) & \text{si } \sigma_t(r_{t_i})=text[st], \\
 & & \textbf{sinon supposons } \sigma_t(r_{t_i})=a[\mathrm{J}], \\
3 & \textit{Merge}(f_i \mid \widehat{F_u}) & \text{si } a\in\pi \text{ et si } \textit{roots}(\widehat{F_u})=\emptyset \text{ ou si } \widehat{F_u}=\widehat{t_u} \cdot \widehat{f_u} \text{ et } r_{\widehat{t_u}}>r_{t_i} \\
4 & \textit{TreeMerge}(t_i \mid \widehat{t_u}) \cdot \textit{Merge}(f_i \mid \widehat{f_u}) & \text{si } a\in\pi, \widehat{F_u}=\widehat{t_u} \cdot \widehat{f_u} \text{ et } r_{t_i}=r_{\widehat{t_u}} \\
5 & t_i \cdot \textit{Merge}(f_i \mid \widehat{F_u}) & \text{si } a\notin\pi
\end{array}
$$

(a) Définition de *Merge*

$$
\begin{array}{lll}
c.1 & \textit{CMerge}(F_i \mid \widehat{F_u}) = & \widehat{F_u} \quad \text{si } \textit{roots}(F_i)=\emptyset, \\
c.1' & & () \quad \text{si } \textit{roots}(\widehat{F_u})=\emptyset, \\
 & & \textbf{sinon supposons } \widehat{F_u}=\widehat{t_u} \cdot \widehat{f_u} \\
c.2 & \widehat{t_u} \cdot \textit{CMerge}(F_i \mid \widehat{f_u}) & \text{si } \widehat{\sigma_u}(r_{\widehat{t_u}})=text[st] \text{ ou } new(r_{\widehat{t_u}})=true, \\
 & & \textbf{sinon supposons } \widehat{\sigma_u}(r_{\widehat{t_u}})=b[K] \text{ et } F_i=t_i \cdot f_i \\
c.3 & \textit{CMerge}(f_i \mid \widehat{F_u}) & \text{si } \sigma_t(r_{t_i})=text[st] \text{ ou } \sigma_t(r_{t_i})=a[\mathrm{J}] \text{ avec } a\in\pi \text{ et } r_{\widehat{t_u}} > r_{t_i} \\
c.4 & \textit{TreeMerge}(t_i \mid \widehat{t_u}) \cdot \textit{CMerge}(f_i \mid \widehat{f_u}) & \text{si } a\in\pi, \sigma_t(r_{t_i})=a[\mathrm{J}], \text{ et } r_{t_i}=r_{\widehat{t_u}} \\
c.5 & t_i \cdot \textit{Merge}(f_i \mid \widehat{f_u}) & \text{si } a\notin\pi \text{ et } \sigma_t(r_{t_i})=a[\mathrm{J}]
\end{array}
$$

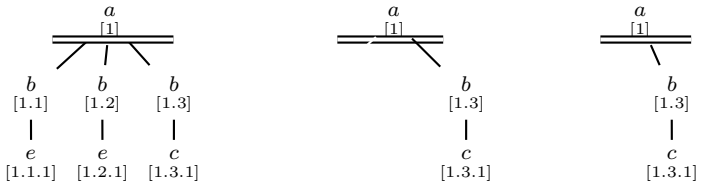
(b) Définition de *CMerge*

FIGURE 4.25 – Définition des fonctions *Merge* et *CMerge*



(1) sous-forêt F_i (2) sous-forêt $\widehat{F_u}$ (3) résultat F_r

FIGURE 4.26 – Illustration de *Merge* : line 2



(1) sous-forêt F_i (2) sous-forêt $\widehat{F_u}$ (3) résultat F_r

FIGURE 4.27 – Illustration de *Merge* : line 3

à jour projeté dans lequel les noeuds $F_i@1.1$ et $F_i@1.2$ ont été projetés (puisque $b \in \pi_{\mathbf{no}}$) puis supprimés par u_2 et la sous-forêt F_r le résultat de la fusion de F_i et $\widehat{F_u}$. Cette fusion est assurée par *Merge* car le parent de F_i est étiqueté a et que $a \in \pi_{\mathbf{no}}$. Le parsing de *Merge* commence avec les noeuds $F_i@1.1$ et $\widehat{F_u}@1.3$. Comme la position 3 du noeud $\widehat{F_u}@1.3$ est supérieure à la position 1 du noeud $F_i@1.1$, la ligne 3 de *Merge* est appliquée. Le traitement effectué par cette ligne consiste à ignorer le noeud $F_i@1.1$ et à examiner le prochain noeud dans F_i. Les noeuds qui sont examinés par *Merge* sont maintenant $F_i@1.2$ et $\widehat{F_u}@1.3$. La position de $\widehat{F_u}$ est toujours supérieure à celle de $F_i@1.2$ entraînant l'application de la ligne 3 à nouveau. Les noeuds examinés par *Merge* sont $F_i@1.3$ et $\widehat{F_u}@1.3$, ils possèdent la même position. Le traitement est maintenant assuré par la ligne 4 de *Merge*. ■

Ligne 4. Cette ligne traite le cas de la synchronisation des noeuds $r_{\widehat{t_u}}$ et r_{t_i} dont les étiquettes peuvent être différentes, du fait d'un renommage. Dans ce cas, l'arbre *TreeMerge*$(t_i \mid \widehat{t_u})$ défini ci-dessus est retourné.

Ligne 5. Cette ligne traite le cas où l'étiquette a de r_{t_i} n'appartient pas au tri-projecteur impliquant que t_i a été élagué. Dans ce cas, t_i est retourné.

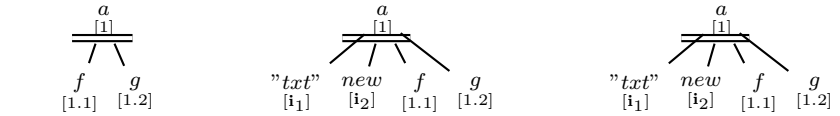
Nous procédons maintenant à l'explication de *CMerge*. Rappelons que *CMerge* fait l'hypothèse $(\dagger\dagger)$ décrite ci-dessus. Cela implique que le parcours simultané de F_i et $\widehat{F_u}$ est guidé par $\widehat{F_u}$.

Lignes c.1, c.1' Ces lignes traitent les cas terminaux. Lorsque la forêt F_i est vide (ligne c.1), $\widehat{F_u}$ est retournée. Lorsque la forêt $\widehat{F_u}$ est vide (ligne c.1') l'hypothèse $(\dagger\dagger)$ implique que la forêt F_i est vide aussi. Donc, la forêt vide est retournée.

Ligne c.2 Cette ligne traite le cas où l'arbre $\widehat{t_u}$ est un texte ou un nouvel élément. Ce dernier sous-cas est capturé par le fait que l'identifiant de la racine $r_{\widehat{t_u}}$ est nouveau ($new(r_{\widehat{t_u}})$=$true$). Donc, l'arbre $\widehat{t_u}$ est retourné. Le parsing de F_i est arrêté. La synchronisation reprend après application des autres lignes de *CMerge*.

Exemple 60. Considérons la mise à jour u_3 spécifiée par :

for $\$x$ *in* $self :: a$ **return insert** $(\text{"}txt\text{"}{<}new/{>})$ **as first into** $\$x$



(1) sub-forest F_i (2) sub-forest $\widehat{F_u}$ (3) result F_r

FIGURE 4.28 – Illustration de *CMerge* : ligne c.2

Le tri-projecteur pour u_3 et D_6 est : $\pi_{\mathbf{no}} = \pi_{\mathbf{eb}} = \emptyset, \pi_{\mathbf{olb}} = \{a\}$.
Les forêts F_i, $\widehat{F_u}$ et F_r sont présentées dans la Figure 4.28. La forêt $\widehat{F_u}$ possède deux nouveaux noeuds : un texte 'txt' suivi d'un élément vide new. Le noeud parent de la forêt $F_i@1$ est étiqueté $a \in \pi_{\mathbf{olb}}$. Par conséquent, les forêts F_i et $\widehat{F_u}$ doivent être fusionnées par *CMerge*. Les premiers noeuds examinés par *CMerge* sont $F_i@1.1$ et $\widehat{F_u}@\mathbf{i}_1$. La condition $\widehat{\sigma_u}(r_{\widehat{t_u}}) = text[st]$ est vérifiée donc la ligne c.2 est appliquée : le noeud texte $\widehat{F_u}@\mathbf{i}_1$ est retourné, le parsing avance dans $\widehat{F_u}$. Le prochain noeud de $\widehat{F_u}$ examiné est celui dont l'identifiant est $\mathbf{i}_2$ indiquant qu'il est nouveau et étiqueté new. Dans ce cas, la ligne c.2 est appliquée : le noeud $\widehat{F_u}@\mathbf{i}_2$ est retourné, le parsing avance dans $\widehat{F_u}$. Les noeuds qui sont examinés, $F_i@1.1$ et $\widehat{F_u}@1.1$, sont traités par la ligne c.5 puisque leur étiquette $f \notin \pi$. ■

Ligne c.3 Cette ligne est similaire à la ligne 3 de *Merge*. Elle traite le cas où le sous-arbre t_i de F_i a été projeté et supprimé. Notez que cette ligne traite également le cas où t_i est un texte. Dans ce cas, il est ignoré puisque le texte qui lui correspond dans $\widehat{F_u}$, ayant été éventuellement mis à jour par u, a pu être retourné par l'application de la ligne c.2.

Ligne c.4 Cette ligne traite le cas dual de celui de la ligne 4 de *Merge* (synchronisation).

Ligne c.5 Cette ligne traite le cas dual de celui de la ligne 5 de *Merge*. Dans ce cas, bien qu'elle soit implicite, l'égalité $r_{t_i}{=}r_{\widehat{t_u}}$ est vérifiée du fait de l'hypothèse $(\dagger\dagger)$. Le noeud identifié par $r_{t_i}{=}r_{\widehat{t_u}}$ appartient aux deux forêts d'où la nécessité de poursuivre le parcours sur F_i et $\widehat{F_u}$ simultanément.

Nous venons de présenter l'étape de fusion qui constitue la dernière étape de notre scénario. Pour rappel, ce scénario est composé, pour une DTD D, une mise à jour u et un document t valide pour D de quatre étapes :

1. inférence du tri-projecteur π pour u et D,
2. élagage du document t à l'aide du tri-projecteur π produisant la projection $\pi(t)$,
3. application de la mise à jour u sur la projection $\pi(t)$ retournant la mise à jour partielle $u(\pi(t))$, et
4. fusion du document original t et de la mise à jour partielle $u(\pi(t))$ pour produire le résultat final $u(t)$.

Le scénario de cette technique est schématisé par la Figure 4.29.

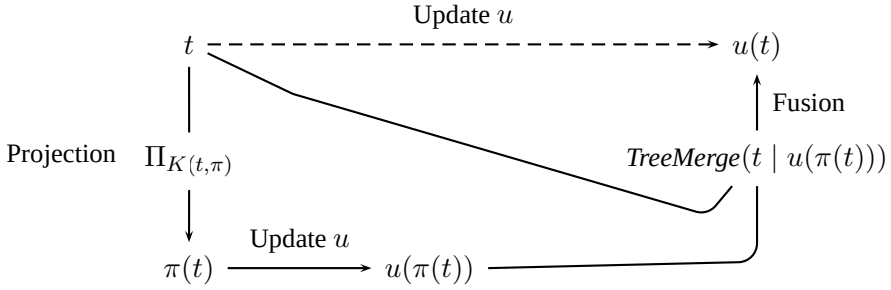


FIGURE 4.29 – Méchanisme de mise à jour : illustration

À partir de ce point, l'exposé des travaux est ma contribution : il s'agit des résultats et de la preuve de la correction de la technique de projection.

Le résultat principal énonce la correction de la fusion c.a.d. que la fusion du document original t avec la mise à jour partielle $u(\pi(t))$ produit effectivement la mise à jour du document original $u(t)$. Formellement,

Théorème 4.
Soit une DTD D, une mise à jour$u$, et π le tri-projecteur inféré pour u et D. Alors pour tout p-document $t{\in}D$ on a :

$$\textit{TreeMerge}(t \mid u(\pi(t))) \simeq u(t) \tag{4.1}$$

La preuve du Théorème 4 s'appuie sur le résultat énoncé par le Théorème suivant :

Théorème 5.
Sous les mêmes hypothèses que le Théorème 4, on a :

$$u(\pi(t)) \simeq \Pi_{\mathrm{J}}(u(t)) \text{ avec J} = \textit{dom}(u(t)) - [\textit{dom}(t) - K(t,\pi)] \tag{4.2a}$$

$$\mathbf{i}{\in}[\textit{dom}(u(t)) - \mathrm{J}] \text{ implique que } u(t)(\mathbf{i}){=}t(\mathbf{i}) \tag{4.2b}$$

Le point (4.2a) est illustré par le diagramme de la Figure 4.30. Intuitivement, il formalise le fait que la projection suivant $K(t,\pi)$ préserve l'effet de la mise à jour u : élaguer $u(t)$ en se servant des identifiants n'appartenant pas à $K(t,\pi)$ produit un document équivalent à $u(\pi(t))$. Cet élagage revient à projeter $u(t)$ suivant J.

Le point (4.2b) exprime le fait que les noeuds qui ne sont pas projetés sont invariants. Ces noeuds sont donnés dans $u(t)$ par $\textit{dom}(u(t)) - \mathrm{J}$.

La preuve du Théorème 5 est présenté dans l'Annexe C. La preuve du Théorème 4 n'est pas détaillée. Elle est fondée comme dit précédemment sur le Théorème 5 et s'effectue par une étude de cas suivant la structure de la spécification de *TreeMerge*.

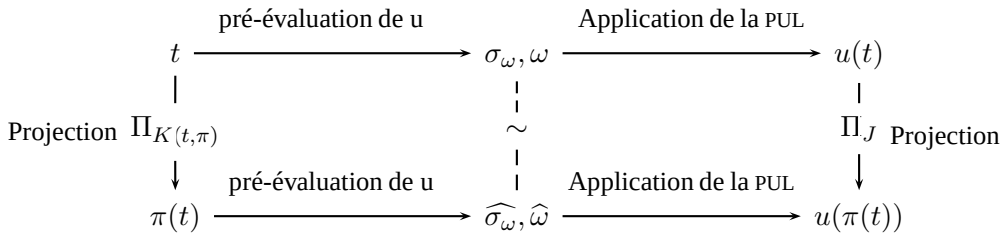


FIGURE 4.30 – Correction du tri-projecteur pour les mises à jour.

4.5 Implantation et validation de la technique

La validation de la méthode d'évaluation des mises à jour par projection consistant en l'implantation et l'expeŕimentation de la technique, a été principalement réalisée par Marina Sahakyan [5]. Nous présentons ici certains aspects techniques de cette implantation ainsi que les résultats des tests réalisés.

4.5.1 Implantation

L'implantation de la projection et de la fusion suit la spécification formelle présentée précédemment mais en abandonnant l'hypothèse que les noeuds du document initial t sont étiquetés par leur position. Cette hypothèse a été posée pour faciliter la présentation formelle de la méthode mais n'est pas nécessaire à ce stade. En effet, pour le document initial, ces positions peuvent être efficacement générées lors du parsing de celui-ci que ce soit pour la phase de projection ou pour la phase de fusion. Par contre, lors de la projection, les positions (dans t) des noeuds projetés sont enregistrées dans le document projeté $\pi(t)$. Elles sont en effet nécessaires pour la phase de fusion car elles ne peuvent bien évidemment pas être générées lors du parsing de $u(\pi(t))$: ce ne sont pas les positions des noeuds dans $u(\pi(t))$ mais dans t.

Par ailleurs, l'implantation ne nécessite pas de générer la position complète de chaque noeud du document initial t. Seule la position relative d'un noeud parmi ses frères est utile. Le rang d'un noeud de t est stocké dans $\pi(t)$ en introduisant un attribut à cet effet. L'espace nécessaire au stockage du rang des noeuds projetés a un impact limité sur les performances de la technique. Il est en général largement compensé par le gain d'espace assuré par la projection.

La projection et la fusion ont été implantées en Java en étendant la bibliothèque SAX [saxa]. Le choix de cette bibliothèque est motivée naturellement par le fait qu'elle permet le parsing de documents XML en streaming contrairement à la bibliothèque DOM [dom] qui construit une représentation interne du document en mémoire.

L'implantation de la projection ne nécessite pas véritablement de commentaires supplémentaires. La fusion est implantée en utilisant deux *threads* qui gèrent les parsings de t et de $u(\pi(t))$. Ces deux threads interagissent entre eux suivant le mode *producteur-consommateur* qui gère la synchronisation des parsing de t et de $u(\pi(t))$. Suivant ce mode, le processus producteur dépose des

5. Marina a effectué cette validation en interaction avec Noor Malla et moi-même sous la direction de Dario Colazzo.

données dans une mémoire partagée puis rentre en phase d'inactivité, le temps que le processus consommateur prélève ces données et réveille le processus producteur. L'exécution de ces deux processus est synchronisée : après initialisation des processus, le producteur ne peut pas déposer de données tant que le consommateur n'a pas prélevé les données dans la mémoire, le consommateur ne peut pas prélever les données si la mémoire est vide. Ce paradigme producteur-consommateur permet de gérer facilement la synchronisation des parsings des documents t et $u(\pi(t))$ en se basant sur les positions des noeuds courants comme spécifié par la fusion.

4.5.2 Expérimentations

Il convient de signaler qu'à ce jour, à notre connaissance, il n'existe pas de benchmark pour les mises à jour XML. Pour expérimenter notre technique, nous avons proposé 7 mises à jour conformes à XQuery Update Facility [XUP] et couvrant la plupart des mises à jour de cette norme (i.e insertion, suppression, remplacement et renomage). Les documents utilisés pour les tests ont été générés en utilisant XMark [SWK$^+$02], l'un des benchmarks les plus utilisés pour les requêtes XML. La taille de ces documents varie entre 52 Mo et 2 Go. Les mises à jour utilisées pour les tests sont présentés, ainsi que leur projecteur associé, dans la Figure 4.32.
La machine utilisée pour les expérimentations possède les caractéristiques suivantes : elle est équipée d'un processeur Intel Core 2 Duo d'une fréquence de 2.53 GHz, elle possède une mémoire vive de 2Go et elle utilise le système d'exploitation Mac OSX version 10.6.4. Nous avons fixé la taille de la mémoire virtuelle de Java à 512 Mo

4.5.2.1 Limite des moteurs mémoire-centrale

La première expérimentation réalisée vise à évaluer les limites des principaux moteurs de mises à jour mémoire-centrale à savoir : MXQuery 0.6.0 [mxq], eXist 1.2.5 [exi], Saxon EE 9.2.0.2 [saxb] et QizX Free-Engine-3.2.0 [qiza]. Nous avons mesuré la taille maximale des documents pouvant être traités par ces moteurs sans projection. La mise à jour choisie pour ce test est u_4 : elle effectue une simple suppression de noeuds et requiert le moins d'espace. La taille maximale des documents pouvant être traités par chacun de ces moteurs est donnée dans la Table 4.4. Parmi les moteurs testés, seul QizX a permis de traiter des documents d'une taille supérieure à 150 Mo ce qui s'explique par le fait qu'il utilise un format de stockage optimisé.

Moteurs	MXQuery	Saxon	eXist	QizX F-E
Taille (Mo)	52	128	148	580

TABLE 4.4 – Tailles maximales des fichiers traités

4.5.2.2 Evaluation de la technique de projection : espace

Afin de mesurer l'impact de la projection en terme de réduction de la taille des documents, nous présentons un graphique (Figure 4.31) montrant pour chaque mise à jour u_i et pour chaque document, la taille du document projeté obtenu en utilisant le projecteur π_{u_i}. L'axe des abscisses indique la taille des documents t et l'axe des ordonnées indique la taille des projections $\pi_{u_i}(t)$. Il est aisé de constater que la projection permet un gain considérable en terme d'espace. Le meilleur taux

de réduction est celui obtenu pour la mise à jour u_4 : la projection permet de réduire la taille de 2 Go (2048 Mo) à 20 Mo. Ceci est dû au fait que le projecteur π_{u_4} est très sélectif. Cependant, même avec un projecteur moins sélectif (par exemple le projecteur π_{u_5}) le gain en espace reste significatif (pour la mise à jour u_5, la projection permet de réduire la taille de 2 Go à 100 Mo).

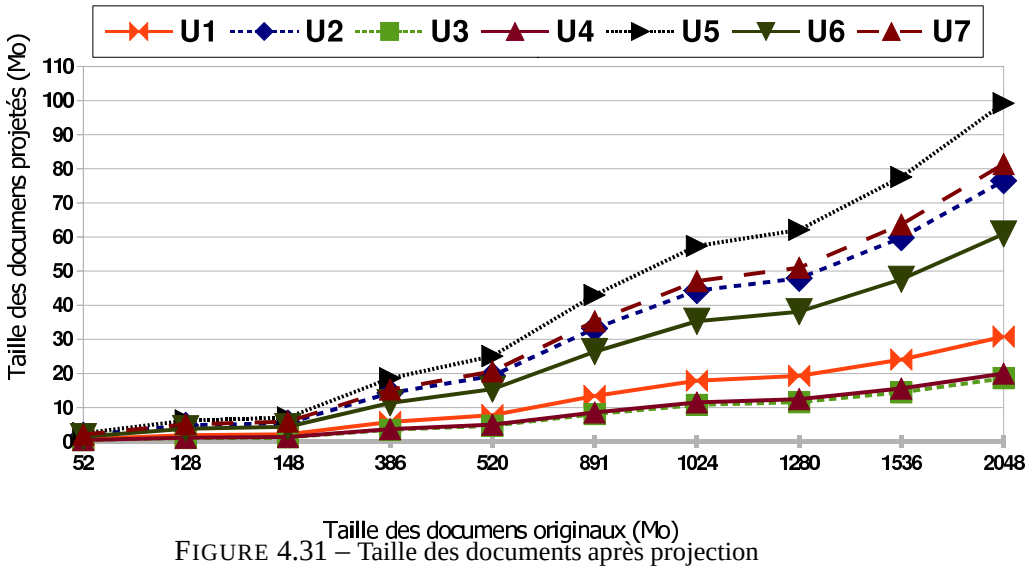


FIGURE 4.31 – Taille des documents après projection

4.5.2.3 Evaluation de la technique de projection : temps

Pour ces expérimentations, nous nous sommes focalisés sur les moteurs Saxon et QizX. Pour chacun de ces moteurs, nous avons mesuré le temps nécessaire à l'exécution des sept mises à jour sans projection et le temps nécessaire à l'exécution des mises à jour en utilisant notre technique de projection. Le temps mesuré pour notre technique de projection comprend : le temps d'exécution des étapes d'élagage, de mise à jour et de fusion plus le temps nécessaire au chargement et à la matérialisation des résultats intermédiaires produits pour et par chacune de ces étapes.

Les résultats de temps d'exécution pour Saxon sont présentés dans la Figure 4.33-(1) pour le cas sans projection et dans la Figure 4.33-(2) pour le cas avec projection. Les valeurs manquantes dans la Figure 4.33-(1) correspondent à des documents qui n'ont pas pu être chargés par Saxon en raison de leur taille. Ces graphiques montrent que notre technique atteint son objectif principal, celui de mettre à jour des documents volumineux avec des moteurs mémoire-centrale. En effet, la projection permet de mettre à jour des documents allant jusqu'à 2 Go. Cependant, nous remarquons que le temps d'exécution pour mettre à jour les documents de 52 Mo et de 128 Mo sans projection est légèrement inférieur au temps pour mettre à jour ces documents en utilisant la projection. La raison d'un tel écart est que le temps comptabilisé pour la méthode de mise à jour avec projection comprend le temps de matérialisation et de chargement des résultats intermédiaires. Ces graphiques donnent un autre éclairage sur la limite en taille des documents pouvant être traités par Saxon. En particulier, la mise à jour u_5 n'a pu être effectuée pour les documents dont la taille est supérieure à 1 Go même si la taille de leur projection est d'environ 60 Mo (donc inférieure à la limite mesurée dans le tableau 62). Nous avons pu expérimenter que la taille d'un document n'est pas le seul facteur limitatif de l'utilisation de Saxon, le nombre de noeuds du document est un autre facteur important.

Les résultats des tests pour QizX sont présentés dans les Figures 4.34-(1) et 4.34-(2). On remarque d'abord que ce moteur est moins limité que Saxon puisqu'il permet de mettre à jour des documents de taille 520 Mo là alors que Saxon s'arrête à 128 Mo. La méthode avec projection permet de mettre à jour des documents dont la taille atteint 2Go. Le temps d'exécution mesuré pour l'évaluation avec projection est meilleur que le temps d'exécution de la méthode sans projection. Ceci est dû au temps consacré par QizX à la construction d'indexes lors du chargement des documents avant leur mise à jour. Intuitivement, plus les documents sont volumineux, plus le temps nécessaire pour cette indexation est important. A titre d'exemple, si on prend le document de 52 Mo, le gain en temps pour chaque mise à jour est :

mise à jour	u_1	u_2	u_3	u_4	u_5	u_6	u_7
gain	45,4%	60,3%	74,3%	72,2%	45,2%	63,6%	24%

Pour les autres documents, le gain est similaire. Notons qu'il a été possible d'exécuter toutes les mises à jour avec projection en utilisant QizX du fait de sa capacité à traiter des documents plus volumineux que Saxon.

Les mises à jour et les projecteurs associés
Toutes les mises à jour ci-dessous partagent un binding :

```
let $doc := document("auctions.xml)
```

u_1

```
for $x in $doc/site/closed_auctions/closed_auction
where not ($x/annotation) return
insert node <annotation>Empty Annotation</annotation>
as last into $x
```

---

u_2

```
for $x in $doc/site/people/person/address
       where $x/country/text()="United States" return
(replace node $x with
 <address>
       <street>{$x/street/text()}</street>
       <city>"NewYork"</city>
       <country>"USA"</country>
       <province>{$x/province/text()}</province>
       <zipcode>{$x/zipcode/text()}</zipcode>
</address>)
```

---

u_3

```
for $x in $doc/site/regions//item/location
 where $x/text()="United States"
 return (replace value of node $x with "USA")
```

u_4

```
delete nodes $doc/site/regions//item/mailbox/mail
```

u_5

```
for $x in $doc/site//text/bold return
rename  node $x as "emph"
```

u_6

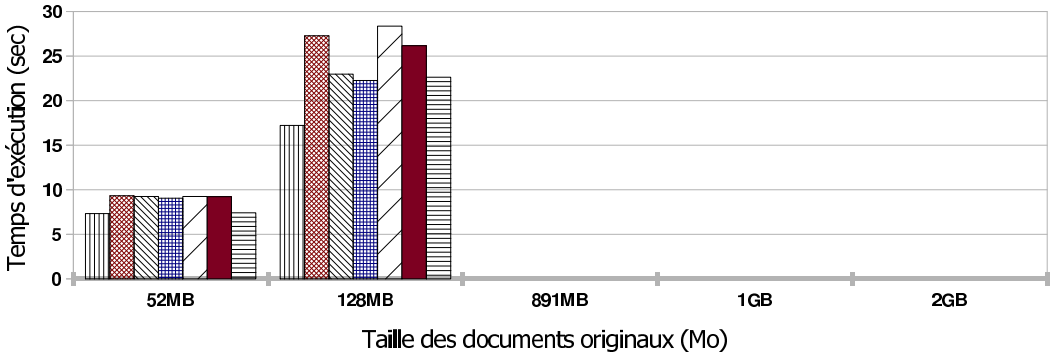
```
for $x in $doc/site/people/person
 where not($x/homepage)
 return insert node
 <homepage>www.{$x/name/text()}Page.com</homepage>
 after $x/emailaddress
```

u_7

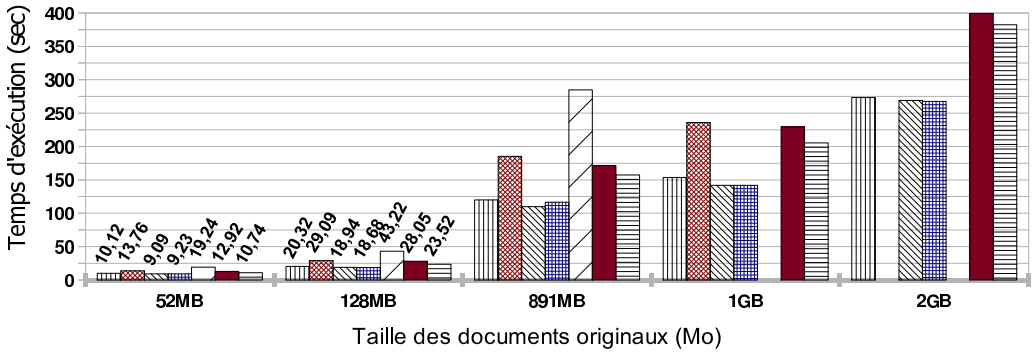
```
for $x in $doc/site/people/person,
for $y in $doc/site/people/person
   where  $x/name/text() = $y/name/text()
   and not ($y/address) and $x/country='Malaysia'
   return insert node $x/address
                      after $y/emailaddress
```

	π_{no}	π_{olb}	π_{eb}
u_1	site, closed_auctions, annotation	closed_auct	$\emptyset$
u_2	site, people, address	person, country, street, province, zipcode	$\emptyset$
u_3	site, regions, africa, asia, australia, europe, namerica, samerica, item	location	$\emptyset$
u_4	site, regions, africa, asia, australia, europe, namerica, samerica, item, mailbox, mail	$\emptyset$	$\emptyset$
u_5	site, regions, africa, asia, australia, europe, namerica, samerica, listitem, bold, mailbox, mail, item, description, text, open_auctions, open_auction closed_auctions, closed_auction, annotation, parlist	$\emptyset$	$\emptyset$
u_6	site, people, homepage	person, name	$\emptyset$
u_7	site, people	person, name, country	address

FIGURE 4.32 – Projecteurs associés aux mises à jour

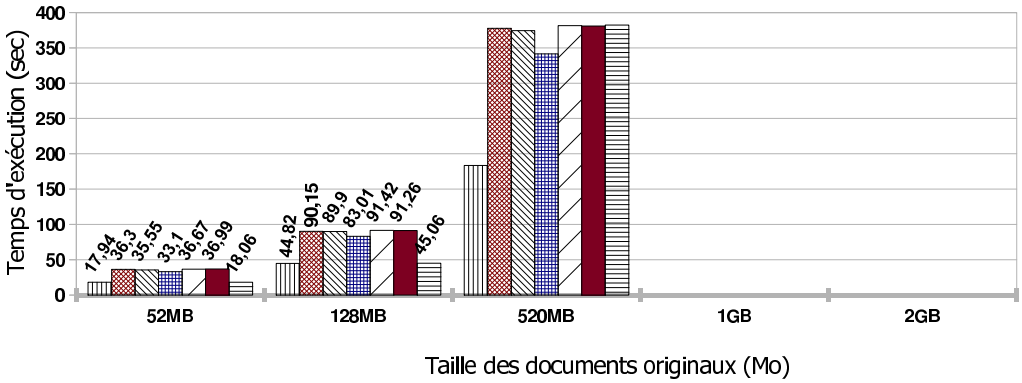


(1) mise à jour sans projection

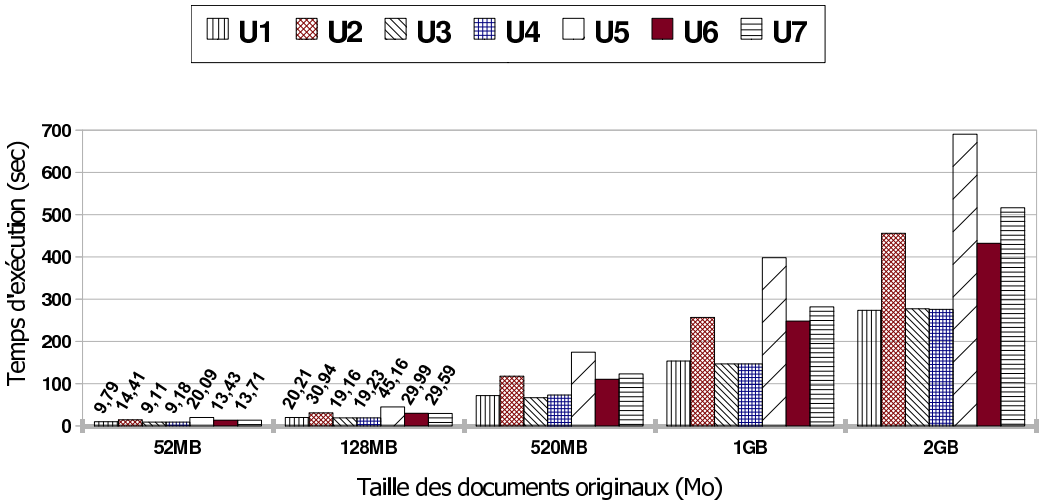


(2) mise à jour avec projection

FIGURE 4.33 – Temps d'exécution pour Saxon



(1) mise à jour sans projection



(2) mise à jour avex projection

FIGURE 4.34 – Temps d'exécution pour Qizx

4.5.2.4 Traitement de workloads

Le dernier test que nous avons effectué concerne l'utilisation de notre méthode pour exécuter un workload de mises à jour i.e. une séquence de mises à jour. En effet, il est immédiat d'étendre notre scénario d'évaluation d'une mise à jour utilisant la projection au cas d'une séquence s. Le tri-projecteur π_s associé à une séquence de mises à jour est construit par union des tri-projecteurs inférés pour chaque mise à jour. Etant donné un document t, il est projeté suivant π_s. La séquence s est évaluée sur le document projeté et la mise à jour partielle est fusionnée avec le document t. Pour cette expérimentation, nous avons considéré la séquence de mises à jour $u_1, u_2, \ldots, u_7$ et un document de taille 128 Mo afin de pouvoir utiliser Saxon. Nous avons comparé, pour chaque moteur, le temps nécessaire lorsque les mises à jour sont appliquées successivement en utilisant la projection séparément et le temps nécessaire lorsque la séquence est traitée en utilisant la projection associée à la séquence. Dans le premier cas, Saxon exécute les mises à jour en 196 secondes contre 181 secondes pour Qizx. Dans le deuxième cas, Saxon exécute les mises à jour en 82 secondes contre 64 secondes pour Qizx.

4.6 Travaux connexes

L'idée d'utiliser la projection pour l'optimisation des mises à jour XML n'a jamais été proposé. Seuls les travaux [MS03] et [BCCN06] proposent d'utiliser la projection pour les requêtes XML. Il existe des travaux [BBFV05, GRS07, GRS08, BC10] qui proposent d'utiliser l'analyse statique pour optimiser l'exécution des mises à jour. Cependant leur objectif est différent. Ils s'intéressent à la détection de l'indépendance entre les opérations de mises à jour afin d'optimiser leur exécution de manière logique.

D'autres travaux [FCB07] et [Feg10] proposent de réécrire une mise à jour u en une requête équivalente Q_u. Le but de ces approches est de pouvoir effectuer des mises à jour en utilisant des moteurs capables de traiter uniquement des requêtes et éventuellement de tirer ainsi bénéfice des optimisations développées pour l'exécution des requêtes. La requête Q_u est une transformation particulière qui va reproduire presque tout le document en entrée et modifier quelques éléments. Donc la requête Q_u va sélectionner pour les reproduire ou les modifier tous les noeuds du document, qu'ils soient potentiellement utiles à la mise à jour ou non. Il en résulte que l'utilisation d'une technique de projection telle que [MS03] ou [BCCN06] pour permettre d'évaluer Q_u sur des documents volumineux est vaine.

Dans [CGM11], les auteurs proposent une technique permettant de propager les effets d'une mise à jour en streaming. Cette technique consiste à extraire la PUL générée pour une mise à jour, à la réordonner et éventuellement effectuer certaines simplifications (par exemple, annuler l'insertion d'un élément lorsque il appartient à un sous-arbre ultérieurement supprimé) puis à appliquer la PUL résultante en streaming. Cette technique permet un gain en terme de temps d'exécution comparée à l'évaluation 'classique' des mises à jour. Cependant, elle ne permet pas de résoudre les problèmes liés à limitation des moteurs mémoire-centrale à traiter des documents volumineux : en effet, la phase de génération des PULs nécessite d'utiliser un moteur de mise à jour. D'autre part, cette technique ne permet pas d'utiliser les moteurs mémoire-centrale directement : en effet, elle nécessite d'interagir avec ces derniers pour pouvoir générer et extraire la PUL. D'autre part, cette méthode ne peut pas s'étendre au traitement de workloads, ayant besoin d'extraire les PULs l'une après l'autre

pour les simplifier avant de les exécuter.

4.7 Discussion et perspectives

Dans la première partie de ce mémoire, nous avons présenté une technique pour l'optimisation de l'exécution des mises à jour XML par des moteurs main-memory. Cette technique repose sur une analyse statique qui infère un tri-projecteur pour les mises à jour. Ce tri-projecteur est conçu dans le but de réduire la taille des documents projetés et pour assurer la correction de l'étape de fusion.

Les résultats des tests réalisés montrent que la projection permet de réduire de manière significative la taille des documents en entrée des moteurs mémoire-centrale. Ce gain en terme d'espace permet à la fois de traiter des documents plus volumineux et d'exécuter les mises à jour plus rapidement en utilisant les moteurs mémoire-centrale. Ceci montre l'impact de la projection sur l'efficacité de l'évaluation des mises à jour et suggère d'explorer des manières de réduire davantage encore la taille des documents projetés afin d'améliorer la technique.

Deux extensions ont été étudiées afin d'augmenter la précision du tri-projecteur dans le but de réduire la taille des documents projetés. Ces extensions ont toutes les deux pour point de départ l'observation suivante concernant le composant π_{olb} d'un tri-projecteur. Ce composant π_{olb} a été introduit pour répondre à deux problèmes :

- prévenir la concaténation de noeuds texte lors de la projection (voir Exemple 37) et
- permettre une fusion correcte du document initial et du document parriellement mis à jour pour les insertions (voir Exemple 36).

Ces problèmes sont de nature différente et la solution unique qui a été donnée (avec succès) comporte quelques limites relativement au critère de précision : la projection en mode 'one level below' projette des noeuds inutiles.

Donc afin d'améliorer la précision du tri-projecteur, nous avons étudié chacun des problèmes ci-dessus séparément. Ceci a donné lieu à la proposition d'un multi-projecteur apportant plus de précision dans le cas de l'insertion d'une part et dans le cas de l'accès à du texte d'autre part. Ce multi-projecteur est introduit à l'aide d'exemples.

4.7.1 Extension du tri-projecteur pour la prise en charge du type des mises à jour

Dans le cadre de sa thèse [Sah11], Marina Sahakyan a proposé une nouvelle classification du projecteur pour les mises à jour qui prend en compte la nature des mises à jour atomiques. Une étude de cas approfondie a été menée pour déterminer les réels besoins de chaque opération de mise à jour et définir la projection que requiert cette opération. Des tests sont en cours pour valider l'approche.

L'exemple suivant est fourni dans le but de montrer l'amélioration pouvant être apportée au tri-projecteur pour une mise à jour d'insertion.

Exemple 61. La Figure 4.35 spécifie, comme d'habitude, une DTD D, un document initial t et une mise à jour u qui entend insérer deux nouveaux éléments e et c en premiers fils des noeuds étiqueté par a.

Cette figure présente deux scénarios :

– le scénario de projection/fusion utilisant le tri-projecteur π associé à la mise à jour$u$ donné dans la Figure 4.35-(5) : l'application de ce projecteur sur le document t sélectionne tous les fils des noeuds étiquetés a.
La raison motivant la projection de tous les fils de a est de permettre à la fonction *Merge* qui fusionne t avec $u(\pi(t))$ de retourner les noeuds $u(\pi(t))@\mathbf{i}_{\mathtt{new}}$ et $u(\pi(t))@\mathbf{j}_{\mathtt{new}}$ en premier lorsqu'elle examine les fils des noeuds $t@1$ et $u(\pi(t))@1$.

– le scénario de projection/fusion utilisant un 4-projecteur associé à la mise à jour u.
Le projecteur étendu est présenté dans la Figure 4.35-(8). Le composant $\pi_{\mathbf{olb}}$ est vide, il est remplacé ici par un nouveau composant $\pi_{\mathbf{asfirst}}$ qui contient l'étiquette a du noeud sous lequel l'insertion peut être effectuée. L'application de ce projecteur sur le document t a pour effet de projeter les noeuds étiquetés a seuls, sans leur fils (voir Figure 4.35-(9)). L'algorithme de fusion est adapté pour prendre en compte ce nouveau composant. Le comportement de la fusion lorsque les noeuds $t@1$ et $u(\pi^{ext}(t))@1$ sont examinés consiste à donner la priorité au document $u(\pi^{ext}(t))$ en retournant tous les fils de $u(\pi^{ext}(t))@1$ en premier : ceci est permis par l'observation $lab(t@1)\in\pi_{\mathbf{asfirst}}$. Les noeuds fils de $t@1$ seront produits par la fusion après.

■

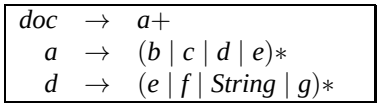
La prise en compte des autres opérations (les mises à jour d'insertion avec les autres directions et les mises à jour de remplacement) suit plus ou moins le même principe que celui de l'exemple ci-dessus : un nouveau composant est ajouté pour chaque genre d'opération et l'algorithme de fusion est adapté pour prendre en compte le traitement spécifique de celle-ci. Une analyse relativement complexe doit être faite pour traiter le cas d'expressions de mise à jour incluant plusieurs genres d'opérations atomiques ciblant potentiellement le même noeud comme c'est le cas pour la mise à jour :

for $\$x$ in $\mathrm{self} :: \mathrm{doc}/\mathrm{a}$ **where** $\$x/c$
return (**insert** <e/><c/> **as first into** $x,
replace $x/b **with** <d/>)

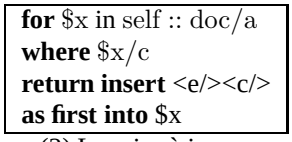
Pour certaines combinaisons, l'utilisation du composant $\pi_{\mathbf{olb}}$ constitue un bon compromis entre efficacité et simplicité, et ceci semble être confirmé par les expérimentations.

4.7.2 Extension du tri-projecteur pour les noeuds texte

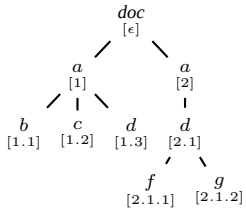
L'extension du tri-projecteur pour la projection des noeuds texte est une perspective de mon travail de thèse que je compte explorer. L'idée sous-jacente est analogue à celle utilisée pour l'extension prenant en compte la nature des opérations de mises à jour et consiste à introduire un nouveau composant $\pi_{\mathbf{so}}$ (pour *string only*) aux trois composants existants ($\pi_{\mathbf{no}}$, $\pi_{\mathbf{olb}}$ et $\pi_{\mathbf{eb}}$) du projecteur. Ce composant contient les étiquettes des parents des noeuds texte qui doivent être projetés. Le comportement de ce composant consiste à projeter de manière systématique le noeud voisin



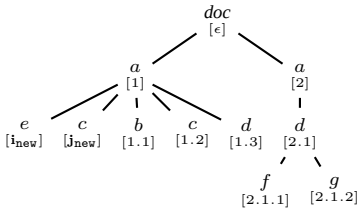
(1) La DTD D



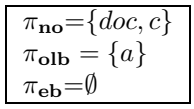
(2) La mise à jour u



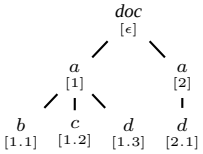
(3) Le document t



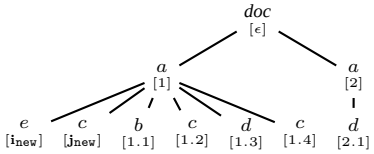
(4) Le résultat final $u(t)$



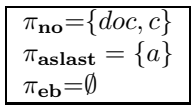
(5) Le tri-projecteur π



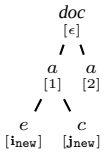
(6) La projection de t pour π



(7) La mise à jour partielle $u(\pi(t))$



(8) Le projecteur π^{ext} pour u (9) La projection de t pour π^{ext}



(10) La mise à jour partielle $u(\pi^{ext}(t))$

FIGURE 4.35 – Mise à jour par projection : tri-projecteur vs projecteur étendu

suivant de chaque noeud texte projeté. Ces noeuds éléments voisins projetés de cette manière (par effet de bord de la projection de noeuds texte) servent de séparateur potentiel entre deux noeuds texte fils d'un même noeud.

Nous procédons de la même manière que pour le cas des mises à jour en présentant un exemple qui donne l'intuition de l'usage et du comportement de ce composant du projecteur.

Exemple 62. Considérons la DTD, la mise à jour et le document initial t de la Figure 4.36. La mise à jour u est censée supprimer les noeuds c fils des noeuds a contenant le texte "ot".

Cette figure présente deux scénarios :

– le scénario de projection/fusion utilisant le tri-projecteur π associé à la mise à jour$u$ donné dans la Figure 4.36-(5) : l'application de ce projecteur sur le document t sélectionne tous les fils des noeuds étiquetés a. Le document projeté est presque aussi volumineux que le document original du fait du nombre de noeuds fils du noeud étiqueté a.
Les noeuds éléments fils de a (à l'exception de c qui est cible d'une suppression) sont projetés uniquement pour éviter que les textes "fo" et "ot" ne soient concaténés.

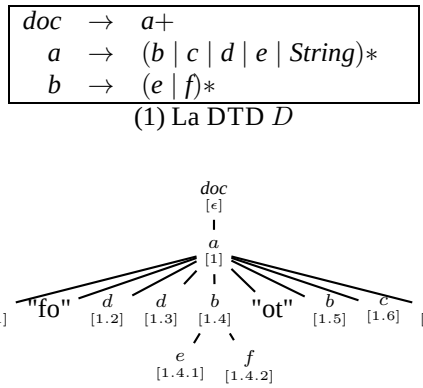
– le scénario de projection/fusion utilisant un 5-projecteur π^{ext} associé à la mise à jour u.
L'élagage du document t à l'aide de π^{ext} renvoie les noeuds texte, fils de $t@1$ et pour chacun d'eux, le noeud frère droit (voir Figure 4.36-(9)).
La fusion du document t et de la mise à jour partielle $u(\pi_1^{ext}(t))$ devra retourner dans leur position d'origine les noeuds qui n'ont pas été projeté et propager l'effet de la mise à jour.

■

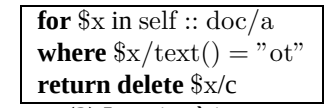
Dans l'exemple précédent, les noeuds texte projetés sont simplement accédés par la mise à jour pour vérifier une condition. Le cas où les textes sont modifiés ou supprimés nécessite un peu plus d'attention notamment pour la fusion comme c'est le cas de la mise à jour :

for $\$x$ in $\mathrm{self} :: \mathrm{doc/a}$
where $\$x/\mathrm{text}() = \text{"ot"}$
return (**delete** $\$x/c$, **insert** "goal" **as last into** $\$x$)

Intuitivement, une analyse des différentes combinaisons de l'accès à du texte avec les opérations de mises à jour de texte est nécessaire pour déterminer le traitement adéquat à effectuer pour la projection et la fusion.

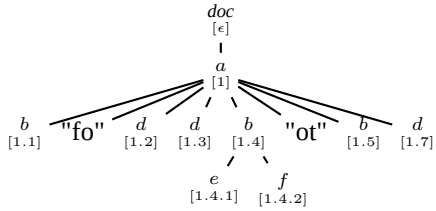


(1) La DTD D

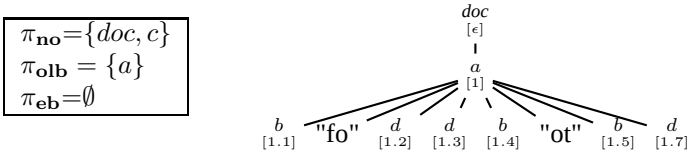


(2) La mise à jour u

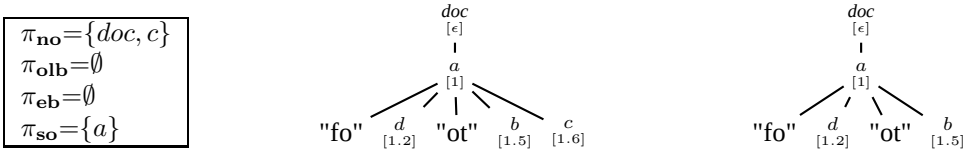
(3) Le document t



(4) Le résultat final $u(t)$



(5) Le tri-projecteur π (6) La mise à jour partielle $u(\pi(t))$



(8) Le projecteur π_1^{ext} pour u (9) La projection de t pour π_1^{ext} (10) La mise à jour partielle $u(\pi_1^{ext}(t))$

FIGURE 4.36 – Extension du tri-projecteur pour la projection des noeuds texte.

Deuxième partie

Mises à jour de documents XML temporels

Introduction de la deuxième partie

Le temps est une dimension importante dans de nombreux domaines d'applications tels que la finance, la comptabilité, la gestion d'inventaire, l'observation de phénomènes naturels, la gestion de dossiers médicaux, etc. Les bases de données conventionnelles ne permettent pas de prendre en compte la dimension temporelle de manière systématique. Elles ont vocation à stocker les valeurs courantes des données d'une application. L'évolution des données résulte, en général, de mises à jour entraînant souvent la destruction des valeurs antérieures considérées obsolètes. Les bases de données temporelles fournissent un cadre pour gérer l'évolution, dans le temps, des données. Elles permettent de garder trace de l'évolution des données et d'accéder à des états antérieurs de ces données, par interogation par exemple.

L'étude de modèles et de langages de requêtes temporels a fait l'objet de travaux de recherche dans le contexte relationnel (voir [CT05] pour un état de l'art). Dans la littérature, deux modèles temporels relationnels cohabitent : le modèle dit abstrait et le modèle dit concret. Le modèle abstrait est très simple et se focalise principalement sur une représentation de l'évolution de la base instant après instant : une instance temporelle est une séquence finie d'instances où d'états. Ce modèle représente le temps de manière implicite. Il sert de base à la formalisation de langages de requêtes, de contraintes et de mises à jour. Cependant, il n'est pas approprié au stockage des données. Stocker une séquence d'instances nécessite un espace de stockage important et comporte fréquemment la répétition de données non modifiées sur des périodes de temps plus ou moins longues.
Le modèle concret vise à représenter les données temporelles de manière concise en tirant parti justement de la répétition des données non modifiées. Ce modèle se sert de l'ajout dans les relations d'un attribut particulier dont le domaine est l'ensemble des intervalles de temps : cet attribut sert à *estampiller* les données et capturer la période de validité de chaque n-uplet. De cette manière, les données inchangées dans le temps ne sont pas dupliquées, elles sont estampillées de manière adéquate. Ces deux modèles, d'un point de vue représentation, permettent de capturer les mêmes données même si les langages de requêtes "temps implicites" et les langages de requêtes "temps explicites" sont différents par leur expressivité.

Gérer l'évolution des données dans un document XML est bien évidemment un sujet dont l'intérêt est motivé par l'émergence de nouvelles applications telles que le commerce électronique, ou le monitoring du contenu du web. A l'instar des bases de données temporelles relationnelles, plusieurs extensions temporelles ont été proposées pour XML. Tous les travaux [CAW99, GS03b, BKTT04a, WZZ05, WZ08, RV08] utilisent le modèle estampillé et se focalisent sur l'interrogation de documents XML estampillés. Il est assez surprenant de constater que très peu de travaux expliquent comment les documents XML estampillés sont construits ou maintenus. Or, cette construction/maintenance n'est pas triviale pour plusieurs raisons. La première est liée au volume des documents estampillés : la taille des documents estampillés empêche de les stocker en mémoire centrale et oblige à favoriser les traitements en streaming par exemple. La deuxième raison est liée à la possibilité d'avoir plusieurs documents estampillés correspondant à une même évolution d'un document, ces différents documents estampillés pouvant différer par l'espace nécessaire à leur stockage. La construction de documents estampillés "compacts" est rendue compliquée par la nécessité de manipuler les documents en streaming.

Dans cette thèse, nous abordons le problème de la construction et la maintenance efficaces de documents XML estampillés. Nous suivons l'approche de [CT05] en travaillant avec un modèle

abstrait et un modèle concret pour capturer l'évolution de documents XML. Par analogie avec le modèle relationnel, un document abstrait est une séquence de documents XML statiques ; un document concret est un document XML estampillé. L'utilisation des documents estampillés permet d'encoder de manière efficace les documents abstraits. Nous définissons ensuite le lien entre les documents abstraits et leurs encodages. Intuitivement, un document estampillé est un encodage d'un document temporel si chaque document de la séquence peut être extrait du document estampillé. Cette définition autorise un même document abstrait à avoir plusieurs encodages, certains encodages pouvant être plus compacts que d'autres. Afin de permettre de comparer les différents encodages d'un document abstrait en terme de stockage, nous introduisons un ordre partiel appelé ordre de compacité.

Dans cette partie, notre objectif consiste à proposer deux méthodes pour générer des encodages d'un document abstrait. La première méthode est destinée à construire et à maintenir des documents estampillés dans le cas général pour lequel aucune hypothèse n'est faite sur l'évolution du document XML : celle-ci peut être le résultat de l'édition successive du document, d'un mélange d'éditions et de traitements effectués au travers de langages de mise à jour, par exemple. La seconde méthode s'intéresse à la situation où le document évolue sous l'effect de mise à jour spécifié par un langage tel que XQuery Update. Pour chaque méthode, l'objectif est à la fois de permettre le traitement de documents volumineux et de générer des documents les plus compacts possibles. Etant confronté au problème de limitation en terme de mémoire centrale, la première méthode est conçue de sorte à manipuler les documents estampillés en streaming. Cela lui permet effectivement de traiter des documents volumineux mais l'empêche de générer, dans certains cas, des documents compacts. La deuxième méthode est basée sur la technique de projection développée pour l'optimisation des mises à jour. L'utilisation de la projection possède plusieurs avantages. Premièrement, cette technique permet de traiter des documents estampillés volumineux en utilisant les moteurs mémoire-centrale. Deuxièmement, n'importe quel moteur peut être utilisé par cette technique sans modification de ce dernier. Troisièmement, les mises à jour posées pour des documents statiques ne nécessitent pas d'être réécrites, pour tenir compte des estampilles. Enfin, les documents obtenus par cette méthode sont plus satisfaisants du point de vue du stockage comparés aux documents obtenus par la méthode qui traite le cas général. Ceci est démontré par une validation expérimentale.

Cette partie est organisée en deux chapitres. Dans le chapitre 5, nous présentons les principales approches des bases de données relationnelles temporelles ainsi que les travaux en lien avec la gestion des données semi-structurées temporelles. Dans le chapitre 6 nous présentons notre contribution qui consiste à développer deux méthodes permettant de générer des encodages compacts pour les documents XML temporels.

Chapitre 5

Gestion des documents XML temporels : état de l'art

Introduction

Ce chapitre est destiné à la présentation des extensions temporelles pour les données relationnelles et les données semi-structurées. La section 5.1 se focalise sur les données relationnelles largement étudiées et dont nous réutilisons certaines notions. La section 5.2 présente les extensions temporelles pour les données semi-structurées, précurseurs de XML. La section 5.3 est consacrée à la gestion des données XML temporelles.

5.1 Données relationnelles temporelles

Les bases de données relationnelles ont fait l'objet de plusieurs études comme peut en témoigner l'état de l'art réalisé par Chomicki et al. [CT05]. Ces études ont porté sur le développement de modèles de données et sur la proposition de langages de requêtes temporels tels que TSQL [SK95] et SQL/TP [Tom98]. Concernant les modèles de données temporels, Chomicki et al. [CT05] ont été les premiers à distinguer entre les modèles abstrait et concret. Cette distinction a pour but de fournir un niveau d'abstraction assez élevé permettant la formulation des requêtes temporelles tout en s'assurant d'une évaluation efficace de celles-ci. Le modèle abstrait considère une séquence d'instances correspondant à des états successifs de la base de données. Il permet de spécifier ce qui est stocké dans la base de données et la façon dont on l'interroge. Il est utilisé pour définir la sémantique des langages de requêtes et de mises à jour.

Exemple 63. Soit le schéma de base de données $\mathcal{R} = \{\text{Patients(nss, nom, service)}\}$. Soit $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3$ une séquence d'instances sur le schéma $\mathcal{R}$ où chaque $\mathbf{I}_i$ correspond à l'état de la base de données à l'instant i (pour $i = 1..3$). Cette séquence modélise une base de données temporelles.

nss	nom	service
771	ben	hémato
793	dana	ophtalmo

$\mathbf{I}_1$

nss	nom	service
771	ben	hématho
793	dana	ophtalmo
891	roy	traumato

$\mathbf{I}_2$

nss	nom	service
771	ben	hématho
891	roy	traumato
1001	lara	cardio

$\mathbf{I}_3$

■

Il est évident qu'il n'est pas intéressant d'un point de vue pratique de considérer le modèle abstrait pour stocker les données temporelles : le stockage de la séquence requiert un espace important vu que le stockage de chaque n-uplet est répété autant de fois qu'il est valide dans le temps (par exemple le n-uplet (771, ben, hémato) qu'on retrouve dans les trois instances). Le modèle concret offre une représentation compacte des données temporelles. Il considère une instance estampillé dans laquelle une estampille est associée chaque n-uplet, i.e un intervalle de temps $[t_1, t_2[$ capturant la période de sa validité.

Exemple 64. La version estampillée de la séquence $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3$, notée $\mathbf{I}^{est}$, est donnée ci-dessous. Un nouvel attribut appelé temps est rajouté au schéma de la base de données pour stocker l'estampille de chaque n-uplet. Les estampilles sont des intervalles de temps ouverts à droite. Le symbole Now est utilisé dans les bases de données temporelles pour indiquer le temps courant. Désormais, tous les noeuds qui étaient répétés sont stockés une seule fois avec une estampille qui capture leur période de validité.

nss	nom	service	*temps*
771	ben	hématho	[1,Now[
793	dana	ophtalmo	[1,2[
891	roy	traumato	[2,Now[
1001	lara	cardio	[3,Now[

$\mathbf{I}^{est}$

■

Remarquez qu'il est trivial de retrouver la séquence $\mathbf{I}_1, \ldots, \mathbf{I}_n$ associée à toute instance estampillée $\mathbf{I}^{est}$: il suffit pour cela de projeter cette dernière en fonction d'une valeur temporelle.

Pour une même base de données temporelles il peut y avoir plusieurs encodages possibles. Certains encodages sont plus compacts que d'autres, c'est à dire, qu'ils permettent d'encoder les mêmes informations en utilisant moins d'espace. Bien évidemment, il est souhaitable en général de pouvoir disposer des encodages les plus compacts possibles.

Exemple 65. Soit $\mathbf{J}^{est}$ l'instance estampillée donnée ci-dessous.

nss	nom	service	*estampille*
771	ben	hématho	[1,2[
771	ben	hématho	[2,Now[
793	dana	ophtalmo	[1,2[
891	roy	traumato	[2,3[
891	roy	traumato	[3,Now[
1001	lara	cardio	[3,Now[

$\mathbf{J}^{est}$

$\mathbf{J}^{est}$ est un encodage pour la séquence de l'exemple 63. Toutefois, $\mathbf{I}^{est}$ est plus compacte que $\mathbf{J}^{est}$ puisque le n-uplet (771, ben, hématho) valide dans l'intervalle [1,Now[est répliqué dans $\mathbf{J}^{est}$ une fois avec l'estampille [1,2[et une autre fois avec l'estampille [2,Now[alors que dans $\mathbf{J}^{est}$ ce n-uplet est stocké une seule fois avec l'estampille [1,Now[. La même remarque s'applique au n-uplet (891, ben, hématho) qui est valide dans l'intervalle [2,Now[. ■

Il existe une technique appelée *coalescing* [BSS96] qui permet de produire des relations sous forme normale en fusionnant les n-uplets répliqués. Pour notre exemple, l'application de cette technique sur $\mathbf{J}^{est}$ produirait $\mathbf{I}^{est}$.

Mises à jour des données temporelles Comme c'est le cas pour la plupart des applications qui manipulent des données, les bases de données temporelles nécessitent un mécanisme de modification des données i.e les mises à jour. Comme pour les requêtes, les mises à jour des données temporelles sont spécifiées relativement au niveau abstrait. Une traduction de ces mises à jour est donc nécessaire pour permettre leur exécution sur l'instance estampillée.

Cela nous conduit à parler des *historiques* qui constituent un cas particulier des bases de données temporelles. Les historiques introduisent une restriction sur l'évolution des instances qui consiste à limiter la modification à l'instance courante. Dans ce cas, la séquence $\mathbf{I}_1, \ldots, \mathbf{I}_n$ vérifie les conditions suivante en considérant $\mathbf{I}_1$ l'instance initiale de la base de données :
chaque instance $\mathbf{I}_i$ est obtenue en mettant à jour l'instance $\mathbf{I}_{i-1}$ pour $i > 1$. Les résultats développés pour le cas général s'appliquent à des historiques de manière directe.

5.2 Données semi-structurées temporelles

Chawathe et al. [CAW99] ont été les premiers à proposer un modèle temporel pour les données semi-structurées. Leur approche consiste à étendre le modèle de données *Object Exchange Model* (OEM) l'un des précurseurs de XML. Ils proposent une technique pour annoter les changements dans un graphe OEM. Ils étudient les aspects liés à l'interrogation des changements ainsi que les mises à jour.

Le modèle OEM consiste en un graphe dont les noeuds représentent les objets modélisés et dont les arcs étiquetés relient les objets à leur contenu. Ce modèle considère un langage de mises à jour propre qui permet de spécifier les actions suivantes :

- création d'un nouveau noeud,
- modification du contenu d'un noeud existant,
- ajout d'un arc entre deux noeuds existants et
- suppression d'un arc existant.

L'extension proposé par [CAW99] consiste à annoter les noeuds ou les arcs du graphe OEM par les mises à jour dont ils sont cibles. Chaque annotation garde trace du temps auquel la mise à jour est appliquée mais aussi du type de cette dernière (l'une des quatre actions citées plus haut). Le graphe annoté, appelé DOEM pour Delta-OEM, est utilisé uniquement comme niveau logique pour permettre la formulation des requêtes temporelles. Il est encodé dans le format OEM afin de pouvoir utiliser les moteurs d'exécution existants sans modification. Les requêtes formulées sur le graphe annoté DOEM sont traduites vers des requêtes classiques pour être évaluées sur des graphes OEM.

De ce point de vue, l'approche de [CAW99] paraît similaire à l'approche de [Tom98] qui distingue deux niveaux d'abstraction des données temporelles.

D'autres travaux [DBJ99] et [COQ04] ont proposé des extensions temporelles pour les données semi-structurées. Dans [DBJ99] les auteurs proposent une approche qui consiste à annoter les arcs d'un graphe avec diverses informations servant de méta-données. Parmi ces informations, il considèrent un intervalle de temps qui capture la période de validité des arcs. Les auteurs de [DBJ99] ne font référence à aucun langage de mises à jour et ne traitent pas l'évaluation des requêtes temporelles. Dans [COQ04], les auteurs proposent carrément un méta-modèle leur permettant d'instancier les modèles temporels développés dans [DBJ99] et [CAW99]. Ils ne considèrent pas les mises à jour des données temporelles.

5.3 Données XML temporelles

Bon nombre de travaux se sont intéressé à la gestion des données XML temporelles. Une bibliographique des principaux travaux dans le domaine est disponible dans [Gra04]. Certains d'entre eux ([CTZ00] et [MACM01]) s'inspirent de techniques issues de la gestion des versions des documents (*document versioning*) notamment des versions des logiciels [Tic85] ou de l'édition collaborative et proposent d'utiliser le *diff* (appelée aussi *delta*) pour garder trace de l'évolution des données XML. D'autres travaux tels que [BKTT04a], [WZ08] et [RV08] s'inspirent des approches développées dans le contexte des données relationnelles et proposent des techniques pour la construction et la maintenance de documents estampillés. On distingue alors entre deux familles d'approches : les approches basées sur les deltas et les approches basées sur les documents estampillés.

5.3.1 Approches basées sur les deltas

Avant de présenter les approches basées sur les deltas, il convient de présenter d'abord les deltas pour XML. Un delta, appelé par abus *diff* en référence aux algorithmes qui permettent de le calculer, décrit la différence entre deux arbres XML t et t'. Il consiste en un ensemble d'opérations primitives qui, appliquées sur t produisent t'. Ces opérations peuvent être soit l'insertion, la suppression d'un sous-arbre ou la modification d'un noeud. L'opération de déplacement d'un sous-arbre (appelée move) est parfois considérée. A l'origine, les deltas ont été développés pour les textes où ils trouvent leur champ d'applications dans plusieurs domaines tels que l'alignement de séquences d'ADN, la gestion de versions (SVN) et les protocoles de synchronisation des fichiers dans le réseaux. Ils ont été adaptés par la suite pour prendre en considération les données hiérarchiques telles que HTML et XML (voir [CAH02] et [Pet05] pour la liste de travaux en lien).

L'utilisation des deltas pour garder trace de l'évolution des documents XML consiste à stocker la version initiale V_0 et la version courante V_{now} plus la suite des deltas $\Delta_1, \ldots, \Delta_n$ obtenus en appliquant l'algorithme de diff entre deux versions successives. Cette stratégie est illustrée dans la Figure 5.1. Lorsqu'une nouvelle version est produite, le delta entre cette version et la version la plus récente V_{now} est intégré à l'historique. Cette stratégie présente des inconvénients. D'abord, elle nécessite de stocker deux versions du document pour lequel on gère l'historique (appelés par la suite document historisé) : la version initiale et la version finale. De plus, retrouver une version V_i nécessite d'appliquer successivement les deltas, opération qui peut s'avérer coûteuse si V_i

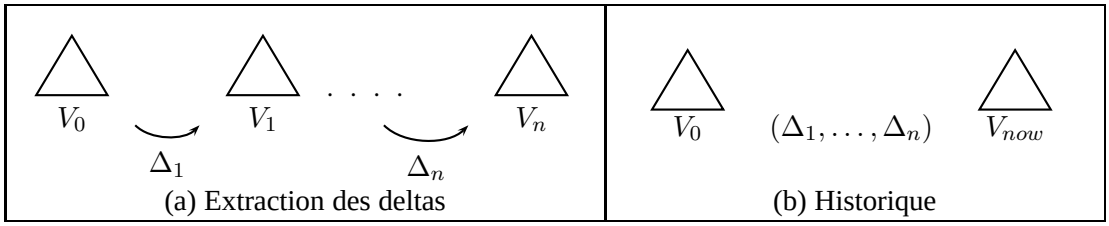


FIGURE 5.1 – Illustration des approches basées sur les deltas

correspond à une version assez récente du document historisé. Il existe une stratégie symétrique qui consiste à stocker la dernière version V_n et la suite des deltas inverses $\Delta_n^r, \ldots, \Delta_1^r$ qui permettent de retrouver les versions antérieures. Cette stratégie permet de reconstruire la version V_{i-1} à partir de la version V_i.

Dans [MACM01], les auteurs utilisent les deltas pour construire et maintenir l'historique des documents XML extraits du web. L'approche qu'ils proposent suit la deuxième stratégie, c'est à dire celle qui consiste à stocker la dernière version plus les deltas inverses. Les deltas considérés dans [MACM01] sont bi-directionnels, c'est à dire qu'ils permettent à la fois de calculer à partir d'une version V_i la version précédente V_{i-1} et la version suivante V_{i+1}. Ces deltas occupent un espace plus important que les deltas classiques ce nécessite de les compresser avant de les stocker.

Dans [CTZ00], les auteurs proposent une approche qui utilise les *edit script*, une autre forme des deltas, générés entre versions successives des documents. Ils utilisent l'algorithme RCS [Tic85] développé pour la comparaison des versions de logiciels. Cet algorithme opère sur des fichiers textes et, par conséquent, ne permet pas de prendre en compte la structure hiérarchique de XML. Ceci oblige les auteurs à développer une méthode de partitionnement pour comparer les documents XML. Ceci constitue une différence majeure par rapport à l'approche de [MACM01] qui utilise un algorithme de diff développé spécifiquement pour XML [CAM02].

Limites des approches basées sur les deltas L'utilisation des deltas pour garder l'historique de l'évolution d'un document XML possède des inconvénients qui ont été soulignés par Buneman et al. [BKTT04a]. D'après ces auteurs, le fait que les deltas soient purement syntaxiques et ne véhiculent aucune information sémantique sur l'évolution des objets les rend particulièrement difficiles à utiliser pour expliquer les changements subis par les objets modélisés dans les documents ou bien pour répondre à des requêtes temporelles. L'évaluation des requêtes temporelles nécessite alors la reconstruction de toutes les versions à partir des deltas. Le coût d'une telle reconstruction est fonction du nombre et de la taille des deltas à appliquer. En gros, plus un document a subi de modifications, plus ce coût sera élevé. Enfin, le dernier inconvénient, que nous constatons nous mêmes, concerne le coût de la génération même des deltas qui peut être prohibitif lorsque les documents comparés sont volumineux sans oublier les problèmes de limitation en terme de mémoire centrale qui peuvent entraver les approches basées sur les deltas.

Les auteurs de [BKTT04a] préconisent l'utilisation des documents estampillés comme approche alternative aux travaux utilisant les deltas. L'avantage d'utiliser les documents estampillés est double. D'abord, les données et les changements sont disponibles à partir d'une seule structure de données contrairement aux approches basées sur les deltas. Enfin, il est possible d'envisager une évaluation efficace des requêtes puisque les techniques d'indexation développées pour le cas des documents statiques peuvent aussi être utilisées pour accélérer l'évaluation des requêtes une fois

ces techniques adaptées pour la prise en compte des donnés temporelles.

5.3.2 Approches basées sur les documents estampillés

Il existe différentes approches qui proposent d'étendre les documents XML pour y incorporer l'aspect temporel. Ces approches développent des techniques permettant de construire et de maintenir des documents estampillés. Nous présentons ces différentes approches en nous focalisant sur :

(a) les hypothèses faites par ces approches concernant la nature des données traitées et les évolutions considérées,

(b) la manière dont les mises à jour ou les informations concernant de l'évolution (deltas, edit script,...) sont exploitées lors de la construction ou la maintenance des documents estampillés,

(c) l'efficacité de la construction et de la maintenance des documents estampillés.

5.3.2.1 Archivage des données scientifiques

Buneman et al. [BKTT04a] ont été les premiers à proposer une technique pour construire un document estampillé à partir d'une séquence de versions (ou *snapshots*). Leur technique est destinée à l'archivage des données scientifiques représentées dans le format XML. Ces données ont une caractéristique particulière comparées aux données XML classiques : elles respectent des contraintes de clés. Les clés sont exploitées par la technique développée par [BKTT04a] pour la construction des documents estampillés.

Les clés pour XML ont été étudiées par ces mêmes auteurs dans [BDF$^+$01] et [BDF$^+$03] ainsi que par Bouchou et. al dans [BCF$^+$07] où elles sont définies à base de grammaires d'attributs. Dans [BDF$^+$01], les clés sont définies à base de chemins XPath. La structure d'une clé en XML est donnée par une paire $(P_{\texttt{cib}}, \{P_1, \ldots, P_n\})$ où $P_{\texttt{cib}}, P_1, \ldots, P_n$ sont des chemins tels que $P_{\texttt{cib}}$ cible les noeuds qui sont contraints par la clé et $P_1, \ldots, P_n$ désignent les noeuds dont le contenu détermine le noeud contraint par la clé ie les chemins $P_1, \ldots, P_n$ jouent le même rôle que les attributs pour les clés relationnelles. Pour qu'un document t soit satisfait par une clé $(P_{\texttt{cib}}, \{P_1, \ldots, P_n\})$ il faut qu'à partir de chaque noeud retourné par $P_{\texttt{cib}}$, l'évaluation de chaque chemin P_i, pour $1 \leq i \leq n$, retourne une valeur unique. La notion de clé pour XML est illustrée à travers l'exemple suivant.

Exemple 66. Soit le document de la Figure 5.2 qui représente les données bibliographiques.

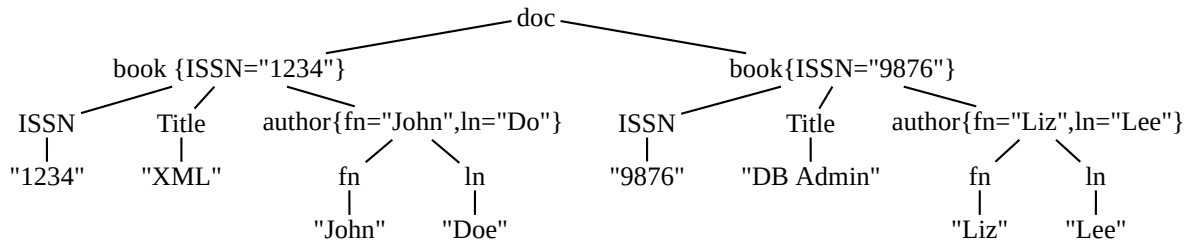


FIGURE 5.2 – Document annoté avec ses clés

Ce document satisfait deux clés :

– $(/doc/book, \{ISSN\})$ qui spécifie qu'un élément book est identifié par le contenu de son noeud fils étiqueté ISSN, et
– $(/doc/book/author, \{fn, ln\})$ qui spécifie qu'un élément author est identifié par ses fils fn et ln.
Concernant la première clé, il est aisé de vérifier qu'il n'existe pas deux noeuds book dont le noeud fils ISSN possède le même contenu. Pour la deuxième clé, on peut aussi facilement vérifier que chaque élément author possède

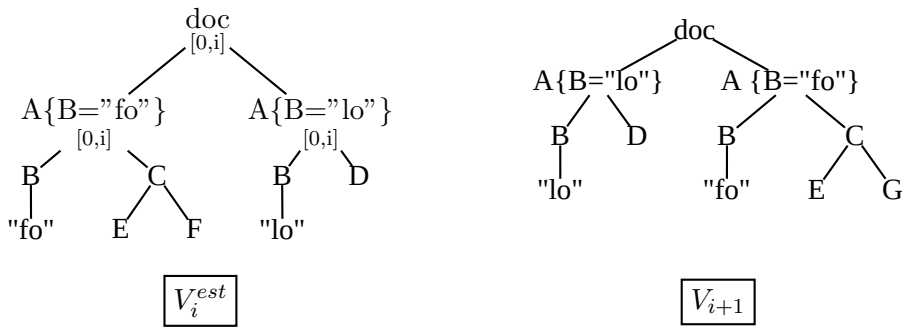
Afin de pouvoir illustrer la technique de [BKTT04a] de construction des documents estampillés, les noeuds du document de la Figure 5.2 qui possèdent une clé sont annotés avec des égalités de la forme $p_0 = val_0, \ldots, p_n = val_n$ où chaque p_i représente un chemin relatif appartenant à la spécification de la clé et où val_i est la résultat de l'évaluation de p_i sur le document. Par exemple, le noeud book le plus à gauche de ce document est annoté par ISSN=1234 pour indiquer qu'il correspond à l'unique noeud book dont le numéro ISSN est 1234. ■

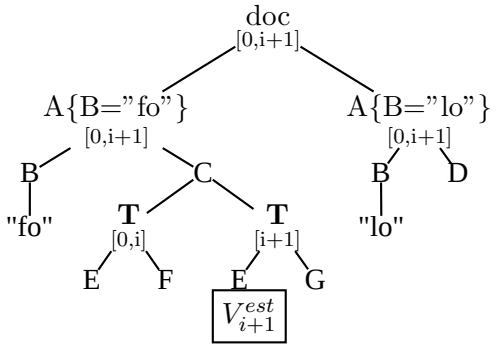
Dans [BKTT04a], les auteurs expliquent que les données scientifiques sont régulières et respectent un schéma logique qui spécifie des contraintes de clés puisqu'elles modélisent des objets qui sont uniques (des protéines, des éléments chimiques). D'autre part, l'évolution de ces données consiste en de simples insertions d'éléments nouveaux qui correspondent à de nouvelles découvertes (découverte d'une nouvelle protéine ou d'un nouvel élément chimique) et par conséquent préservent les contraintes de clés. Ces deux hypothèses sont suffisantes pour utiliser les clés comme identifiants pour les noeuds.

La principale contribution de [BKTT04a] est une technique de compactage permettant d'encoder une séquence de snapshots $V_1, \ldots, V_n$ en un document estampillé V^{est}. Cette technique repose sur l'utilisation des clés pour identifier les noeuds décrivant les mêmes objets dans différents snapshots. Son but est de fusionner ces noeuds et de leur associer une estampille capturant tous les instants de temps auxquels ils sont valides.

La technique de compactage développée dans [BKTT04a] est un processus itératif où chaque itération consiste à intégrer le snapshot V_{i+1} au document estampillé V_i^{est} encodant les versions $V_1, \ldots, V_i$. Cette opération nécessite une étape préalable ayant pour but de calculer les valeurs de clés pour les noeuds de V_{i+1} et de V_i^{est} et de les annoter avec les paires (clé=valeur). La compactage de V_{i+1} avec V_i^{est} se fait niveau par niveau en suivant la structure arborescente de ces documents. Il est illustré à l'aide de l'exemple suivant.

Exemple 67. Soit un document estampillé V_i^{est} et un snpashot V_{i+1} déjà annotés avec les valeurs de leur clés. La seule clé satisfaite par ce document est spécifiée par $(/doc/A, \{B\})$ indiquant que la valeur que les noeuds A sont déterminé par le contenu de son noeud fils B. Le résultat du compactage de ces deux documents est donné par V_{i+1}^{est}.





Les estampilles sont indiquées pour les documents V_i^{est} et V_{i+1}^{est} en-dessous des étiquettes des noeuds. Remarquez que certains noeuds ne possèdent pas d'estampilles puisqu'ils les héritent de leur parent ou ancêtres.

La procédure de compactage de V_i^{est} et V_{i+1} consiste à analyser leurs noeuds niveau par niveau et à identifier les noeuds ayant la même valeur de clés pour les fusionner. Le traitement débute à la racine de ces documents qui est généralement inchangée dans le temps. Pour notre exemple, la racine doc de V_i^{est} est retournée après incrémentation de la borne supérieure de son estampille de 1. Le traitement se poursuit avec les noeuds du niveau en dessous de la racine et consiste à examiner les clés des noeuds étiquetés A de chacun des deux arbres. Cet examen conclut que le noeud A de V_i^{est} se trouvant à gauche et ayant pour valeur de clé "fo" correspond au noeud A de V_{i+1} à droite. Ces deux noeuds sont alors fusionnés : l'estampille du noeud A de V_i^{est} est incrémentée de 1 et leurs noeuds fils sont examinés de manière analogue. Le même traitement est effectué pour le noeud A de V_i^{est} qui se trouve à droite et le noeud A de V_{i+1} lui correspondant (celui qui se trouve à gauche).

Nous nous focalisons maintenant sur le traitement des fils du noeud A dont la valeur de la clé est "fo". Il est clair que les noeuds B de V_i^{est} et de V_{i+1} respectivement, dont le contenu sert de valeur de clé, sont fusionnés. Le traitement des noeuds C voisins de noeuds B est différent puisque ni le contenu de C n'est utilisé par une clé et ni C lui même ne contient de clé. Par ailleurs, le contenu de ce noeud a été modifié à partir de l'instant i+1 : le noeud F a été remplacé par le noeud G. Dans ce cas, la version du sous-arbre enraciné en C pendant [0,i], i.e le sous-arbre provenant du document estampillé V_i^{est}, est stockée à l'intérieur d'un noeud spécifique étiqueté **T** estampillé avec [0,i]. La nouvelle version de ce sous-arbre est stockée à l'intérieur d'un autre noeud étiqueté **T** ayant pour estampille $[\mathrm{i}+1]$.

■

Discussion concernant l'approche de [BKTT04a] L'approche proposée dans [BKTT04a] pour gérer les documents XML temporels possède quelques inconvénients. Le premier concerne la technique de compactage qui se limite à la fusion des noeuds possédant des clés. Les noeuds qui ne possèdent pas de clé et dont le contenu change se traitent en répliquant leurs sous-arbres sans effectuer une analyse approfondie de ces derniers. Cela entraîne un coût de stockage qui peut être important lorsqu'il y a beaucoup de noeuds qui ne possèdent pas de clés et que le contenu de ces noeuds change souvent. Les auteurs semblent conscients de ce problème et proposent d'y remédier en appliquant le diff sur le contenu de ces noeuds. L'information retournée par le diff serait utilisée pour encoder de manière plus compacte le contenu des noeuds sans clés qui évolue dans le temps.

Bien qu'une telle solution permette d'éviter la duplication de plusieurs sous-arbres et de parvenir à construire des documents estampillés compacts, aucune spécification de la solution proposée n'est fournie. Seul un exemple donné dans la version journal [BKTT04b] permet de comprendre l'idée sous-jacente. Nous illustrons cette idée par l'exemple 67 en nous focalisons sur les sous-arbres de V_i^{est} et V_{i+1} enracinés en C puisque C ne possède pas de clé. Le traitement de ces sous-arbres est illustré dans la Figure 5.3.

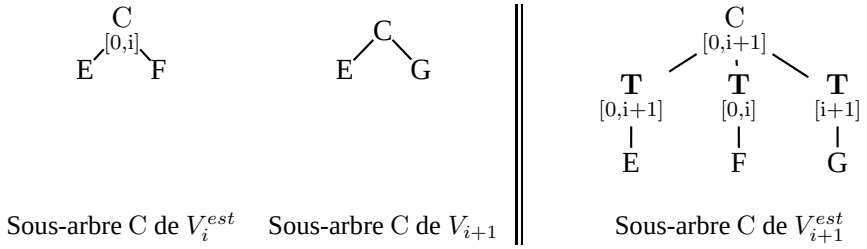


FIGURE 5.3 – Traitement des noeuds sans clé [BKTT04b]

Il est aisé de constater qu'entre les instants i et i+1 le noeud F fils de C est remplacé par le noeud G alors que son noeud voisin E ne change pas. L'application du diff sur ces deux arbres détecterait la suppression du noeud étiqueté F et l'insertion du noeud G. La solution proposée dans [BKTT04b] consiste alors à fermer l'estampille du noeud F indiquant qu'il n'est plus valide après l'instant i et à insérer le noeud G avec l'estampille i+1 qui désigne que la validité de ce noeud commence à l'instant i+1 L'estampille d noeud E est $[0, \mathrm{i} + 1]$ puisque ce dernier n'a pas subi de modification tout au long de cette période. Comme pour le cas précédent, le noeud **T** est utilisé pour garder trace de l'évolution du contenu du noeud C dans le temps. Ce noeud sera estampillé avec un intervalle capturant la période de validité de son noeud fils.

L'autre inconvénient de l'approche de [BKTT04a] concerne la modification du modèle de donnée par l'introduction d'un noeud spécifique **T** pour garder trace de l'évolution des noeuds qui ne possèdent pas de clés. Cette modification a d'abord un impact sur l'espace de stockage qui sera plus important du fait du stockage de ces noeuds spécifiques. Elle a aussi un impact sur l'évaluation des requêtes temporelles puisque les requêtes formulées par les utilisateurs doivent être traduites vers des requêtes qui prennent en compte les noeuds **T**.

Une autre limitation de l'approche de [BKTT04a] est liée au fait que l'ordre des noeuds n'est pas géré. Cela rende cette technique inutilisable pour le traitement des documents XML génériques pour lesquels l'ordre des noeuds importe. Remarquez que dans l'exemple 67, les noeuds A ont permuté leur position dans le snapshot V_{i+1} mais cette permutation n'est pas reflétée dans le résultat V_{i+1}^{est}. Les auteurs de [BKTT04a] expliquent que l'ordre n'a pas d'importance pour les données qu'ils manipulent puisqu'elles vérifient des contraintes de clés. Il faut préciser tout de même qu'ignorer l'ordre coûte une étape supplémentaire lors du compactage des snapshots. Cette étape consiste à trier, à chaque niveau parcouru, les noeuds de ce niveau sur la base de la valeur de leur clé. Le but de ce tri est d'accélérer l'étape de fusion. Dans l'exemple 67, ce tri permet de trouver rapidement le noeud A de V_{i+1} correspondant à chaque noeud A de V_i^{est} lors du compactage de ces deux documents.

L'autre point qu'il convient de citer concerne le coût de l'étape de pré-traitement effectuée avant chaque application de la méthode de compactage pour annoter les noeuds des documents à compacter. Le coût de cette étape dépend de la taille des documents V_i^{est} et $Vi + 1$ et de la spécification des clés. Cette étape nécessite, bien évidemment, de parcourir les documents V_i^{est} et

$Vi+1$ et à évaluer sur chacun d'eux les chemins spécifiés dans les clés. Elle nécessite aussi de stocker de façon temporaire le contenu de certains sous-arbre qui servent de valeur pour certaines clés. Cette stratégie peut s'avérer coûteuse du point de vue du stockage mais aussi du point de vue du traitement (la comparaison des valeurs des clés) effectué lors du compactage du document estampillé et du snpashot. Les auteurs proposent d'appliquer une fonction de hachage permettant de calculer une "emprunte" qui sera stockée à la place des sous-arbres et utilisée pour accélérer la comparaison des noeuds.

Pour résumer, l'approche proposée par [BKTT04a] est destinée principalement aux données scientifiques. Ces données sont deux caractéristiques :
- elles respectent des contraintes de clés,
- les mises à jour les plus fréquentes consistent en de simples insertions d'éléments qui préservent les clés.

La technique proposée pour la l'encodage de l'historique repose sur les clés et n'utilise aucune information sur les mises à jour. Enfin, l'efficacité de cette technique a été étudiée par les auteurs qui semblent conscients du problème lié à la limitation en terme de mémoire disponible pour la construction de l'encodage. Ils proposent une stratégie de compression visant à optimiser le stockage des noeuds. Cette stratégie consiste à associer à chaque étiquette une valeur numérique qui sera manipulée à la place de l'étiquette d'origine du noeud. Toutefois, il est nécessaire d'utiliser et de maintenir un dictionnaire de données permettant de retrouver les correspondances entre les étiquettes et les valeurs numériques qui leur ont été affectées. L'autre stratégie d'optimisation consiste à stocker les clés et les données dans des fichiers séparés afin de pouvoir charger les données et les clés à la demande. Bien qu'ils se soient soucier de problèmes liés à la limitation de mémoire centrale, les auteurs de [BKTT04a] ne précisent pas dans les résultats d'expérimentations la taille des fichiers qu'ils peuvent traiter avec leur technique.

5.3.2.2 Modélisation des documents XML temporels à base de graphes [RV08]

Rizzolo et al. [RV08] modélisent les documents temporels à l'aide d'un graphes. Ils étudient les contraintes temporelles que devront respecter les documents temporels et proposent à la fois des méthodes pour permettre de vérifier si ces documents satisfont certaines contraintes et des méthodes pour réparer les documents qui ne satisfont pas ces contraintes. Ils s'intéressent également à l'interrogation temporelle et proposent d'étendre le langage XPath [XPa] pour y incorporer l'aspect temporel. Ce langage de chemins est au coeur du langage de mises à jour qu'ils développent pour mettre à jour leur document temporel.

Le modèle de données proposé dans ce travail consiste en un graphe acyclique dont les sommets correspondent aux noeuds du document XML pour lequel l'historique est construit et dont les arcs, estampillés, sont utilisés pour capturer l'une des deux relations différentes :
- la relation parent-fils classique qui lie les noeuds d'un document XML,
- la relation de référence introduite pour permettre le partage des sous-arbres.

Le deuxième type d'arc permet de capturer le déplacement d'un sous-arbre d'un noeud vers un autre noeud sans avoir besoin de dupliquer le contenu de ce sous-arbre.

Exemple 68. Considérons le document de la Figure 5.4 qui modélise l'historique d'édition d'un livre. Cet historique est modélisé par un graphe dont chaque sommet possède un identifiant (indiqué dans un cercle) et une étiquette (indiquée au dessus de cet identifiant). Les deux types d'arcs sont distingués par l'utilisation de traits différents : un trait plein est utilisé pour les arcs parent-fils et

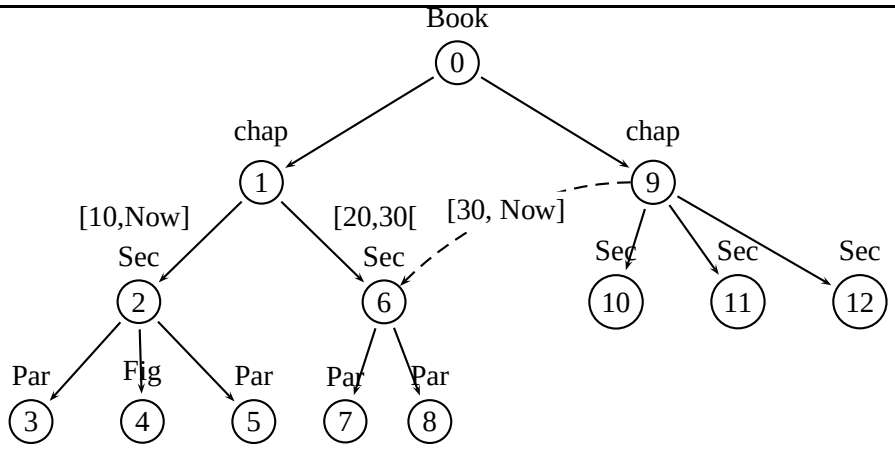


FIGURE 5.4 – Modélisation sous forme de graphe d'un document XML temporel [RV08]

un trait discontinu pour les arcs référence. Les estampilles sont utilisées uniquement lorsqu'il y a besoin d'indiquer une évolution des données. Par exemple, l'arc liant le noeud 1 au noeud identifié par 2 est estampillé par [10,Now[indiquant que le sous-arbre enraciné 2 est inséré à l'instant 10 (et n'est pas modifié depuis cet instant). Le seul arc référence de cet exemple lie le noeud 9 au noeud 6 indiquant que le sous-arbre enraciné en 6 a été déplacé à l'instant 40 à l'intérieur du noeud 9. ■

Les auteurs proposent deux stratégies pour traduire la représentation graphique vers XML : une stratégie qui optimise le stockage des sous-arbres déplacés et une stratégie qui se contente de répliquer ces sous-arbres à l'intérieur des noeuds à chaque fois qu'ils sont déplacés. Pour les deux stratégies la traduction consiste à associé à chaque sommet un noeud et de transformer chaque arc parent-fils en imbriquant noeuds de manière à retrouver la structure hiérarchique représentée par le graphe. La stratégie qui optimise le stockage des sous-arbres déplacés s'appuie sur le mécanisme des ID-IDREF utilisé pour la définition de références dans XML. La représentation sous forme XML du graphe de la Figure 5.4 est donnée ci-dessous.

```
<Book>
  <chap>
    <Sec [10, Now]>
      <Par/> <Fig/> <Par/>
    </Sec>
    <Sec [20, 30] ID=6>
      <Par/> <Par/>
    </Sec>
  </chap>
  <chap IDREF=6 [30, Now] >
    <Sec/> <Sec/> <Sec/>
  </chap>
</Book>
```

Le langage temporel de chemins proposé par [RV08], appelé TXPath, étend le langage XPath de

la manière suivante. Au niveau syntaxique, le langage TXPath introduit de nouveaux mots clé pour permettre d'accéder aux estampilles et utilise deux opérateurs (meet et contains) pour comparer les intervalles de temps. Au niveau sémantique, l'évaluation d'un chemin TXPath considère à la fois l'ensemble de noeuds accédés par le chemin et l'ensemble des estampilles leur correspondant. Une définition formelle de la sémantique du langage est fournie dans [RV08].

Le langage de mises à jour proposé dans [RV08] s'inspire d'un langage de mise à jour développé par [TIHW01b] pour les documents XML statiques. Ce langage ressemble à XQuery Update et permet les opérations de bases sur les arbres, à savoir l'insertion d'un nouveau noeud, la suppression ou le remplacement d'un noeud existant. Ce langage ne permet pas le déplacement d'un sous-arbre d'un noeud vers un autre noeud.

Les différentes primitives de mises à jour sont traduites vers des mises à jour qui permettent de prendre en compte la manière dont les données temporelles sont encodées. Les insertions de nouveaux noeuds nécessitent seulement d'attribuer des estampilles aux noeuds insérés. La suppression de noeuds existant s'effectue en fermant des estampilles, i.e en substituant leur borne supérieure par l'instant courant. La mise à jour des arcs, sans doute la mise à jour qui nécessite le plus de traitement, s'effectue en fonction de la stratégie de stockage qui a été retenue (stratégie avec ou sans réplication). Il est important de souligner que seuls les noeuds correspondant à l'état courant des données temporelles peuvent être modifiés, i.e les mises à jour n'affectent pas les données du passé.

Les auteurs expliquent que la mise à jour des arcs peut introduire des cycles et rendre les données inconsistantes. Ils étudient les différents types d'inconsistance et proposent des techniques pour les détecter et les éliminer.

Discussion autour de l'approche de [RV08] L'approche proposée par [RV08] possède certains avantages relativement à celle de [BKTT04a]. D'abord, elle s'applique à toute sorte de données XML contrairement à la technique de [BKTT04a] qui se limite aux données scientifiques. De plus, les mises à jour considérées autorisent les principales opérations définies sur les arbres (insertions, suppressions et déplacement) et spécifiées par des expressions complexes qui utilisent des chemins XPath. Dans [BKTT04a], les mises à jour se limitent aux insertions et ne sont pas générées par un langage de mises à jour particulier. Le dernier point de différence entre les deux techniques concerne la gestion de l'ordre des noeuds. La technique de [BKTT04a] ne se soucie pas de cet ordre puisqu'elle est destinée à gérer des données non-ordonnées. La technique de [RV08] permet quant à elle de gérer l'ordre.

Les avantages de la technique de [RV08] ont un impact sur son efficacité. La mise en oeuvre de cette technique requiert l'utilisation de structures de données auxiliaires et d'indexes pour accélérer l'évaluation des chemins temporels. Ces indexes et ces structures ont un coût de construction et de maintenance qu'il faut prendre en compte. L'extraction des snapshots, opération nécessaire pour l'évaluation des requêtes et des mises à jour, nécessite au moins deux parcours des documents estampillés dans le cas où la stratégie de stockage sans réplication est utilisée. Afin de se convaincre de la nécessité de ces deux parcours, il suffit de considérer le document de l'exemple 68. Dans ce cas, retrouver le snapshot courant nécessite d'extraire le sous-arbre enraciné au noeud identifié par 6 soit à l'intérieur du noeud chap identifié par 9. Comme le parcours s'effectue en profondeur et de gauche à droite, il est clair qu'un seul parcours du document estampillé ne permet pas cette extraction.

Le dernier point qu'il est intéressant d'aborder concerne la possibilité de traitement des documents volumineux. Les auteurs de [RV08] semblent avoir été confrontés aux problèmes rencontrés par l'utilisation de moteurs mémoire-centrale et ce dans une expérience précédente [MRV04]. Ils ont résolu ce problème dans [RV08] en utilisant des SGBD relationnels et en exploitant les indexes. D'après l'étude expérimentale qu'ils ont effectuée, il est possible de mettre à jour des documents de taille 100 MO. Les auteurs ne disent rien concernant d'éventuelles limites de taille des documents pouvant être traités mais on peut supposer que le fait de recourir à un SGBD relationnel permet de surpasser les limitations en terme de mémoire centrale.

5.4 Conclusion

Nous venons de présenter différents travaux sur la gestion des données temporelles. Les premiers travaux présentés concernent les données relationnelles. Ces travaux ont traité différents aspects allant de la modélisation jusqu'à l'interrogation en passant par la représentation efficace de données temporelles. Depuis quelques années, l'attention s'est focalisée sur XML, le format des données du Web. A l'instar du cas relationnel, les problématiques classiques liées à la modélisation, à la représentation et à l'interrogation des données temporelles ont été abordées pour XML. D'autres problématiques qui n'étaient pas identifiées dans le cas relationnel ont émergé dans le cas XML. Ces problématiques sont liées à l'utilisation d'un nouveau paradigme permettant de traiter les données XML directement en mémoire centrale en utilisant les moteurs mémoire-centrale. La principale limitation est l'incapacité des moteurs mémoire-centrale à traiter des documents volumineux. Or, les documents temporels sont supposés à priori plus volumineux que les documents statiques rendant impératif la proposition d'une solution pour gérer les documents temporels. Les approches proposées dans la littérature contournent ce problème soit en utilisant des méthodes ad-hoc tel que dans [BKTT04a] soit en recourant aux SGBD relationnels pour bénéficier de leur techniques d'optimisation comme dans [RV08] .

Chapitre 6

Gestion des documents XML temporels

Introduction

L'état de l'art qui précède montre qu'il existe assez peu de travaux traitant de l'estampillage des documents XML alors que tous les articles font l'hypothèse que les documents XML temporels sont estampillés. La question que nous avons donc voulu traiter de manière plus approfondie est celle de la gestion des estampilles i.e. la création et la maintenance de documents estampillés. Pour cela nous avons suivi la démarche de [Tom98] en introduisant un modèle abstrait qui permet de voir un document XML temporel simplement comme une suite de documents, et un modèle concret celui des documents estampillés. Le modèle abstrait est un outil de référence qui va nous permettre de spécifier **les** documents estampillés équivalents au sens où ils correspondent au même document abstrait. En particulier, ces deux niveaux de modélisation permettent de comparer les documents estampillés équivalents en termes d'efficacité i.e.d'espace mémoire nécessaire au stockage. Les contributions centrales de ce chapitre sont deux techniques introduites dans [BBTC11] pour créer et maintenir des documents temporels.

La première technique est générale. Elle ne fait aucune hypothèse sur la manière utilisée pour modifier le document (le document peut évoluer par édition, mise à jour, ...). Cette technique s'avère utile dans plusieurs contextes tels que le maintien à jour d'entrepôts de données temporels [MACM01, WZ08] ou l'archivage de pages web [SJ08], où des documents (souvent appelés snapshots) sont produits à intervalles réguliers et nécessitent d'être intégrés dans une archive globale. La deuxième situation traite le cas où les documents temporels correspondent à un historique dans lequel chaque document de la séquence est obtenu de la mise à jour du document qui le précède. Cette situation survient pour des scénarios classiques où des applications interagissent avec des SGBD qui permettent de gérer les données XML (requêtes, mises à jour, etc).

Le but de ces deux méthodes est à la fois de produire un document estampillé compact mais aussi de pouvoir traiter des documents volumineux par l'utilisation de moteurs mémoire-centrale.

Ce chapitre est organisé comme suit. La section 6.1 introduit les notations et définitions nécessaires à la présentation de notre contribution. La section 6.2 présente les deux méthodes d'encodages développées. Les résultats des tests effectués pour valider les deux méthodes sont expliqués en section 6.3.

6.1 Modèles XML temporels

Cette section est construite sur la base des définitions et notations introduites dans le chapitre 2. Nous allons modifier dans la suite certaines de nos conventions de notation. En effet, il est naturel d'utiliser t pour désigner un instant. De ce fait, un document XML (arbre) est maintenant noté d (comme document) à la place de t.

Le domaine temporel est spécifié par l'ensemble des entiers naturels $\mathbb{N}$ muni de l'ordre $\leq$. Dans la suite nous utilisons des estampilles modélisées simplement par des intrevalles dont le domaine Int est défini par :

$$Int = \{[t_1, t_2[\mid t_1 \in \mathbb{N}, t_2 \in \mathbb{N} \cup \{Now\}, t_1 < t_2 \text{ ou } t_2 = Now\}$$

où Now est une variable capturant l'instant courant. L'inclusion et l'intersection des estampilles sont définies de manière triviale.

Exemple 69. Considérons les tests et les opérations sur les estampilles suivants :

- $[0, 10[\subseteq [0, Now[$,
- $[20, Now[\subseteq [10, Now[$ et
- $[10, Now[\cap [0, 30[= [10, 30[$.

■

Nous procédons maintenant à la présentation du modèle abstrait et du modèle concret pour les documents XML en nous inspirant du travail de Toman et al. [CT05]. Dans ce travail, le modèle abstrait pour les bases de données relationnelles est défini par un séquence d'instances. Nous procédons exactement de la même manière pour les documents XML à ceci près que nous devons prendre quelques précautions pour pouvoir garder trace de l'evolution des éléments d'un document XML.

Rappelons que dans le chapitre 2, un document XML d est modélisé par la notion de store dont les identifiants capturent les noeuds afin de leur associer une étiquette ou du texte et éventuellement un contenu (fils). Ici, il est nécessaire d'étendre la notion de store pour associer à chaque noeud (à chaque identifiant du domaine du store) une information sous la forme d'un identifiant dit *explicite* servant, comme dans le modèle objet à capturer l'identité des éléments. En effet lors de son évolution, un document peut changer de différentes manières. L'exemple suivant va nous servir à illuster ceci.

Exemple 70. Considérons les documents de la Figure 6.1.

Pour chaque noeud, son identifiant est indiqué entre "[" et "]" comme d'habitude et son identifiant explicite quant à lui est précédé du symbole #. Nous constatons les trois points suivants :

- la structure arborescente du document peut changer et le domaine du store avec, mais les identifiants explicites restent inchangés. Dans la figure, un noeud étiqueté e et repéré par l'identifiant explicite $\#6$ a été inséré, dans le document d_j entre les deux noeuds ayant pour identifiants explicites $\#1$ et $\#2$ respectivement. A cause de cette insertion, les identifiants de store alloués dans d_j et d_k aux noeuds repérés par les identifiants explicites $\#2$ et $\#5$ sont différents de ceux alloués initialement à ces mêmes noeuds dans d_i. Par exemple, le noeud étiqueté a et repéré par $\#2$ possède l'identifiant de store 2 dans le document d_i et l'identifiant (de store) 3 dans d_j et d_k. Remarquons que le domaine de d_j est $\{\epsilon, 1, 2, 1.1, 1.2, 2, 3, 3.1\}$.

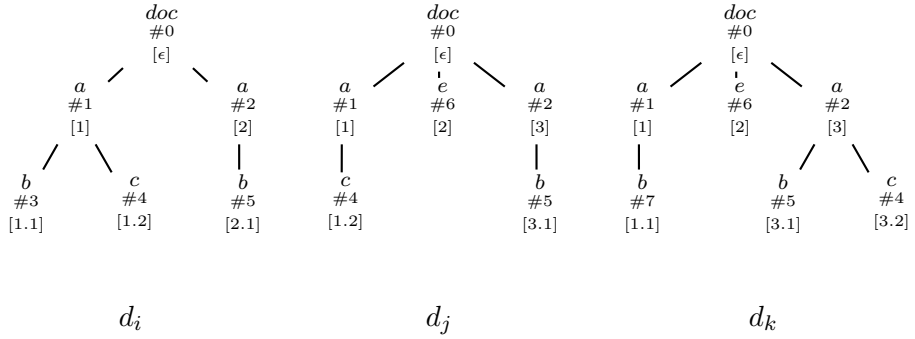


FIGURE 6.1 – Illustration des identifiants explicites.

– il est toujours possible de "suivre" un élement du document au cours de l'évolution de celui-ci à l'aide des identifiants explicites. C'est le cas de l'élément c repéré par l'identifiant explicite $\#4$ qui, dans d_i et d_j est fils du noeud étiqueté a et repéré par l'identifiant $\#1$ alors que dans d_k il est fils du noeud étiqueté a et repéré par l'identifiant $\#2$.

– les identifiants explicites des noeuds qui sont supprimés dans un document d_t ne peuvent pas "réapparaître" dans un document $d_{t'}$ où $t' > t$ (c'est une hypothèse que nous allons poser). Remarquez que le noeud étiqueté b et repéré par $\#3$ est supprimé entre les instants 0 et 1. Pour autant, son identifiant explicite $\#3$ n'est pas attribué au noeud étiqueté e inséré dans d_j ou au noeud étiqueté b inséré comme fils de du noeud a repéré par $\#1$ dans le document d_k.

■

Donc dans la suite un store σ est étendu par une application γ définie sur $dom(\sigma)$. Etant donné un identifiant $\mathbf{i}\in dom(\sigma)$, on nomme $\gamma(\mathbf{i})$ l'identifiant explicite associé à $\mathbf{i}$. Des conditions seront ultérieurement introduites suivant que ces stores étendus seront exploités dans la définition d'un document temporel abstrait ou dans la définition d'un document concret ie estampillé.

Par extension, un arbre est noté (r_d, σ, γ), une forêt est notée $(\mathrm{J}, \sigma, \gamma)$.

Nous pouvons maintenant introduire le modèle abstrait temporel pour XML.

Définition 24 (Document Temporel Abstrait)**.**
Un document temporel abstrait **d** sur le *domaine temporel* $[0, n]$ est une séquence $d_0, \ldots, d_n$ de documents statiques (appelés *snapshots*) tels que : pour chaque document $d_t{=}(r, \sigma_t, \gamma_t)$, l'application γ_t vérifie :
$\forall \mathbf{i}, \mathbf{i}' \in dom(\sigma_t), \mathbf{i}{\neq}\mathbf{i}'{\Rightarrow}\gamma_t(\mathbf{i}){\neq}\gamma_t(\mathbf{i}')$.

La condition qui porte sur γ_t oblige les identifiants générés par γ_t à se comporter comme des identifiants explicites interdisant à deux noeuds d'un même document d_t de posséder le même identifiant explicite.

Dans la suite, les identifiants du store seront capturés par les positions des noeuds (cf. Chapitre 2 Section 2.2). Cela va permettre dans certaines figures, de ne pas les représenter ou de n'en faire figurer que quelques uns en fonction des besoins. Rappelons que la position de la racine est ϵ et que la position d'un noeud n autre que la racine est obtenue en concaténant à la position de son parent le rang du noeud n parmi ses frères. Dans la suite :

– la notation $d@\mathbf{i}$ introduite dans le chapitre 2 désigne le noeud du document d dont l'identifiant du store est **i**.
– la notation $d\#x$ désigne le noeud **i** dont l'identifiant explicite est x i.e.$\gamma(\mathbf{i}) = x$. Bien sûr, ceci est possible tant que γ reste une bijection.

La Figure 6.1 présente un document abstrait $\mathbf{d}_{ex} = d_i, d_j, d_k$.

Le modèle concret repose l'idée d'associer à chaque élément une estampille temporelle (un intervalle de temps) pour capturer sa période de validité. L'objectif est d'éviter la duplication systématique des noeuds non modifiés pendant plusieurs instants consécutifs.

Définition 25 (Document Estampillé)**.**
Un document XML estampillé $\Delta{=}(r, \sigma, \gamma, \tau)$ est un document XML enrichi d'une application τ: $dom(\sigma){\rightarrow}Int$ qui satisfait les propriétés suivantes :

(1) $\forall \mathbf{i}, \mathbf{j}{\in}dom(\sigma) - \{r\},\ \mathbf{j}{\in}Child(\sigma, \{\mathbf{i}\})$ implique $\tau(\mathbf{j}){\subseteq}\tau(\mathbf{i})$, et

(2) $\forall \mathbf{i}, \mathbf{j}{\in}dom(\sigma),\ \mathbf{i}{\neq}\mathbf{j}$ et $\gamma(\mathbf{i}){=}\gamma(\mathbf{j})$ implique $\tau(\mathbf{i}){\cap}\tau(\mathbf{j}){=}\emptyset$.

La condition (1) est classique et assure la consistance hiérarchique des estampilles : l'estampille d'un noeud doit être incluse dans l'estampille de son parent. L'estampille de la racine est utilisée pour stocker le domaine temporel du document estampillé : elle permet d'expliciter la valeur de Now.

Comme ils sont destinés à encoder des documents abstraits, les documents estampillés doivent stocker les identifiants explicites utilisés pour suivre l'évolution des éléments. La condition (2) permet de forcer qu'à chaque instant t, un identifiant explicite x soit utilisé par au plus un noeud du document (statique) d_t. Pour ce faire, il suffit d'assurer que deux noeuds **i** et **j** d'un document estampillé qui correspondent à l'évolution d'un seul et même élément (celui dont l'identifiant explicite est $\gamma(\mathbf{i}) = \gamma(\mathbf{j})$), ne peuvent être estampillés par des intervalles qui se chevauchent.

Exemple 71. Considérons le document estampillé Δ_{ex} présenté dans la Figure 6.2. Dans ce document, l'estampille de la racine est indiquée au-dessus de son étiquette. Pour les autres noeuds, l'estampille est représentée sur l'arc liant le noeud à son parent. Pour la clarté de la présentation, Now est désigné par N au niveau des arbres.

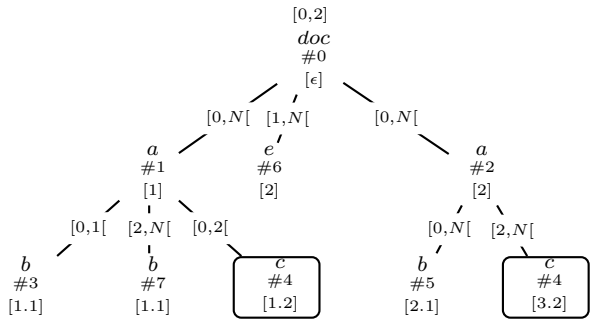


FIGURE 6.2 – Exemple d'un document estampillé Δ_{ex}.

Dans le document estampillé Δ, les deux noeuds qui sont encadrés et qui possèdent le même identifiant explicite $\#4$ portent des estampilles disjointes. Le premier noeud possède l'estampille

$[0, 2[$ indiquant qu'il est valide jusqu'à l'instant 2. Le deuxième noeud possède l'estampille $[2, N[$ indiquant que sa validité commence à l'instant 2. Intuitivement, ces deux noeuds permettent de coder les différents positions d'un seul et même élément, celui identifié par $\#4$. Pendant les instants 0 et 1, cet élément est fils du noeud étiqueté a et ayant comme identifiant explicite $\#1$. A partir de l'instant 2, cet élément est fils du noeud étiqueté a et repéré par l'identifiant explicite $\#2$.

■

Notation
Notons que pour les documents estampillés, γ n'est plus une bijection et donc qu'un identifiant explicite x peut faire référence à plusieurs noeuds d'un document XML estampillé. Dans la suite, étant donné un identifiant explicite x, on définit par $\Delta\#x = \{\mathbf{i} \mid \gamma(\mathbf{i}) = x\}$ l'ensemble des noeuds du document estampillé Δ ayant x pour identifiant explicite.

Une forêt estampillée est notée $\Phi = (\mathrm{J}, \sigma, \gamma, \tau)$ où J est l'id-seq des racines de σ, et où σ, γ et τ sont donnés comme dans la Définition 25.

Les fonctions suivantes permettent la manipulation des estampilles. Considérons une forêt estampillée $\Phi{=}(\mathrm{J}, \sigma, \gamma, \tau)$ et une forêt statique $F{=}(\mathrm{J}', \sigma', \gamma')$. Soit $\mathbf{i}{\in}\mathit{dom}(\sigma)$.

- $\tau^+(\mathbf{i})$ dénote la borne supérieure de l'estampille associée à $\mathbf{i}$.
- $\Phi^{t_1 \leftarrow t_2}$ est la forêt estampillée obtenue en substituant chaque estampille $[t, t_1[$ dans Φ par l'estampille $[t, t_2[$.
- $F^{[t_1, t_2[}$ est la forêt estampillée Φ obtenue en étendant F par une application τ telle que pour tout $\mathbf{i} \in \mathit{dom}(F)$ on a $\tau(\mathbf{i}) = [t_1, t_2[$.

La notion de projection temporelle est définie en étendant la projection classique définie pour les documents statiques dans la Section 2.1 du Chapitre 2.

Définition 26 (Projection Temporelle)**.**
Soit $\Delta{=}(r, \sigma, \gamma, \tau)$ un document estampillé.
La projection temporelle de Δ sur un instant $t{\in}\tau(r)$, notée $Snap(\Delta, t)$, est le document statique d défini par :

$$d{=}(\ r,\ \ \Pi_{T(\Delta,t)}(\sigma),\ \ \gamma_{|T(\Delta,t)}) \quad \text{avec } T(\Delta, t){=}\{\mathbf{i} \mid t{\in}\tau(\mathbf{i})\}$$

Si $t \notin \tau(r)$ alors la projection $Snap(\Delta, t)$ n'est pas définie.

La projection temporelle d'un document estampillé Δ sur un instant t s'exprime comme une projection classique en utilisant l'ensemble $T(\Delta, t)$ qui collecte les identifiants des noeuds dont l'estampille contient t. Notons que le résultat de la projection temporelle est forcément un arbre puisque les documents estampillés sont, par définition, consistants pour les estampilles (cf. Définition 25).

Exemple 72. Considérons le document estampillé Δ_{ex} de la Figure 6.2. Il est aisé de vérifier que $Snap(\Delta_{ex}, 1) = d_j$ où d_j est le document présenté dans la Figure 6.1. ■

Le lien entre la représentation abstraite et la représentation concrète est défini au travers de la notion d'encodage correct et complet présentée ci-dessous.

Définition 27 (Encodage Correct et Complet)**.**
Soit $\mathbf{d}$=$d_0, \ldots, d_n$ un document abstrait de taille small n+1 et Δ=$(r, \sigma, \gamma, \tau)$ un document estampillé.

- Δ est un encodage correct pour $\mathbf{d}$ ssi : $\forall t \in \tau(r)$, $Snap(\Delta, t)$=d_t.
- Δ est un encodage complet pour $\mathbf{d}$ ssi : $\forall t \in [0, n]$, $Snap(\Delta, t)$=d_t.

La notion de correction définie ci-dessus est triviale : elle consiste à s'assurer que chaque snapshot encodé dans le document estampillé Δ est un document de $\mathbf{d}$. Il est clair que cette définition autorise le document estampillé à contenir plus d'informations qu'il n'y en a dans le document abstrait. La notion de complétude assure donc l'inclusion inverse.

Exemple 73. Le document Δ de la Figure 6.2 constitue un encodage correct et complet pour le document abstrait $\mathbf{d}$ présenté dans la Figure 6.1. En effet, on peut facilement vérifier que pour $i \in [0, 2]$ on a $Snap(\Delta, i) = d_i$.

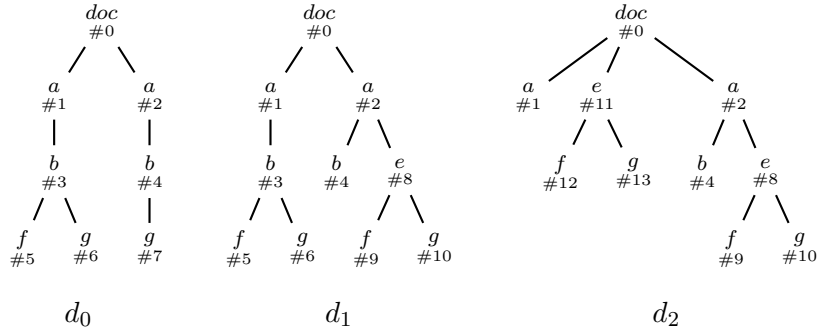
■

Dans la suite, nous ne considérons que des encodages corrects et complets et le terme encodage sera utilisé pour désigner un encodage correct et complet.

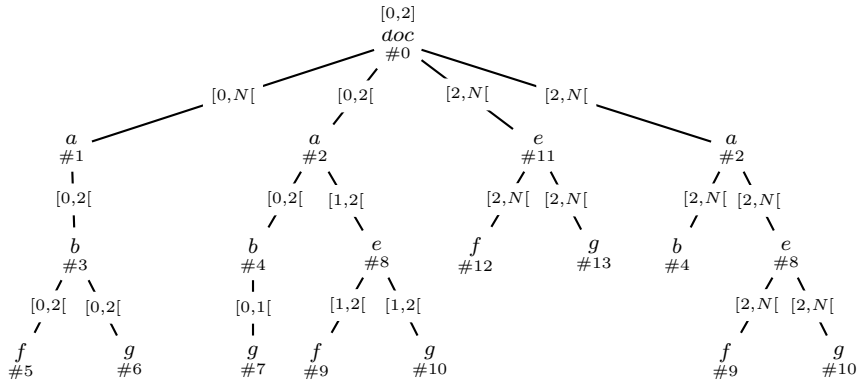
Notation
La Table ci-dessous récapitule les notations de cette section.

DTD	D
Document XML statique Forêt statique	d f ou F
Document temporel abstrait	$\mathbf{d}$
Document estampillé. Forêt estampillée	Δ Φ

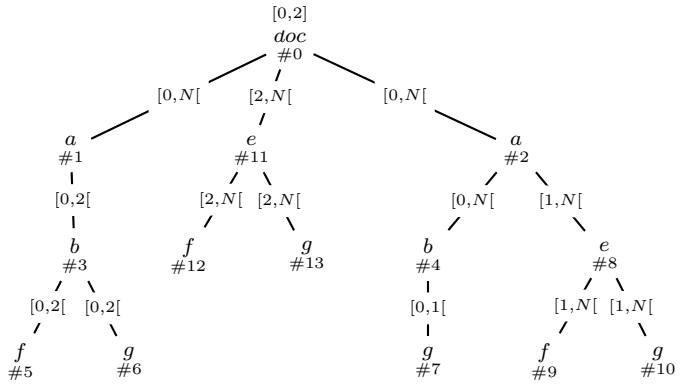
Dans la suite de ce chapitre, les exemples utilisés utilisent le document abstrait $\mathbf{d} = d_0, d_1, d_2$ présenté dans la Figure 6.3-(a) et les documents estampillés Δ et Δ' présentés dans les Figures 6.3-(b) et 6.3-(c) respectivement qui représentent tous les deux des encodages concrets pour $\mathbf{d}$.



(a) Le document abstrait $\mathbf{d}$.



(b) Le document estampillé Δ.



(c) Le document estampillé Δ'.

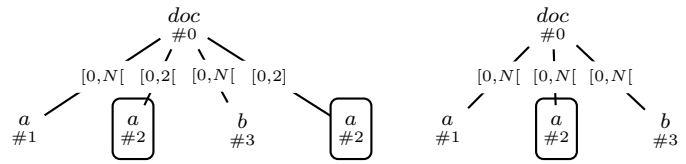
FIGURE 6.3 – Exemple d'un document abstrait $\mathbf{d}$ et de ses encodages Δ et Δ'.

6.1.1 Ordre de compacité

La Définition 27 ne restreint en aucune manière les documents estampillés pouvant être considérés comme des encodages pour un document abstrait : différents encodages peuvent exister pour un même document abstrait. Certains de ces encodages occupent moins d'espace que d'autres : ils stockent moins de noeuds. Nous formalisons cette notion d'efficacité de stockage (appelée par la suite *compacité*) en introduisant un pré-ordre noté $\preceq$ permettant de comparer les documents estampillés.

Intuitivement, comparer deux documents estampillés par rapport à leur compacité repose sur le comptage des noeuds de chacun des documents. Plus précisément, il s'agit de déterminer et de comparer la cardinalité des noeuds ayant une même étiquette et un même identifiant explicite pour chacun des documents comparés.

Exemple 74. Soient les deux documents estampillés Δ_l et Δ'_l ci-dessous.



Δ_l Δ'_l

Il est aisé de voir que $\Delta'_l \preceq \Delta_l$ puisque le noeud étiqueté a et identifié explicitement par $\#2$ est répliqué dans Δ_l : une première "version" de ce noeud est donnée avec l'estampille $[0, 2[$ et se trouve à la position 2, une deuxième version est donnée avec l'estampille $[2, Now[$ et occupe la dernière position des fils de la racine doc. ■

Pour définir $\preceq$, étant donnée une forêt estampillée $\Phi{=}(\mathrm{J}, \sigma, \gamma, \tau)$, nous introduisons la fonction $S_\Phi(l, x)$ définie pour un identifiant explicite x et une étiquette l par :

$$S_\Phi(l, x){=}\{\mathbf{i}{\in}\mathrm{J} \mid \sigma(\mathbf{i}){=}l[\mathrm{K}] \text{ et } \gamma(\mathbf{i}){=}x\}.$$

Intuitivement, cette fonction récolte les noeuds étiquetés par l et explicitement identifiés par x parmi les racines de la forêt Φ.

Définition 28 (Ordre de compacité)**.**
Considérons deux forêts estampillées $\Phi_1{=}(I_1, \sigma_1, \tau_1, \gamma_1)$ et $\Phi_2{=}(I_2, \sigma_2, \tau_2, \gamma_2)$.
Dans la suite, l est une étiquette dans Φ_1 ou Φ_2 et x est un identifiant explicite, i.e. $x{\in}\gamma_1(dom(\Phi_1)){\cup}\gamma_2(dom(\Phi_2))$.
On dit que Φ_1 est *plus compacte* que Φ_2, noté $\Phi_1{\preceq}\Phi_2$, si pour tout l et x, on a :

(1) $|S_{\Phi_1}(l, x)|{\leq}|S_{\Phi_2}(l, x)|$ et

(2) $\Pi_{K_1}(\Phi_1){\preceq}\Pi_{K_2}(\Phi_2)$,
où $K_i{=}\mathit{Desc}(\sigma_i, S_{\Phi_i}(l, x))$ pour $i = 1, 2$.

Cette définition est basée sur la structure arborescente des forêts comparées. La condition (1) permet de comparer le nombre des sous-arbres enracinés aux noeuds partageant la même étiquette l et le même identifiant explicite x. La condition (2) itère la comparaison sur les sous-forêts enracinées en ces noeuds.

Exemple 75. Considérons les documents estampillés Δ et Δ' de la Figure 6.3. Intuitivement, il est facile de se convaincre que Δ' est plus compact que Δ : en effet Δ contient plus de noeuds que Δ'. Ceci va être vérifié en utilisant la définition 28.

Cette définition procède en comparant le nombre de noeuds racine de Δ et Δ', ayant les mêmes étiquettes et les mêmes identifiants explicites. Cette comparaison est triviale puisque les racines sont identiques.

La comparaison se poursuit pour les sous-forêts de Δ et de Δ' que nous notons Φ et Φ' respectivement. Dans ce cas, les identifiants explicites des racines de Φ et Φ' sont : $\#1$, $\#11$ et $\#2$ et les étiquettes des racines de ces forêts sont : a et e. L'application de la Définition 28 consiste à successivement comparer :

- $|S_\Phi(a,\#1)|$ et $|S_{\Phi'}(a,\#1)|$, $|S_\Phi(a,\#11)|$ et $|S_{\Phi'}(a,\#11)|$, $|S_\Phi(a,\#2)|$ et $|S_{\Phi'}(a,\#2)|$
- $|S_\Phi(e,\#1)|$ et $|S_{\Phi'}(e,\#1)|$, $|S_\Phi(e,\#11)|$ et $|S_{\Phi'}(e,\#11)|$, $|S_\Phi(e,\#2)|$ et $|S_{\Phi'}(e,\#2)|$

Pour l'exemple, on vérifie que $|S_{\Phi'}(l,x)| \leq |S_\Phi(l,x)|$ dans tous les cas. Pour poursuivre la présentation de l'exemple, nous notons $\Phi_{l,x}$ (respectivement $\Phi'_{l,x}$) la forêt ayant pour racines les noeuds de $S_\Delta(l,x)$ (respectivement, les noeuds de $S_{\Delta'}(l,x)$). Ces forêts sont présentées dans la Figure 6.4. Nous constatons que

- les forêts $\Phi_{a,1}$ et $\Phi'_{a,1}$ sont identiques.
- de même pour les forêts $\Phi_{e,11}$ et $\Phi'_{e,11}$.

Nous commentons donc la suite du traitement des forêts $\Phi_{a,2}$ et $\Phi'_{a,2}$ qui consiste à comparer séparément :

- les forêts sub-$\Phi_{b,\#4}$ et sub-$\Phi'_{b,\#4}$ de la figure 6.5, d'une part et
- les forêts sub-$\Phi_{e,\#8}$ et sub-$\Phi'_{e,\#8}$

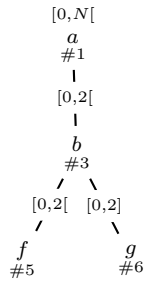
Ces comparaisons permettent de conclure que $\Delta' \preceq \Delta$.

■

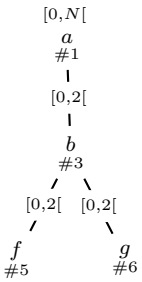
Remarquons qu'il est parfois impossible de comparer des documents estampillés en utilisant la définition 28. L'exemple suivant illustre un cas où une telle comparaison est impossible.

Exemple 76. Les documents estampillés Δ_1 et Δ_2 de la Figure 76 ne sont pas comparables. En effet, les deux documents ont des racines identiques. Maintenant, en examinant les noeuds sous les racines, nous constatons que le nombre de noeuds de niveau 1 (fils de la racine) dans Δ_1 étiquetés par a et explicitement identifiés par $\#1$ est supérieur au nombre de noeuds de niveau 1 dans Δ_2 ayant les mêmes caractéristiques. Donc si Δ_1 et Δ_2 sont comparables alors nous devrions avoir $\Delta_2 \preceq \Delta_1$. Mais en fait, si on poursuit les comparaisons en examinant dans Δ_1 et Δ_2 les fils des noeuds étiquetés par a et explicitement identifiés par $\#2$, alors nous constatons que le nombre de ces noeuds dans Δ_1 étiquetés par d et explicitement identifiés par $\#4$ est inférieur au nombre de noeuds dans Δ_2 ayant les mêmes caractéristiques. Ceci ne permet pas de conclure que $\Delta_2 \preceq \Delta_1$.

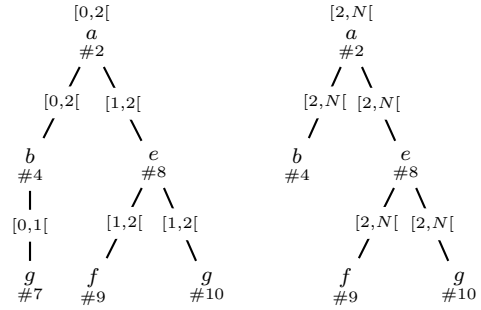
■



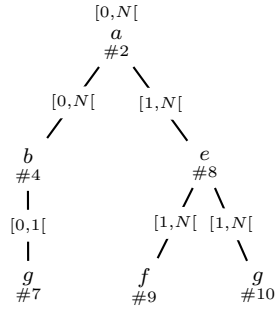
$\Phi_{a,1}$ la sous-forêt de Δ enracinée en $S_\Delta(a,1)$



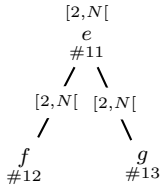
$\Phi'_{a,1}$ la sous-forêt de Δ' enracinée en $S_{\Delta'}(a,1)$



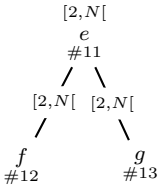
$\Phi_{a,2}$ la sous-forêt de Δ enracinée en $S_\Delta(a,2)$



$\Phi'_{a,2}$ la sous-forêt de Δ' enracinée en $S_{\Delta'}(a,2)$



$\Phi_{e,11}$ la sous-forêt de Δ enracinée en $S_\Delta(e,11)$



$\Phi'_{e,11}$ la sous-forêt de Δ' enracinée en $S_{\Delta'}(e,11)$

FIGURE 6.4 – Comparaison de Δ et Δ'.

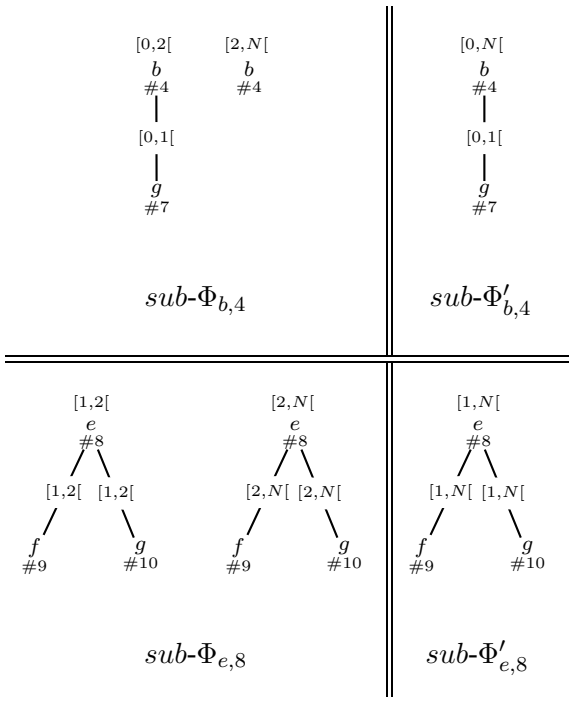


FIGURE 6.5 – Comparaison de $\Phi_{a,2}$ et $\Phi'_{a,2}$.

Dans le cas général, la relation $\preceq$ est un pré-ordre : elle est réflexive, transitive mais elle n'est pas antisymétrique. Montrer la réflexivité et la transitivité est trivial. Le contre-exemple ci-dessous montre que $\preceq$ n'est pas antisymétrique.

Exemple 77. Considérons les documents estampillés Δ_1 et Δ_3 de la Figure 77. On a :

- $\Delta_1 \preceq \Delta_3$,
- $\Delta_3 \preceq \Delta_1$ et
- $\Delta_1 \neq \Delta_3$

■

Notons que dans l'exemple précédent, Δ_1 et Δ_3 n'ont pas le même domaine temporel. Comparer des documents qui diffèrent par leur étiquettes ou leur domaine temporel n'est pas très intéressant dans notre cas. Dans la suite, nous comparons uniquement des paires de documents estampillés qui sont un encodage du même document abstrait.

Propriété 10.
Sous les conditions ci-dessus, $\preceq$ est un ordre partiel.

Pour un document abstrait $\mathbf{d} = d_0, \ldots, d_n$, nous notons $\mathrm{Top}(\mathbf{d})$ l'encodage le "moins compact" de $\mathbf{d}$ i.e. l'encodage tel que $\Delta \preceq \mathrm{Top}(\mathbf{d})$ pour tout encodage Δ de $\mathbf{d}$.

Exemple 78. Soit le document abstrait $\mathbf{d}$ de la Figure 6.3-(a). $\mathrm{Top}(\mathbf{d})$ est présenté dans la Figure 6.8. Intuitivement cet encodage est construit concaténant toutes les sous forêts des documents statiques d_i estampillées chacune de manière adéquate i.e. par $[i, i+1[$. ■

FIGURE 6.6 – Exemple de documents estampillés non comparables.

FIGURE 6.7 – Antisymétrie de $\preceq$: contre exemple.

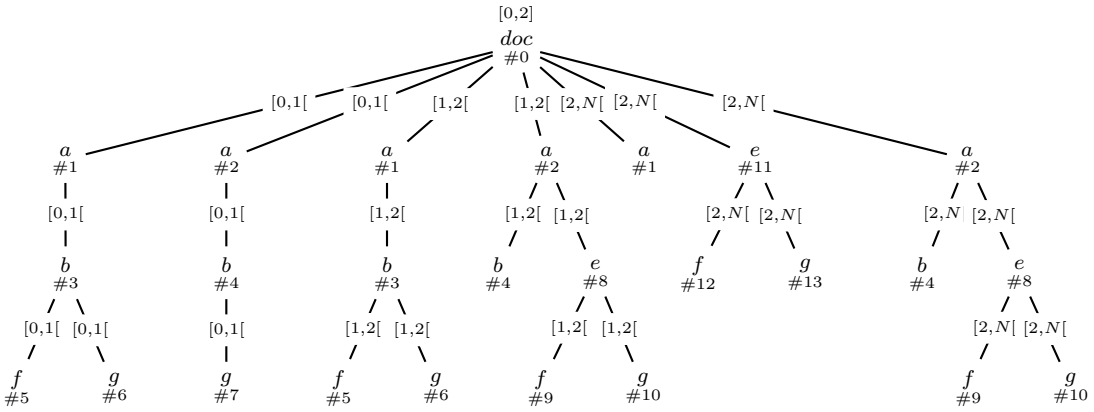


FIGURE 6.8 – L'encodage $\mathrm{Top}(\mathbf{d})$.

6.2 Construction d'encodages compacts

Cette section présente deux méthodes permettant la construction et la maintenance d'un encodage concret pour un document abstrait. L'objectif de ces méthodes est double. Il s'agit d'abord de permettre le traitement de documents très volumineux puisque la taille des documents temporels est susceptible d'être importante et peut constituer un obstacle pour la génération d'encodage concrets. Il s'agit aussi de générer des encodages compacts afin de faciliter leur traitement (requêtes, mises à jour, etc).

6.2.1 Encodage de documents abstraits : cas général

Il existe diverses applications qui nécessitent la création et la maintenance d'archives temporelles. On peut citer à titre d'exemple le cas des entrepôts de données temporels [MACM01, WZ08] ou l'archivage de pages web [SJ08]. L'une des caractéristiques de ces applications est que les données sont produites et modifiées suivant différents procédés. Il peut s'agir du scénario classique qu'on connaît dans les bases de données consistant à utiliser les mises à jour pour modifier ces données. Mais le plus souvent, ces données sont modifiées par d'autres types de traitements qui ne sont pas forcément des mises à jour. On peut citer à titre d'exemple la modification des pages web le plus souvent effectuée par simple édition de leur code HTML. Dans ce type de situation, les seules informations disponibles sont les versions successives des documents. Toute information sur les opérations ayant permis d'obtenir les différentes versions est absente.

La méthode que nous proposons, appelée *It-Comp*, a pour objectif de prendre en compte ce type de situations. Elle permet de construire et de maintenir un encodage pour une séquence de documents sans faire d'hypothèse sur la manière dont les documents de cette séquence ont été générés. Pour permettre de faire face aux limitations d'espace mémoire, la méthode *It-Comp* est conçue pour traiter les documents en streaming. Ce choix s'avère particulièrement intéressant puisqu'il rend possible la prise en compte de certains types de scénarios identifiés dans la littérature par [BBD$^+$02] dans lesquels les données ne sont plus stockées en mémoire secondaire mais traitées directement au vol au moment de leur génération.

6.2.1.1 Scénario de la méthode *It-Comp*

Étant donné un document abstrait $\mathbf{d}$=$d_0, \ldots, d_n$, la construction d'un document estampillé Δ_n encodant $\mathbf{d}$ s'appuie sur une procédure itérative. Le document initial d_0 de la séquence est transformé en un document estampillé Δ_0 en assignant à ses noeuds l'estampille $[0, Now[$, i.e. $\Delta_0 = d_0^{[0,Now[}$. En supposant que Δ_{t-1} est un encodage de $d_0, \ldots, d_{t-1}$, le document d_t est intégré à Δ_{t-1} afin de produire Δ_t. Cette "intégration" est fait par la procédure *Comp* qui parcourt en parallèle et de manière synchronisée les documents Δ_{t-1} et d_t dans le but de fusionner les noeuds de d_t avec ceux de Δ_{t-1}.

Exemple 79. Nous illustrons la méthode pour le document abstrait $\mathbf{d}$ de la Figure 6.3 (page 153).

La construction d'un encodage pour ce document de taille 3 s'effectue en deux étapes :

– la construction du document estampillé Δ_1 par application de *Comp* sur Δ_0 et d_1,

– la construcion du document estampillé Δ présenté dans la Figure 6.3-(c) par application de *Comp* sur Δ_1 et d_2.

Nous supposons que le document Δ_1 est construit et nous nous focalisons sur l'application de *Comp* sur Δ_0 et d_1 (voir Figure 6.9).

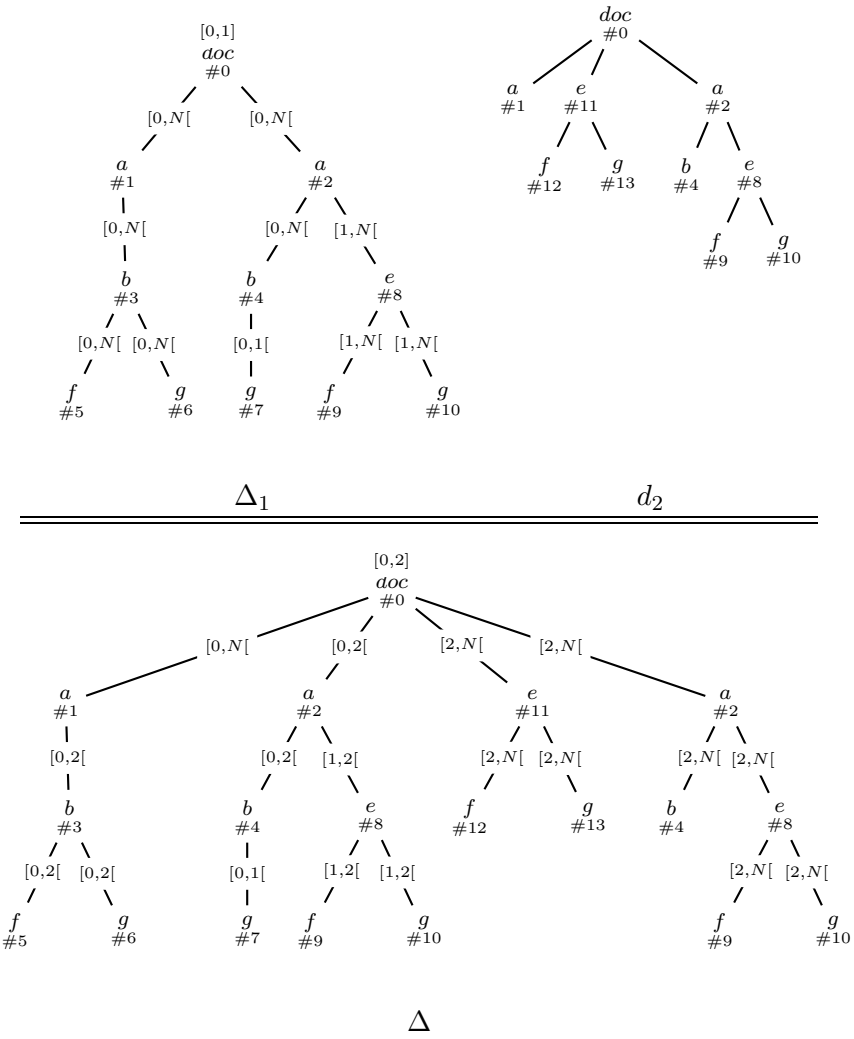


FIGURE 6.9 – Illustration de *Comp*(Δ_1, d_2)

Afin de pouvoir désigner de manière unique les noeuds de chaque document, nous utilisons les identifiants des stores qui sont les positions des noeuds dans les documents respectifs. Ces identifiants ne sont pas notés dans les documents de la Figure 6.9 pour éviter de surcharger ceux-ci.

Le parcours de Δ_1 et d_2 s'effectue suivant l'ordre des documents et consiste à analyser les identifiants **explicites**, les étiquettes et les estampilles de chaque noeud examiné.

– Le traitement des noeuds racines $\Delta_1@\epsilon$ et $d_2@\epsilon$ est simple : la racine $\Delta_1@\epsilon$ est retournée après incrémentation de la borne supérieure de son estampille.

– Le traitement se poursuit par l'examen des noeuds $\Delta_1@1$ et $d_2@1$: le noeud $\Delta_1@1$ est retourné sans aucune modification car Δ_1 n'a subi aucune modification.

– Lorsque le noeud $\Delta_1@1.1$ étiqueté par b et identifié explicitement pas $\#3$ est examiné, le sous-arbre de d_2 enraciné en $d_2@1$ a déjà été entièrement parcouru. Cela signifie que le sous-arbre enraciné au noeud $\Delta_1@1.1$ a été supprimé ou encore déplacé. Le sous-arbre $\Delta_1@1.1$ est donc retourné en "fermant" les estampilles de tous ses noeuds, i.e en substituant la valeur 2 à Now.

– Les noeuds suivants examinés sont $\Delta_1@2$ et le noeud $d_2@2$. Les identifiants explicites et les étiquettes de ces noeuds sont distincts signifiant que le sous-arbre de Δ_1 enraciné en $\Delta_1@2$ a été supprimé ou encore déplacé. Il est donc retourné en fermant les estampilles de ses noeuds comme précédemment. Le sous-arbre $d_2@2$ est quant à lui retourné après avoir assigné à ces noeuds l'estampille $[2, Now[$.

– A ce stade du traitement, le document Δ_1 a été entièrement parcouru alors que le parcours de d_2 en est au noeud $d_2@3$. Le sous-arbre enraciné en $d_2@3$ retourné avec l'estampille $[2, Now[$.

■

Dans l'exemple précédant, nous pouvons remarquer que l'encodage produit par *Comp* n'est pas le meilleur possible en terme de compacité. En fait, le document estampillé Δ' de la Figure 6.3-(c) page 153 est un encodage de $\mathbf{d}$ plus compact que Δ. Dans Δ on remarque qu'il existe deux sous-arbres enracinés en des noeuds ayant pour identifiant explicite $\#2$ alors que dans Δ' il n'y a qu'un seul sous-arbre ayant cette caractéristique. La génération par *Comp* de deux sous-arbres enracinés en des noeuds marqués par $\#2$ s'explique de par :

(i) les documents Δ_1 et d_2 sont parcourus en streaming empêchant toute possibilité pour *Comp* d'explorer les documents pour déterminer si $\Delta_1@2$ a simplement changé de position entre les instants 1 et 2, et n'a pas été supprimé, et

(ii) l'ordre des fils d'un noeud doit être pris en compte dans le document estampillé produit par *Comp* afin produire des encodages corrects.

Les points (i) et (ii) obligent *Comp* à prendre une décision localement sur la base des noeuds visités au moment du parcours des deux documents. Pour notre exemple, les décisions qui s'offrent sont :

(a) retourner $\Delta_1@2$ puis $d_2@3$ ou
(b) (inversement) retourner $d_2@3$ puis $\Delta_1@2$ ou
(c) ignorer $\Delta_1@2$ (dans ce cas le parcours de Δ_1 est simplement stoppé) et retourner $d_2@3$, en espérant retrouver le noeud correspondant à $\Delta_1@2$ plus loin dans d_2.

La décision choisie par *Comp* est (a). Elle entraîne la réplication de $\Delta_1@2$ (et de ses descendants). Les autres décisions ne peuvent pas être considérées car, dans le cas général, elles peuvent mener à un résultat incorrect.

La spécification formelle de *Comp* est présentée dans la Figure 6.10. En supposant que Δ_{t-1} est un encodage de $d_0, \ldots, d_{t-1}$, et que d_t est le document à intégrer, les paramètres en entrée de la fonction *Comp* sont :

– une forêt estampillée F_s correspondant à une sous-forêt de Δ_{t-1},
– une forêt F correspondant à une sous-forêt de d_t, et
– l'instant t.

L'explication ligne par ligne est donnée ci-dessous.

Ligne 1. Cette ligne correspond aux cas terminaux :

Comp$(F_s, F, t) =$		
1	$F^{[t,Now]}$	si *roots*$(F_s) = \emptyset$
1bis	$F_s^{Now \leftarrow t}$	si *roots*$(F) = \emptyset$
		sinon supposer F_s=$\Delta \cdot \Phi$ et F=$d \cdot f$
2	Δ · *Comp*(Φ, F, t)	si $\tau^+_{\Delta_{t-1}}(r_\Delta) \neq Now$
		sinon supposer $\tau^+_{\Delta_{t-1}}(r_\Delta)$=$Now$
3	*TComp*(Δ, d, t) · *Comp*(Φ, f, t)	si *lab*(r_Δ)=*lab*(r_d) et $\gamma_{\Delta_{t-1}}(r_\Delta)$=$\gamma_{d_t}(r_d)$
4	Δ · *Comp*(Φ, f, t)	si $\sigma_{\Delta_{t-1}}(r_\Delta)$=$\sigma_{d_t}(r_{d_t})$=$text[st]$
5	$\Delta^{Now \leftarrow t}$ · $d^{[t,Now]}$ · *Comp*(Φ, f, t)	$sinon$

FIGURE 6.10 – Definition de *Comp*

– si la forêt F_s est vide (ligne 1), les arbres de F sont considérés comme "nouveaux" et insérés avec l'estampille $[t, Now[$;
– si la forêt F est vide (ligne 1bis), les arbres de F_s ont été supprimés ou déplacés, par conséquent, ils sont retournés en fermant leur estampille.

Ligne 2. Cette ligne traite le cas où le noeud examiné r_Δ n'est pas valide à l'instant courant. Dans ce cas, le sous-arbre Δ est simplement retourné.

Ligne 3. Cette ligne traite le cas où les noeuds examinés r_Δ et r_d correspondent au même élément qui n'a pas été modifié. Dans ce cas, le document construit par la fonction *TComp*(Δ, d, t) est retourné. Cette fonction construit un arbre ayant comme racine r_Δ et dont la sous-forêt est construite par l'application récursive de *Comp*.
Formellement,

$$\textit{TComp}(\Delta, d, t) = tree(r_\Delta, lab(r_\Delta), F_r)$$
$$\text{où } F_r = \textit{Comp}(\textit{subfor}(\Delta), \textit{subfor}(d), t)$$

Ligne 4. Cette ligne traite le cas où les deux noeuds parcourus sont le même texte. Le noeud texte de Δ est retourné (ce qui revient à retourner Δ).

Ligne 5. Cette ligne capture les différents cas de changement : r_Δ a été supprimé entre les instant $t-1$ et t ou il a été déplacé modifiant ainsi l'ordre entre les noeuds. Le traitement consiste à retourner Δ en fermant les estampilles de ses noeuds, suivi de d estampillé par $[t, Now[$.

Nous pouvons maintenant formellement définir la méthode qui construit l'encodage d'un document temporel abstrait $\mathbf{d}$.

Définition 29 (*It-Comp*$(\mathbf{d})$).
Soit $\mathbf{d}$=$d_0, \ldots, d_n$ un document abstrait. L'encodage *It-Comp*$(\mathbf{d})$ de $\mathbf{d}$ est le document Δ_n défini par :

– Δ_0=$d_0^{[0,Now[}$, et
– Δ_t=*Comp*$\Delta_{t-1} d_t$, pour $t = 1..n$

Le résultat suivant énonce que la correction de la méthode d'encodage *It-Comp*.

Propriété 11. *Soit* $\mathbf{d}$ *un document abstrait.* *It-Comp*$(\mathbf{d})$ *est un encodage concret pour* $\mathbf{d}$*, et* *It-Comp*$(\mathbf{d}) \preceq \mathrm{Top}(\mathbf{d})$.

6.2.2 Encodage d'un historique

Dans la section précédente, nous avons présenté une technique qui construit un encodage d'un document abstrait sans faire d'hypothèse sur ce document. Nous avons vu que, dans certaines situations, cette méthode est limitée lorsqu'il s'agit de générer des documents estampillés compacts. La raison de cette limitation tient essentiellement au fait qu'aucune information sur l'évolution des documents n'est utilisée et que cette méthode procède en streaming dans le souci de manipuler des documents volumineux.

Dans cette section, nous considérons une classe de documents temporels abstraits appelés historiques. Un document abstrait $\mathbf{d} = d_0, \ldots, d_n$ est un historique lorsque chaque document d_t est obtenu par application d'une mise à jour u_t sur le document d_{t-1}. L'historique $\mathbf{d}$ est spécifié, de manière équivalente, par un document initial d_0 et une séquence de mises à jour $u_1, \ldots, u_n$. Le langage de mises à jour considéré ici est XQuery Update [XUP] présenté dans le Chapitre 2. Nous supposons que chaque document d_i est valide relativement à une DTD D, ne permettant pas ici l'évolution des schémas.

Générer un document estampillé qui encode un historique $\mathbf{d}$ spécifié par d_0 et $u_1, \ldots, u_n$ est réalisé au travers d'un processus appelé *It-Update* qui s'inspire fortement de la technique d'évaluation des mises à jour par projection présentée dans la premièr partie de ce mémoire.
Ce processus itératif commence, comme précédemment, par transformer le document initial d_0 en un document estampillé $\Delta_0 = d^{[0,Now[}$. En supposant que Δ_{t-1} est un document estampillé qui encode l'historique spécifié par d_0 et $u_1, \ldots, u_{t-1}$, la mise à jour u_t est propagée dans Δ_{t-1} pour produire le document estampillé Δ_t en utilisant la fonction $Update$ i.e $\Delta_t = Update(\Delta_{t-1}, u_t)$.

Scénario pour $Update(\Delta_{t-1}, u_t)$**.** La technique de projection développée dans la première partie de ce manuscrit pour les documents statiques est exploitée par $Update(\Delta_{t-1}, u_t)$ pour d'une part optimiser l'espace nécessaire au traitement des documents estampillés et d'autre part être compatible avec n'importe quel moteur mémoire-centrale. La propagation d'une mise à jour u_t sur un document estampillé Δ_{t-1} repose sur le scénario présenté dans le chapitre 4. Le scénario correspondant à $Update(\Delta_{t-1}, u_t)$ est présenté dans la Figure 6.11. Il consiste en quatre étapes :

(1) le tri-projecteur π est inféré pour la mise à jour u_t et la DTD D suivant la technique d'inférence développée dans le chapitre 4.

(2) le document estampillé Δ_{t-1} est projeté, lors de son chargement en mémoire, suivant le projecteur temporel π^{Now} qui combine à la fois :

- la projection temporelle à l'instant de temps $t-1$ (ou Now), et
- la projection en utilisant le tri-projecteur π,

$$\pi^{\text{Now}}(\Delta_{t-1}) \stackrel{def}{=} \pi(Snap(\Delta_{t-1}, Now)).$$

Le document ainsi obtenu est équivalent à la projection du document $d_{t-1} = u_{t-1}(...u_0(d_0))$ (le document obtenu en appliquant la séquence des mises à jour $u_0, \ldots, u_{t-1}$ à partir du document initial d_0), i.e $\pi^{\text{Now}}(\Delta_{t-1}) \simeq \pi(d_{t-1})$.

(3) la mise à jour u_t est alors évaluée sur le document projeté $\pi^{\text{Now}}(\Delta_{t-1})$ et produit une mise à jour partielle $u_t(\pi^{\text{Now}}(\Delta_{t-1}))$. Rappelons que cette étape ne nécessite aucune réécriture de u_t et peut être effectuée par n'importe quel moteur de mise à jour.

(4) la mise à jour de la projection $u_t(\pi^{\text{Now}}(\Delta_{t-1}))$ est intégrée au document estampillé Δ_{t-1}. Cette étape diffère de l'étape de fusion de la technique de projection pour le cas statique (cf. Chapitre 4) pour deux points.

Premièrement, elle doit propager les mises à jour appliquées sur $\pi^{\text{Now}}(\Delta_{t-1})$ de manière à conserver les "anciennes" versions des noeuds de Δ_{t-1}.

Deuxièmement, elle doit maintenir les estampilles de Δ_{t-1}.

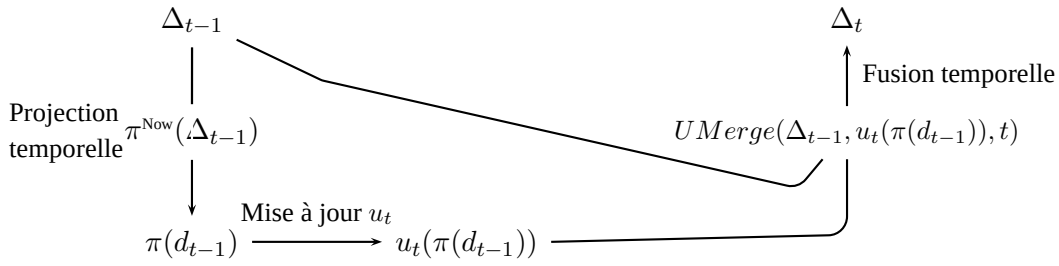


FIGURE 6.11 – Le scénario de $Update(\Delta_{t-1}, u_t)$

La méthode d'encodage à partir d'un historique ne nécessite plus l'utilisation des identifiants explicites. Par contre, puisque cette méthode utilise la projection pour les mises à jour, les identifiants du store doivent correspondre aux positions des noeuds dans les documents.

La propagation des effets d'une mise à jour u_t sur un document estampillé Δ_{t-1} est assurée par une fonction appelée $TreeUMerge$. Cette fonction parcourt le document estampillé Δ_{t-1} et le document partiellement mis àjour $u_t(\pi^{\text{Now}}(\Delta_{t-1}))$ en parallèle et de manière synchronisée. A chaque étape, un noeud de Δ_t et un noeud de $u_t(\pi^{\text{Now}}(\Delta_{t-1}))$ sont examinés afin de décider lequel doit être retourné et quelle doit être son estampille.
Durant ce parcours, les versions "anciennes" des sous-abres de Δ_{t-1} estampillés avec $[k_1, k_2[$ où $k_2 \neq Now$ sont retournées directement car ces sous-arbres ont été élagués par la projection temporelle et n'ont pas été modifiés par la mise à jour.
Les noeuds de Δ_{t-1} estampillés avec $[k_1, Now[$ et qui correspondent de ce fait aux noeuds de $Snap(\Delta_{t-1}, Now)$ nécessitent un examen particulier puisque certains de ces noeuds ont pu être élagués et d'autres ont pu être projetés et modifiés. La vérification qu'un noeud n de $u_t(\pi^{\text{Now}}(\Delta_{t-1}))$ correspond à un noeud m de Δ_{t-1} s'effectue en vérifiant les positions de ces deux noeuds sur la base des hypothèses suivantes :

- les positions des noeuds de Δ_{t-1} qui sont projetés sont stockées (comme pour l'évaluation de mise à jour simple),
- les nouveaux noeuds insérés par u_t ne possèdent pas d'identifiants explicites (et n'ont donc pas de position). Ceci est dû au fait que les mises à jour ne sont pas réécrites.

Formellement, on a :

$$Update(\Delta_{t-1}, u_t) = TreeUMerge(\Delta_{t-1}, u_t(\pi^{\text{Now}}(\Delta_{t-1})), t)$$

Illustration de $TreeUMerge$**.** $TreeUMerge$ est illustrée en utilisant l'exemple de la Figure 6.12. Cette Figure présente une DTD D, une mise à jour u_2 et le projecteur π inféré à partir de u_2 pour D. Elle présente également un document estampillé Δ_1 qui constitue un encodage de d_0, d_1 présentés dans la Figure 6.3 (page 153). D'autre part, on peut vérifier avec la Figure 6.3 (page 153) que d_2 est obtenu en appliquant à d_1 la mise à jour u_2.

La projection temporelle du document estampillé Δ_1 est présentée dans la Figure **??**-(5). Cette projection a élagué le noeud $\Delta_1@2.1.1$ étiqueté g puisqu'il n'est pas valide à l'instant 2. La projection basée sur les types permet quant à elle d'élaguer le noeud $\Delta_1@1.1.1$ étiqueté parce que celle-ci n'appartient pas au tri-projecteur π. Notons que le sous-arbre enraciné au noeud $\Delta_1@2.2$ et étiqueté e est projeté parce que e est dans $\pi_{\mathbf{eb}}$.

La mise à jour de la projection $u_2(\pi^{\text{Now}}(\Delta_1))$, notée δ_2, est présentée dans la Figure **??**-(6). Elle a supprimé le sous-arbre $\pi^{\text{Now}}(\Delta_1)@1.1$ et a inséré un nouveau sous-arbre obtenu en copiant le sous-arbre enraciné en $\pi^{\text{Now}}(\Delta_1)@2.2$, Rappelons que les identifiants du store des noeuds nouvellement insérés ne véhiculent aucune information sur la position. Ils sont notés par n_1, n_2, et n_3 dans notre exemple.

Le résultat de l'application de $TreeUMerge$ sur le document estampillé Δ_1 et la mise à jour partielle δ_2 est présenté dans la Figure 6.12-(7). Le parcours de ces deux documents commence à leur racine qui sont identiques. Le traitement consiste alors simplement à retourner la racine de Δ_1 en mettant à jour sont estampille qui devient $[0, 2]$. On rappelle ici que l'estampille de la racine sert à enregistrer la valeur courante de Now, donc ici 2.

Le parcours se poursuit dans Δ_1 avec le noeud $\Delta_1@1$ qui a été projeté par π^{Now} car son étiquette a appartient au tri-projecteur ($a \in \pi_{\mathbf{no}}$) et son estampille est $[0, Now[$, signifiant qu'il est valide à l'instant 2, et dans δ_2 avec le noeud $\delta_2@1$ qui possède la même étiquette et la même position que $\Delta_1@1$. Ceci permet de déduire que $\Delta_1@1$ n'a pas été modifié par u_2. Il est donc retourné sans aucune modification.

Le parcours se poursuit dans Δ_1 avec le noeud $\Delta_1@1.1$ qui a été projeté par π^{Now} parce que son étiquette b appartient à $\pi_{\mathbf{no}}$ et que son estampille est $[0, Now[$. Dans δ_2, le sous-arbre enraciné en $d_2@1$ est vide signifiant que $\Delta_1@1.1$ a été supprimé par u_2. Ce noeud et ses descendants sont alors retournés en fermant leurs estampilles.
Les noeuds qui sont ensuite examinés sont $\Delta_1@2$ pour Δ_1 et $\delta_2@n_1$ pour δ_2. Le noeud $\delta_2@n_1$ possède un nouvel identifiant n_1 signifiant qu'il vient d'être inséré par u_2. Il est donc retourné avec tous ses descendants en leur assignant l'estampille $[2, Now[$.

Le parcours de δ_2 se poursuit alors que celui de Δ_1 est maintenu au niveau du noeud $\Delta_1@2$. Le parcours de δ_2 amène à examiner le noeud $\delta_2@2$ dont l'étiquette a et l'identifiant 2 sont ceux du noeud $\Delta_1@2$ signifiant que ces deux noeuds sont identiques, le noeud de $\Delta_1@2$ est donc retourné. Le traitement se poursuit jusqu'à ce que les deux documents soient entièrement parcourus.

doc→(*a*|*e*)∗
a→*b**,*c**,*e* ?
c→*String*|*f*
b,*e*→(*f*|*g*)∗

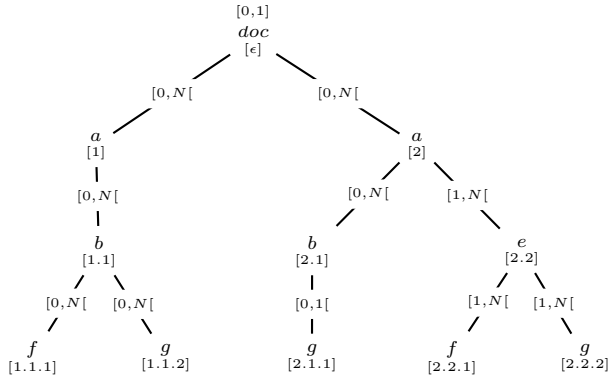
(1) La DTD D.

π_{no}=$\{a, b, g\}$
π_{olb}=$\{doc\}$
π_{eb}=$\{e\}$.

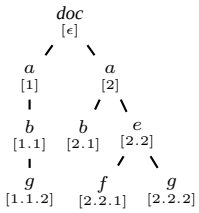
(2) Le tri-projecteur π.

for \$x in `self` :: doc/a
where \$x/b/g
return (**insert** `self`::doc/a/e **after** \$x ,
delete \$x/b)

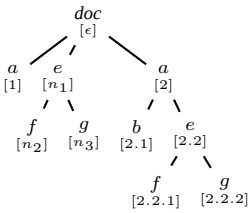
(3) La mise à jour u_2.



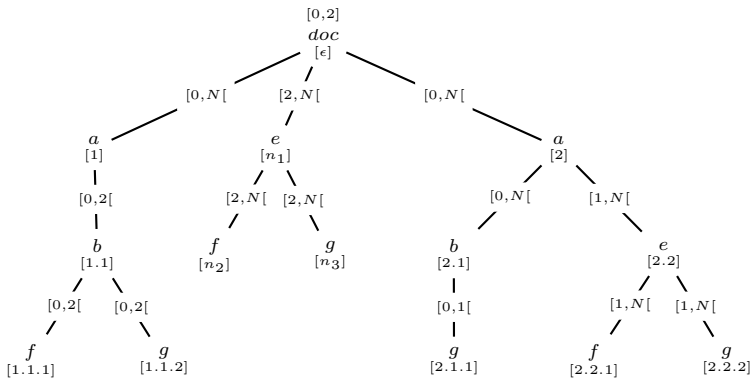
(4) Le document estampillé Δ_1



(5) Le projection temporelle $\pi^{\text{Now}}(\Delta_1)$.



(6) La mise à jour partielle δ_2=$u_2(\pi^{\text{Now}}(\Delta_1))$.



(7) Le document estampillé $UMerge(\Delta_1, u_2, 2)$

FIGURE 6.12 – Illustration de la technique d'encodage basée sur les mises à jour.

$UMerge_{\mathbf{no}}(F_s, \widehat{F_u}, t) =$

1	()	si $roots(F_s)=\emptyset$ **sinon** supposons $F_s=\Delta \cdot \Phi$
2	$\Delta \cdot\ UMerge_{\mathbf{no}}(\Phi, \widehat{F_u}, t)$	si $\sigma_{\Delta_{t-1}}(r_\Delta)=text[s]$ ou $\tau^+_{\Delta_{t-1}}(r_\Delta)\neq Now$, **sinon** supposons $\sigma_{\Delta_{t-1}}(r_\Delta)=a[\mathrm{I}_\Delta]$ et $\tau^+_{\Delta_{t-1}}(r_\Delta)=Now$
3	$\Delta^{Now\leftarrow t} \cdot\ UMerge_{\mathbf{no}}(\Phi, \widehat{F_u}, t)$	si $a\in\pi$ et ($roots(\widehat{F_u})=\emptyset$ ou $\widehat{F_u}=\widehat{d_u}\cdot\widehat{f_u}$ avec $r_{\widehat{d_u}}>r_\Delta$)
4	$TreeUMerge(\Delta, \widehat{d_u}, t) \cdot\ UMerge_{\mathbf{no}}(\Phi, \widehat{f_u}, t)$	si $a\in\pi$ et $\widehat{F_u}=\widehat{d_u}\cdot\widehat{f_u}$ et $r_\Delta=r_{\widehat{d_u}}$
5	$\Delta \cdot\ UMerge_{\mathbf{no}}(\Phi, \widehat{F_u}, t)$	si $a\notin\pi$

(a) La définition de $UMerge_{\mathbf{no}}$

$UMerge_{\alpha}(F_s, \widehat{F_u}, t) =$

$b1$	$\widehat{F_u}^{[t,Now[}$	si $roots(F_s)=\emptyset$ **sinon** supposons $F_s=\Delta\cdot\Phi, \widehat{F_u}=\widehat{d_u}\cdot\widehat{f_u}$
$b2$	$\Delta \cdot UMerge_{\alpha}(\Phi, \widehat{F_u}, t)$	if $\tau^+_{\Delta_{t-1}}(r_\Delta)\neq Now$ **sinon** supposons $\sigma_{\Delta_{t-1}}(r_\Delta)=text[s]$, $\tau^+_{\Delta_{t-1}}(r_\Delta)=Now$
$b3$	$\Delta \cdot UMerge_{\alpha}(\Phi, \widehat{f_u}, t)$	si $\sigma_{\delta_t}(r_{\widehat{d_u}})=text[s]$
$b4$	$\Delta^{Now\leftarrow t} \cdot\ UMerge_{\alpha}(\Phi, \widehat{F_u}, t)$	si ($\sigma_{\delta_t}(r_{\widehat{d_u}})=text[s']$ et $s\neq s'$) ou $\sigma_{\delta_t}(r_{\widehat{d_u}})=b[\widehat{\mathrm{I}}]$ **sinon** supposons $\sigma_{\Delta_{t-1}}(r_\Delta)=a[I_\Delta]$,$\tau^+_{\Delta_{t-1}}(r_\Delta)=Now$
$b5$	$\widehat{d_u}^{[t,Now[} \cdot UMerge_{\alpha}(F_s, \widehat{f_u}, t)$	si $\sigma_{\delta_t}(r_{\widehat{d_u}})=text[s]$ ou ($\sigma_{\delta_t}(r_{\widehat{d_u}})=b[I_{\widehat{d_u}}], new(r_{\widehat{d_u}})=true$)
$b6$	$\Delta^{Now\leftarrow t} \cdot\ UMerge_{\alpha}(\Phi, \widehat{F_u}, t)$	si $a\in\pi$ et ($roots(\widehat{F_u})=\emptyset$ ou $r_{\widehat{d_u}}>r_\Delta$)
$b7$	$TreeUMerge_{\alpha}(\Delta, \widehat{d_u}, t) \cdot\ UMerge_{\alpha}(\Phi, \widehat{f_u}, t)$	si $a\in\pi$ et $r_\Delta=r_{\widehat{d_u}}$
$b8$	$\Delta \cdot\ UMerge_{\alpha}(\Phi, \widehat{f_u}, t)$	si $a\notin\pi$

(b) La définition de $UMerge_{\mathbf{olb}}$ et $UMerge_{\mathbf{eb}}$

FIGURE 6.13 – La spécification de $UMerge$

Remarque sur les positions des noeuds.

Les positions des noeuds dans Δ_1 et dans $TreeUMerge(\Delta_1, u_2(\pi^{\text{Now}}(\Delta_1)), 2)$ sont indiquées dans l'exemple bien qu'en pratique elles ne sont pas stockées pour ces documents. Ces positions sont générées au vol lors de la projection temporelle et lors de l'étape de $TreeUMerge$. De plus, ces positions ne sont pas générées pour tout le document estampillé : seules les positions des noeuds de $Snap(\Delta_{t-1}, Now)$ sont utiles. Comme déjà rappelé, les positions sont stockées dans la projection $\pi^{\text{Now}}(\Delta_{t-1})$. Notons qu'en pratique, il est suffisant de générer les positions permettant de localiser un noeud sur la base de son "rang" relativement à ses voisins. Les positions "complètes" qui stockent également la position des parents des noeuds ne sont pas nécessaires. Cela permet un gain significatif en terme de stockage.

Formalisation de $TreeUMerge$**.** La procédure $TreeUMerge$ prend en entrée deux sous-arbres :

- Δ un sous-arbre du document estampillé Δ_{t-1},
- $\widehat{d_u}$ un sous-arbre de la mise à jour partielle $u_t(\pi^{\text{Now}}(\Delta_{t-1}))$, et
- t qui se réfère à la l'instant de validité de u_t.

$TreeUMerge$ fait les hypothèses suivantes :

- la racine r_Δ de Δ est projetée par π^{Now} i.e $\tau^+(r_\Delta)=Now$ et $lab(r_\Delta)\in\pi$.
- les étiquettes de r_Δ et de $r_{\widehat{d_u}}$ sont identiques.

Posons $lab(r_\Delta) = lab(r_{\widehat{d_u}}) = a$ et notons F_s la forêt de Δ et $\widehat{F_u}$ la forêt de $\widehat{d_u}$.
$TreeUMerge$ construit un sous-arbre dont la racine est r_Δ et dont la forêt est obtenue en fusionnant les forêts F_s et $\widehat{F_u}$. Cette fusion est assurée par l'une des procédures $UMerge_{\mathbf{no}}$, $UMerge_{\mathbf{olb}}$ ou $UMerge_{\mathbf{eb}}$ en fonction de l'appartenance de $lab(r_\Delta)$ à $\pi_{\mathbf{no}}$, à $\pi_{\mathbf{olb}}$ ou à $\pi_{\mathbf{eb}}$.

Formellement,

$$TreeUMerge(\Delta, \widehat{d_u}, t) = tree(r_\Delta, lab(r_\Delta), F_r)$$
$$\text{où } F_r = \; UMerge_\alpha(\textit{subfor}(\Delta), \textit{subfor}(\widehat{d_u}), t) \quad \text{si } lab(r_\Delta)\in\pi_\alpha$$

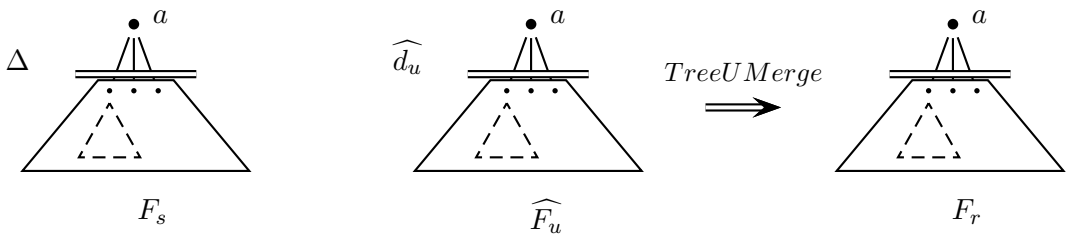


FIGURE 6.14 – Illustration de $TreeUMerge$

La fonction $UMerge_{\mathbf{no}}$ est formalisée dans la Figure 6.13-(a) en page 167. Les fonctions $UMerge_{\mathbf{olb}}$ et $UMerge_{\mathbf{eb}}$ sont spécifiées par la fonction $UMerge_\alpha$ qui est donnée dans la Figure 6.13-(b) en page 167 où α fait référence à *olb* ou à *eb*. Chacune de ces trois fonctions prend en entrée :

- une sous-forêt F_s de Δ_{t-1},
- une sous-forêt $\widehat{F_u}$ de $u_t(\pi^{\text{Now}}(\Delta_{t-1}))$ et

- t qui se réfère à la période de validité de u_t.

Soit n un noeud du document estampillée Δ et m un noeud du document $\widehat{d_u}$. Les trois fonctions se distinguent sur la base des pré-conditions suivantes :

– $UMerge_{\mathbf{no}}$ suppose que (†) le noeud n parent de la forêt estampillée F_s est de catégorie 'node only' ce qui implique que, à cause de la synchronisation (le fait que n correspond à m) :

($\dagger_i$) aucune racine des arbres de $\widehat{F_u}$ n'est un texte, et

($\dagger_{ii}$) les racines des arbres de $\widehat{F_u}$ figurent parmi les racines des arbres de F_s valides à l'instant $t-1$, i.e

$$\mathit{roots}(\widehat{F_u}) \subseteq \mathit{roots}^{Now}(F_s) \text{ où } \mathit{roots}^{Now}(F_s) = \{\mathbf{i} \mid \mathbf{i} \in \mathit{roots}(F_s) \text{ et } \tau^{+}(\mathbf{i}) = Now\}$$

– $UMerge_{\mathbf{olb}}$ suppose que (‡) le noeud n parent de F_s est de catégorie 'one level below' ce qui implique que **tout noeud** de $\mathit{roots}^{Now}(F_s)$ a été projeté.

– $UMerge_{\mathbf{eb}}$ suppose que (‡‡) le noeud n parent de F_s est de catégorie 'everything below' impliquant que **chaque sous-abre** dont la racine appartient à $\mathit{roots}^{Now}(F_s)$ a été projeté.

$\boxed{UMerge_{\mathbf{no}}}$

Cette fonction procède de la manière suivante :

Ligne 1. Cette ligne capture le cas terminal : la forêt F_s est vide. Dans ce cas, la forêt $\widehat{F_u}$ est aussi vide d'après le point ($\dagger_{ii}$).

Ligne 2. Cette ligne capture le cas où la racine du sous-arbre estampillé Δ de F_s qui est examiné est :

(i) soit un noeud qui a été élagué par la projection temporelle car il n'est pas valide à l'instant courant, i.e $\tau^{+}_{\Delta_{t-1}}(r_\Delta) \neq Now$,

(ii) soit un texte qui n'a pas été projeté à cause de l'hypothèse ($\dagger_i$).

Dans ces deux cas, l'arbre estampillé Δ est retourné.

Ligne 3. Cette ligne traite du cas où l'arbre Δ de F_s a été supprimé par la mise à jour u_t. Ceci est identifié par le fait que :

– l'étiquette a de la racine r_Δ appartient au tri-projecteur π et donc que r_Δ a été projeté,

– la racine r_Δ de Δ ne figure pas dans $\widehat{F_u}$ soit parce que $\widehat{F_u}$ est vide soit à cause de la comparaison $r_{\widehat{d_u}} > r_\Delta$ qui indique que l'arbre $\widehat{d_u}$ précède l'arbre Δ dans la forêt F_s.

Par conséquent, r_Δ est retourné avec ses descendants en fermant leurs estampilles.
Ce cas est illustré avec l'exemple suivant.

Exemple 80. Considérons la mise à jour u_1, la DTD D_1 et le document estampillé Δ_{t-1} de la Figure 6.15. La mise à jour u_1 est intégrée au document estampillé Δ_{t-1} et produit le document estampillé Δ_t. Ce processus se déroule comme suit :

- Le tri-projecteur π_1 présenté dans la Figure 6.15-(3) est inféré pour u_1 et D_1. Le composant $\pi_{\mathbf{no}}$ contient les étiquettes des noeuds traversés par u_1 qui sont doc et c et l'étiquette a du noeud supprimé par cette mise à jour. Le composant $\pi_{\mathbf{olb}}$ est vide, et le composant

$\pi_{\mathbf{eb}}$ contient l'étiquette b du noeud dont le sous-élément est extrait par u_1. Le projection temporelle permet d'élaguer le noeud étiqueté e fils du noeud $\Delta_{t-1}@1$ puisque l'estampille $[2, 10[$ du premier indique qu'il n'est pas valide à l'instant t.

- La mise à jour partielle δ_t donnée dans la Figure 6.15-(5) est obtenue en projetant le document estampillé Δ_{t-1} suivant π_1^{Now} puis en mettant à jour le résultat de cette projection par application de u_1, i.e $\delta_t = u_1(\pi_1^{Now}(\Delta_{t-1}))$.
- Le résultat Δ_t (voir Figure 6.15-(6)) est obtenu en appliquant $TreeUMerge$ sur Δ_{t-1} et δ_t. C'est cette étape que nous commentons plus en détails.

Posons $F_s = \textit{subfor}(\Delta_{t-1})$ et $\widehat{F_u} = \textit{subfor}(\delta_t)$. Etant donné que doc l'étiquette de la racine de Δ_{t-1} appartient au composant $\pi_{\mathbf{no}}$, les forêts F_s et $\widehat{F_u}$ sont examinées par $UMerge_{\mathbf{no}}$. Ceci commence par l'examen des noeuds $F_s@1$ et $\widehat{F_u}@2$ encerclés dans la Figure 6.15. Le fait que l'estampille du noeud $F_s@1$ soit $[2, Now[$ et que son étiquette a appartienne au composant $\pi_{\mathbf{no}}$ nous permet de conclure que ce noeud a été projeté. La comparaison de la position de $F_s@1$ qui est 1 avec la position de $\widehat{F_u}@2$ qui est 2, permet de conclure que $F_s@1$ a été supprimé par la mise à jour. C'est pour cette raison que le noeud $F_s@1$ est retourné avec ses descendants en fermant leurs estampilles :
les estampilles initialement de la forme $[k_1, Now[$ deviennent $[k_1, t[$, les estampilles de la forme $[k_1, k_2[$ pour $k_2 \neq Now$ ne changent pas.´
Le parcours se poursuit en progressant dans la forêt F_s seulement, le parcours est figé sur la forêt $\widehat{F_u}$. Le prochain noeud de F_s qui est examiné est $F_s@2$. L'examen de l'étiquette et de l'estampille de ce noeud permet de déduire qu'il est projeté. Comme sa position est égale à celle du noeud $\widehat{F_u}@2$, on déduit qu'il a été conservé par u_1. Dans ce cas, le traitement est assuré par la ligne 4 qui est commentée ci-dessous.

■

Ligne 4. Cette ligne traite du cas où le parcours est synchronisé sur les mêmes sous-arbres. Cela est indiqué par le fait que les positions r_Δ et r_{d_u} sont égales. Rappelons que les mises à jour de renommage qui ne sont pas considérées. Les étiquettes des noeuds identifiés par r_Δ et r_{d_u} sont donc forcément identiques. Le sous-arbre qui est retourné est construit par $TreeUMerge(\Delta, d_u, t)$.

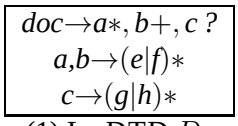
Ligne 5. Cette ligne traite du cas où l'étiquette de r_Δ n'appartient pas au tri-projecteur π signifiant que Δ n'a pas été projeté. Donc, Δ est retourné par $UMerge_{\mathbf{no}}$.

$\boxed{UMerge_\alpha}$
Cette fonction est utilisée pour spécifier les procédures $UMerge_{\mathbf{olb}}$ et $UMerge_{\mathbf{eb}}$. Rappelons que ces deux procédures se distinguent sur la base des pré-conditions (‡) et (‡‡).

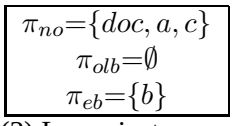
Ligne b1. Cette ligne capture le cas de base : la forêt F_s est vide. Donc ce cas, $\widehat{F_u}$ n'est pas forcément vide puisqu'elle peut contenir de nouveaux noeuds insérés par la mise à jour. Dans ce cas, les sous-arbres de la forêt $\widehat{F_u}$ sont retournés avec l'estampille $[t, Now[$.

Ligne b2. Cette ligne traite du cas où le noeud r_Δ du document estampillé Δ est "ancien". L'arbre Δ est donc retourné sans aucune modification.

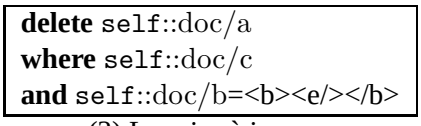
Lignes b3 et b4. Ces deux lignes traitent du cas où l'arbre Δ de F_s qui est examiné est un texte s qui a été projeté. La ligne b3 traite le cas où le *String* s n'a pas été modifié par la mise à jour et doit, par conséquent, être retourné.



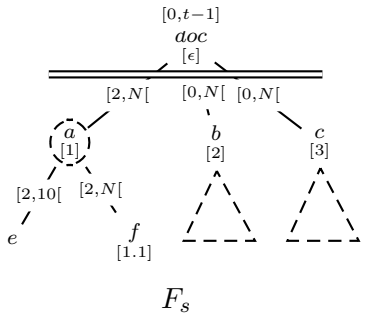
(1) La DTD D_1

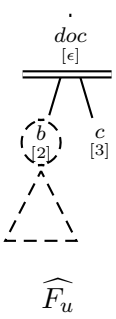


(2) Le projecteur π_1

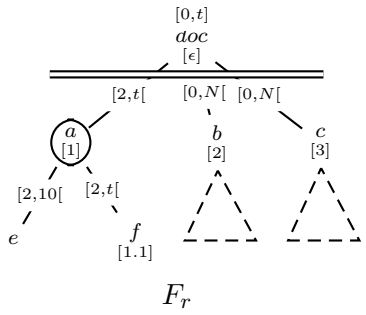


(3) La mise à jour u_1





(4) Le document estampillé Δ_{t-1} (5) La mise à jour partielle δ_t

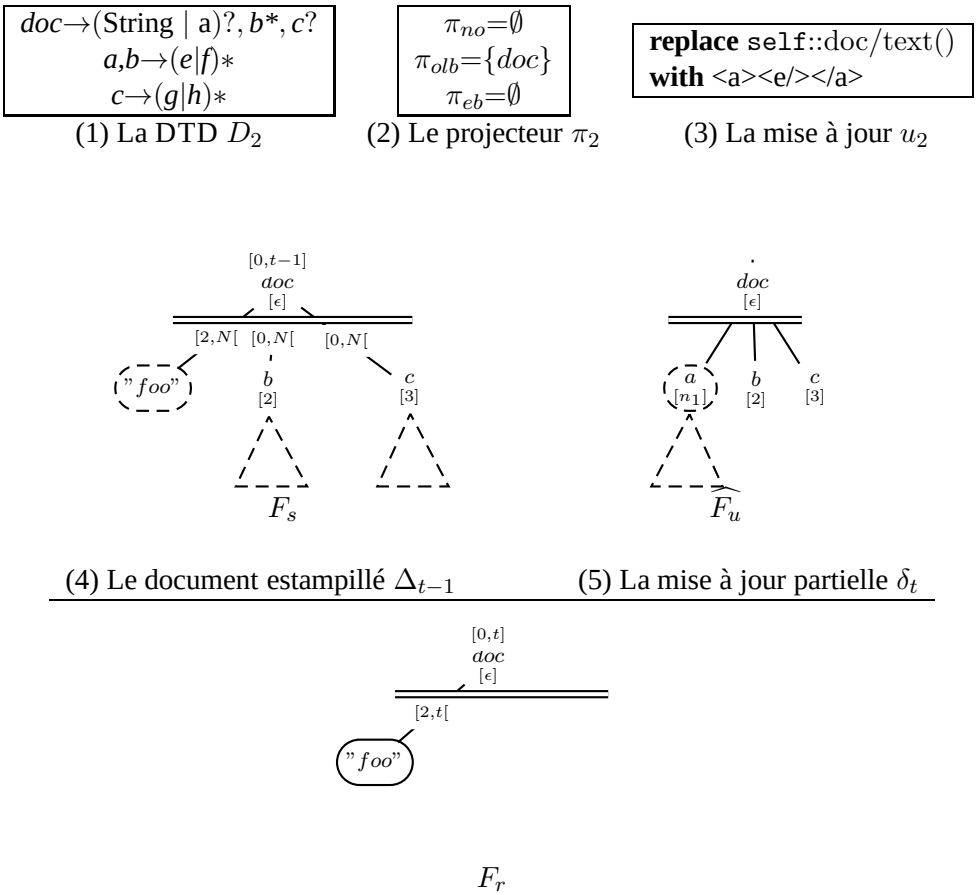


(6) Le document estampillé Δ_t.

FIGURE 6.15 – Illustration de $UMerge_{\mathbf{no}}$: ligne 3

La ligne b4 traite deux cas :
– le cas où s est modifié par la mise à jour, ceci est indiqué par la condition $\sigma_{\delta_t}(r_{d_u})$=$text[s']$ et $s \neq s'$,
– le cas où s est supprimé par la mise à jour, ceci est indiqué par la condition $\sigma_{\delta_t}(r_{\widehat{d_u}})$=$b[\widehat{\mathrm{I}}]$.
Dans les deux cas, le texte s est retourné en fermant son estampille.
Notons que pour ce cas, le parcours de d_t est figé.
La ligne b4 est illustrée par l'exemple suivant.

Exemple 81. Considérons la DTD D_2 obtenue en modifiant la DTD de l'exemple 80 pour pouvoir générer des textes. Considérons la mise à jour u_2, le document estampillé Δ_{t-1} et la mise à jour partielle $\delta_t = u_2(\pi_2^{Now}(\Delta_{t-1}))$ de la Figure 6.16.



(1) La DTD D_2 (2) Le projecteur π_2 (3) La mise à jour u_2

(4) Le document estampillé Δ_{t-1} (5) La mise à jour partielle δ_t

(6) Construction du document estampillé Δ_t.

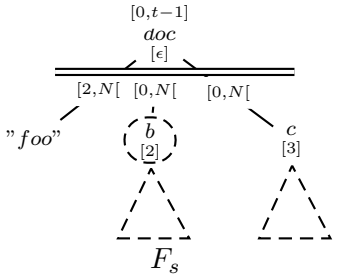
FIGURE 6.16 – Illustration de $UMerge_\alpha$: ligne b4

Comme $doc \in \pi_{\mathbf{olb}}$, les sous-forêts de $\Delta_{t-1}@\epsilon$ et $\delta_t@\epsilon$ sont examinés avec la fonction $UMerge_{\mathbf{olb}}$. Lorsque $UMerge_{\mathbf{olb}}$ examine le noeud texte "foo" de Δ_{t-1} et le noeud $\delta_t@n_1$, la ligne b4 est appliquée et permet de retourner le texte "foo" estampillé avec $[2, t[$. A ce moment précis, la synchronisation est perdue puisque le parcours de δ_t est figé. ■

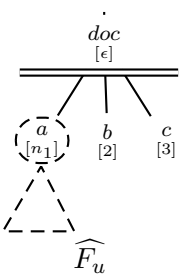
Ligne b5. Cette ligne traite du cas où le noeud $r_{\widehat{d_u}}$ de $\widehat{F_u}$ qui est examiné est soit de type *String* soit un nouvel élément inséré par u_2. Ce dernier cas est identifié en vérifiant si l'identifiant

r_{d_u} est nouveau, i.e si $r_{d_u} \notin \mathit{dom}(\sigma_{\Delta_{t-1}})$. Le sous-arbre enraciné en $r_{\widehat{d_u}}$ est retourné avec l'estampille $[t, Now[$.

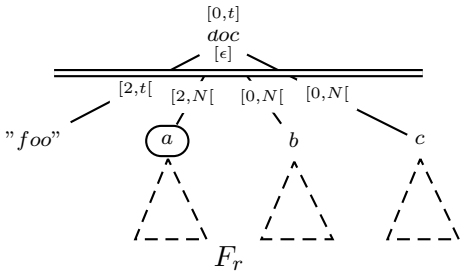
Exemple 82. Dans cet exemple, nous poursuivons le traitement entamé dans l'exemple 81. Le document estampillé Δ_{t-1}, la mise à jour partielle δ_t et le résultat intermédiaire de l'application de $UMerge_{\mathbf{olb}}$ sur ces deux documents sont présentés dans la Figure 6.17.
Après avoir retourné le texte "foo" par l'application de la ligne b4, le parcours avance sur F_s et amène au noeud $F_s@2$. L'examen de $\widehat{F_u}$ reste sur le noeud $\widehat{F_u}@n_1$. Ce noeud possède un nouvel identifiant n_1 indiquant qu'il a été inséré par u_2. Dans ce cas, il est retourné avec ses descendants avec l'estampille $[t, Now[$. Le parcours reprend dans $\widehat{F_u}$ et amène à examiner le noeud $\widehat{F_u}@2$. Désormais, le parcours des forêts F_s et de $\widehat{F_u}$ est synchronisé sur le noeud étiqueté b et dont la position est 2. Le noeud $F_s@2$ a été projeté parce que $doc \in \pi_{\mathbf{olb}}$ et que d'après l'hypothèse $(\ddagger)$ tous les noeuds de $\mathit{roots}^{Now}(F_s)$ sont projetés. L'arbre enraciné en $F_s@2$ n'est, quant à lui pas projeté. Dans ce cas, le traitement consiste à retourner cet arbre et à avancer à la fois dans F_s et dans $\widehat{F_u}$. Ce traitement est assuré par la ligne b8 qui sera expliquée plus loin. Les deux prochains noeuds visités par $UMerge_{\mathbf{olb}}$ sont $F_s@3$ et $\widehat{F_u}@3$. Le noeud $F_s@3$ a été projeté parce que $doc \in \pi_{\mathbf{olb}}$, donc l'arbre enraciné en $F_s@3$ est retourné.



(4) Le document estampillé Δ_{t-1}



(5) La mise à jour partielle δ_t



(6) Construction du document estampillé Δ_t.

FIGURE 6.17 – Illustration de $UMerge_\alpha$: ligne b5

Remarquer que le cas traité par cette ligne correspond à une des deux situations pour laquelle la méthode générale présentée dans la Section 6.2.1 génère un résultat avec réplication.

■

Ligne b6. Cette ligne est similaire à la ligne 3 de $UMerge_{\mathbf{no}}$. Elle traite le cas où r_Δ a été supprimé par la mise à jour. Dans ce cas, r_Δ est retourné avec tous ses descendants en fermant leurs estampilles.

Ligne b7. Cette ligne est analogue à la ligne 4 de $UMerge_{\mathbf{no}}$ qui capture le cas de synchronisation. Toutefois, étant donné que la procédure $UMerge_{\alpha}$ est utilisée pour spécifier à la fois $UMerge_{\mathbf{olb}}$ et $UMerge_{\mathbf{eb}}$, le traitement spécifié par la ligne b7 est, bien entendu, déterminé en fonction de la catégorie α. Nous allons donc présenter le traitement pour chaque catégorie séparément.

Cas où $\alpha = \mathbf{olb}$ Dans ce cas, les forêts F_s et $\widehat{F_u}$ sont examinées par la procédure $UMerge_{\mathbf{olb}}$. Cette procédure suppose que tous les noeuds dans $\mathit{roots}^{Now}(F_s)$ ont été projetés. Les descendants de ces noeuds sont, quant à eux, projetés en fonction de leur étiquette et du tri-projecteur utilisé.
Dans ce cas, la ligne b7 retourne un arbre construit en appliquant la procédure $TreeUMerge$ sur Δ et $\widehat{d_u}$.

Formellement,

$$TreeUMerge_{\mathbf{olb}}(\Delta, \widehat{d_u}, t) = TreeUMerge(\Delta, \widehat{d_u}, t)$$

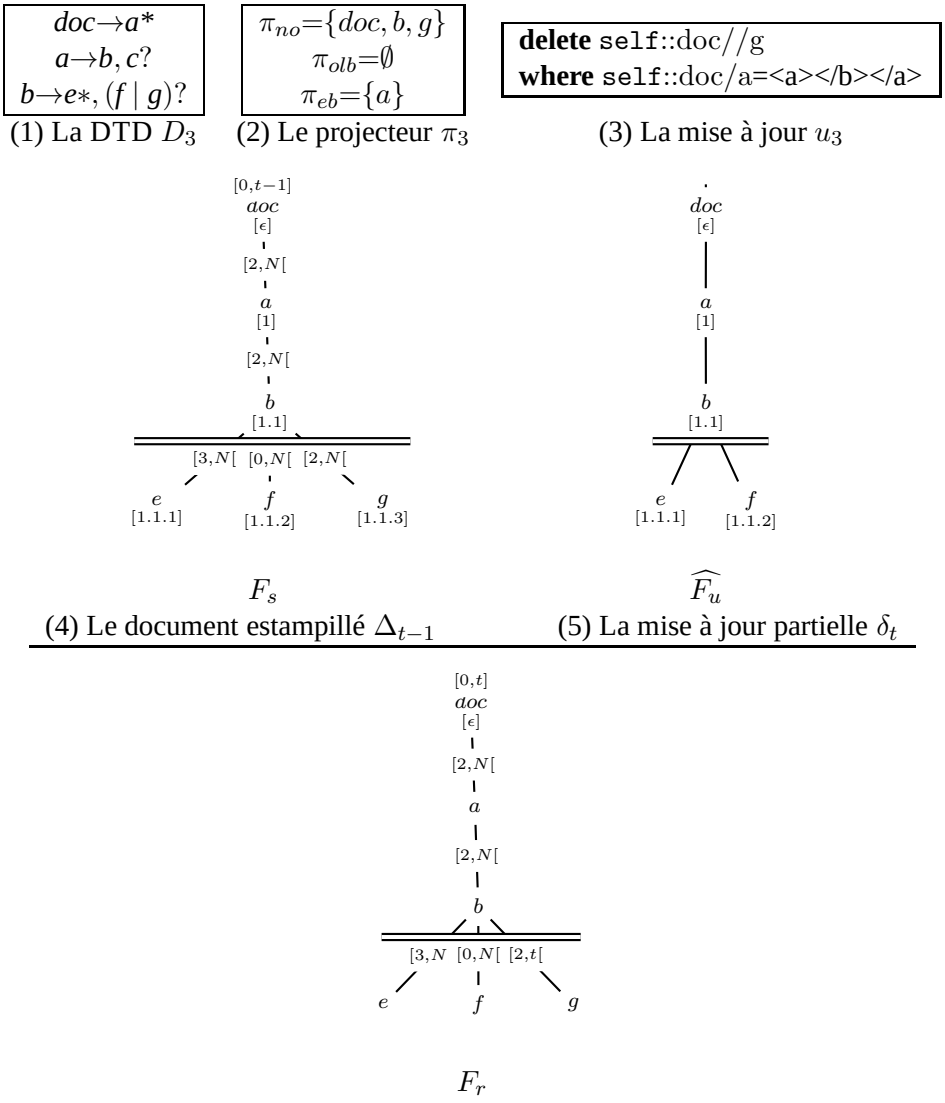
Cas où $\alpha = \mathbf{eb}$ Dans ce cas, les forêts F_s et $\widehat{F_u}$ sont examinées par la procédure $UMerge_{\mathbf{eb}}$. Cette procédure suppose que chaque sous-arbre dont la racine appartient à $\mathit{roots}^{Now}(F_s)$ a été projeté par la projection temporelle, i.e $\widehat{d_u} = u_t(Snap(\Delta, Now))$. Autrement dit, tous les noeuds de Δ valides à l'instant t sont projetés.
Intuitivement, pour un noeud n descendant de $\mathit{roots}^{Now}(F_s)$ dont l'étiquette appartient au tri-projecteur, l'utilisation de $TreeUMerge$ pour le traitement de l'arbre de Δ enraciné en $\Delta@n$ et de l'arbre de $\widehat{d_u}$ enraciné en $\widehat{d_u}@n$ conduirait à sélectionner l'une des procédures $UMerge_{\mathbf{no}}$, $UMerge_{\mathbf{olb}}$ ou $UMerge_{\mathbf{eb}}$ en fonction de la catégorie de $lab(n)$. Or, l'utilisation de $UMerge_{\mathbf{no}}$ ou de $UMerge_{\mathbf{olb}}$ dans le cas où tous les descendants de n ont été projetés ne permet pas d'aboutir puisque ces deux procédures ne font pas la même hypothèse que la procédure $UMerge_{\mathbf{eb}}$ à savoir, tous les noeuds dans $\mathit{roots}^{Now}(F_s)$ ont été projetés.
Dans ce cas, la ligne b7 retourne un arbre construit en appliquant la procédure $TreeUMerge_{\mathbf{eb}}$ sur les arbres sur Δ et $\widehat{d_u}$, où,

$$TreeUMerge_{\mathbf{eb}}(\Delta, \widehat{d_u}, t) = tree(r_{\Delta}, lab(r_{\Delta}), F_r)$$
où $F_r = UMerge_{\mathbf{eb}}(\mathit{subfor}(\Delta), \mathit{subfor}(\widehat{d_u}), t)$.

La procédure $UMerge_{\mathbf{eb}}$ diffère de $TreeUMerge$ qui fait appel à l'une des procédures $UMerge_{\mathbf{no}}$, $UMerge_{\mathbf{olb}}$ ou $UMerge_{\mathbf{eb}}$ en fonction de la catégorie de $lab(r_{\Delta})$ du fait qu'elle force les descendants de F_s et $\widehat{F_u}$ à être examinés par $UMerge_{\mathbf{eb}}$.
L'exemple suivant illustre la ligne b7 pour le cas $\alpha = \mathbf{eb}$.

Exemple 83. Considérons la DTD D_3, la mise à jour u_3 et le document estampillé Δ_{t-1} de la Figure 6.18. Les étapes suives pour intégrer la mise à jour u_3 sur Δ_{t-1} sont les mêmes que pour les exemples précédents :

– le tri-projecteur π_3 donné dans la Figure 6.18-(3) est inféré pour u_3 et D_3. Le composant $\pi_{\mathbf{no}}$ contient les étiquettes doc, b des noeuds traversés par u_3 et l'étiquette g du noeud supprimé par u_3. Le composant $\pi_{\mathbf{olb}}$ est vide. Le composant $\pi_{\mathbf{eb}}$ contient a parce que la mise à jour extrait le sous-arbre dont la racine porte cette étiquette.



(1) La DTD D_3 (2) Le projecteur π_3 (3) La mise à jour u_3

(4) Le document estampillé Δ_{t-1} (5) La mise à jour partielle δ_t

(6) Le document estampillé Δ_t

FIGURE 6.18 – Illustration de $UMerge_\alpha$: ligne b7

– le document estampillé Δ_{t-1} est projeté puis mis à jour, le résultat de cette étape δ_t est donné dans la Figure 6.18-(5).

– Le document estampillé Δ_t présenté dans la Figure 6.18-(6) est construit en appliquant $TreeUMerge$ sur le document estampillé Δ_{t-1} et la mise à jour partielle δ_t. Comme $doc \in \pi_{\mathbf{no}}$, la racine de Δ_{t-1} est retournée après incrémentation de la valeur de son estampille et la procédure $UMerge_{\mathbf{no}}$ est appliquée sur la forêt de Δ_{t-1} (réduite à l'arbre enraciné en $\Delta_{t-1}@1$) et la forêt de δ_t correspondante.
Dans ce cas, c'est la procédure $TreeUMerge$ qui est appliquée sur ces arbres. En examinant le noeud $\Delta_{t-1}@1$ étiqueté a du côté de Δ_{t-1} et le noeud $\delta_t@1$ étiqueté a du côté de δ_t, $TreeUMerge$ décide de retourner le noeud $\Delta_{t-1}@1$ et d'appliquer $UMerge_{\mathbf{eb}}$ sur le sous-arbre de Δ_{t-1} enraciné en $\Delta_{t-1}@1.1$ et le sous-arbre de δ_t enraciné en $\delta_t@1.1$ puisque $a \in \pi_{\mathbf{eb}}$.
Dans ce cas, c'est la la procédure $TreeUMerge_{\mathbf{eb}}$ qui est appliquée sur ces arbres. Cette procédure n'a qu'un seul choix : retourner le noeud $\Delta_{t-1}@1.1$ de Δ_{t-1} et appliquer $UMerge_{\mathbf{eb}}$ sur les forêts F_s et $\widehat{F_u}$ indiquées dans la Figure 6.18.
Le traitement de ces forêts par $UMerge_{\mathbf{eb}}$ s'effectue de la manière suivante. Les premiers noeuds à être examinés sont le noeud $\Delta_{t-1}@1.1.1$ de Δ_{t-1} et $\delta_t@1.1.1$ de δ_t. Ce cas correspond à la ligne b8 expliquée ci-dessous.

■

Ligne b8. Cette ligne traite le cas où l'étiquette du noeud r_Δ n'appartient pas au tri-projecteur. Dans ce cas, le sous-arbre Δ est simplement retourné et les prochains noeuds dans F_s et $\widehat{F_u}$ sont visités.

Nous pouvons maintenant formellement définir la méthode qui construit l'encodage d'un historique $\mathbf{d}$.

Définition 30 (*It-Update*($\mathbf{d}$)).
Soit un historique $\mathbf{d}$ spécifié par un document XML d_0 et la séquence de mises à jour $u_1, \ldots, u_n$. L'encodage *It-Update*($\mathbf{d}$) de $\mathbf{d}$ est le document Δ_n défini par :

– $\Delta_0 = d_0^{[0,Now[}$, et

– $\Delta_t = Update(\Delta_{t-1}, u_t)$ pour $t = 1..n$, sachant que

$Update(\Delta_{t-1}, u_t) = TreeUMerge(\Delta_{t-1}, u_t(\pi^{\text{Now}}(\Delta_{t-1})), t)$

Le résultat suivant énonce la correction d'encodage spécifié par *It-Update*.

Propriété 12. *Etant donné un historique* $\mathbf{d}$*, on a :*

(i) *It-Update*($\mathbf{d}$) *est un encodage de* $\mathbf{d}$*, et*

(ii) *It-Update*($\mathbf{d}$)$\preceq$*It-Comp*($\mathbf{d}$)*, et donc*

(iii) *It-Update*($\mathbf{d}$)$\preceq \mathrm{Top}(t)$.

Le point (i) exprime la correction de la méthode d'encodage basée sur la projection.

Le point (ii) exprime que lorsque le document abstrait est un historique, la méthode basée sur la projection permet d'obtenir un document plus compacte que la méthode générale *It-Comp*. On peut constater que pour le document abstrait $\mathbf{d}$ de le Figure 6.3-(a) donnée en page 153, la méthode *It-Update* produit le document estampillé Δ' présenté dans la Figure 6.3-(c) qui est plus compact que Δ le résultat de l'application de *It-Comp* sur $\mathbf{d}$. Ce point est validé par les expérimentations que nous avons effectués pour comparer les deux méthodes.

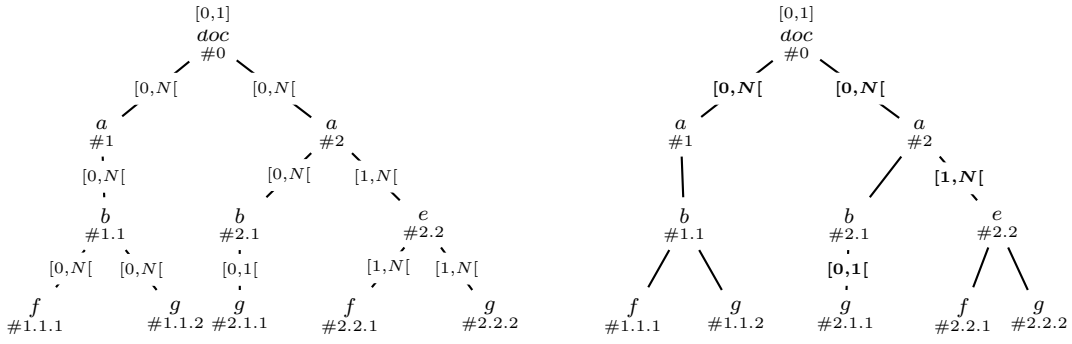
6.3 Implantation et validation expérimentale

L'implantation et l'expérimentation des méthodes *It-Comp* et *It-Update* constituent une contribution personnelle. Nous présentons ici les aspects techniques liés aux implantations des ces méthodes ainsi que les résultats de leur évaluation.

6.3.1 Implantation

L'implantation des méthodes *It-Comp* et *It-Update* suit leurs spécifications formelles présentées dans les Sections 6.2.1 et 6.2.2 respectivement en modifiant certaines hypothèses faites lors de chacune des présentations. L'hypothèse commune à ces deux méthodes concerne le stockage des estampilles. Dans la présentation formelle, nous avions choisi d'associer à chaque noeud d'un document estampillé une estampille. Cette hypothèse s'avère inutile en pratique puisque, le plus souvent, il est possible de déduire l'estampille d'un noeud de manière directe à partir de l'estampille de son parent ou plus généralement de celle d'un ancêtre. Ce même principe a été utilisé dans [BKTT04a] où il est appelé héritage des estampilles. Il est illustré avec l'exemple suivant.

Exemple 84. Considérons les documents estampillés de la Figure 6.19. Le document à gauche de cette Figure représente la version "tout estampillé", le document à droite représente la version qui optimise le stockage des estampilles. Dans la première version, on constate que les noeuds identifiés



Version "tout estampillé" Version estampilles optimisées

FIGURE 6.19 – Optimisation du stockage des estampilles

par 1, 1.1, 1.1.1 et 1.1.2 ont tous la même estampille $[0, N[$. Comme les noeuds 1.1.1 et 1.1.2 sont fils du noeud 1.1 lui même fils du noeud 1, il suffit de stocker l'estampille de ce dernier seulement et ses descendants hériteront de cette estampille. Lorsque cette stratégie est choisie et qu un noeud

possède une estampille différente de celle de son parent ou d'un ancêtre, cette estampille est stockée. Ceci est le cas des noeuds identifié par 2 et par 2.1.

Pour ce document, qui est peu profond, on constate que la stratégie avec optimisation permet un gain de plus de 50% par rapport à la stratégie "tout estampillé". Dans le cas pratique où les documents sont très volumineux ce gain peut être plus important. ■

La deuxième hypothèse concerne les identifiants explicites des noeuds. Ces identifiants sont stockés uniquement pour les documents estampillés générés par la méthode *It-Comp* puisqu'ils sont utilisés pour identifier les noeuds qui capturent le même élément. Les documents estampillés générés par la méthode *It-Update* ne contiennent pas d'identifiants explicites puisque l'identification des noeuds se fait de manière sémantique en utilisant les requêtes de mises à jour. La gestion des positions pour la méthode *It-Update* a déjà été commentée lors de la présentation de cette méthode et reprend la stratégie mise en oeuvre pour l'évaluation de mises à jour par la méthode de projection.

L'implantation de la méthode *It-Comp* a été réalisée en Java en utilisant deux threads permettant de gérer le parsing de Δ_{t-1} et d_t. Ces deux threads interagissent suivant le mode producteur-consommateur un peu comme la méthode de fusion développée dans le Chapitre 4. L'utilisation du paradigme producteur-consommateur permet de gérer facilement la synchronisation des parsings de Δ_{t-1} et d_t qui se base sur l'examen des identifiants explicites.

L'implantation de la méthode *It-Update* possède des points en commun avec l'implantation de la fusion présentée dans le 4. L'implantation de la projection temporelle a nécessité d'adapter l'implantation de la projection pour les types afin de prendre en compte les estampilles, elle ne nécessite pas de commentaires particuliers.

L'implantation de la fusion temporelle $TreeUMerge$ a été réalisée de manière à prendre en compte efficacement certains aspects techniques tels que la gestion optimisée des estampilles. Le principe utilisé dans cette implantation reste le même et utilise les threads afin de gérer le parsing des documents Δ_{t-1} et $u_t(\pi^{\text{Now}}\Delta_{t-1})$.

Scénario des tests Les expérimentations ont été réalisées sur un machine équipée d'un processeur Intel Core 2 Duo d'une fréquence de 2.53 GHz, possédant une mémoire vive de 2Go et utilisant le système d'exploitation Mac OSX version 10.6.4. Le moteur mémoire-centrale utilisé pour effectuer le test de la méthode *It-Update* est Qizx [qiza].

Actuellement, il n'existe pas de benchmark pour les données XML temporelles ni même de benchmark pour les mises à jour XML. Afin d'effectuer nos tests, nous avons généré des documents abstraits à partir de documents XMark [SWK+02]. Comme notre objectif est de comparer les méthodes *It-Comp* et *It-Update*, nous avons choisi de traiter des documents abstraits qui représentent un historique. Nous considérons quatre documents abstraits $\mathbf{d}_0, ..., \mathbf{d}_3$ où, pour $i = 0, .., 3$ chaque $\mathbf{d}_i$ est de la forme : $d_0^i, d_1^i, d_2^i, d_3^i$, et où, pour $j = 0, .., 3$ chaque d_j^i est obtenu en appliquant une mise à jour u_j sur le document d_{j-1}^i.

Les mises à jour utilisées sont données ci-dessous. Les tailles des document initiaux d_0^i de chaque historique d_i varient entre 45 Mo et 1Go.

u_1. **delete node** /site/regions/australia

u_2. **for** $x *in* /site/closed_auctions/closed_auction
where $x/annotation
return insert node <amount> to be determined </amount>
after $x/price

u_3. **for** $x *in* /site/open_auctions/open_auction
where $x/privacy
return
delete node $x

Rappelons que la méthode *It-Comp* nécessite de stocker les identifiants explicites qu'elle utilise pour identifier les éléments. Dans le but d'évaluer cette méthode, nous avons dû générer les documents d_j^i avec des identifiants explicites associés à leurs noeuds. Cette tâche a nécessité un effort supplémentaire d'implémentation entraîné par le besoin de gérer l'interaction entre l'application des mises à jour et la génération des identifiants explicites et par la nécessité de manipuler des documents volumineux (notons que notre objectif est de tester *It-Comp* pour des documents atteignant 1 Go). Le but de gérer l'interaction entre l'application des mises à jour et la génération des identifiants explicites est de respecter l'hypothèse faite au début de ce chapitre selon laquelle les identifiants explicites des noeuds supprimés ne doivent pas être réutilisés. Dans notre cas, le travail a consisté à empêcher que les identifiants des noeuds supprimés par la mise à jour u_1 ne soient attribués aux noeuds insérés par la mise à jour u_2. Enfin, pour pouvoir générer des documents de taille de 1 Go, à partir des mises à jour u_1, u_2 et u_3, nous avons eu recours à la méthode de projection pour dans le Chapitre **??**.

Il est important de rappeler que l'évaluation de la méthode *It-Update* ne nécessite pas un traitement particulier puisqu'elle utilise le document initial d_0 et seulement des mises à jour .

Les résultats des tests sont donnés dans la Table 6.1. Dans cette Table :

- Δ_i est le résultat de l'application de *It-Comp* sur les documents $d_0, \ldots, d_i$,
- Δ'_i est le résultat de l'application de l'application de la mise à jour u_i sur le document estampillé Δ'_{i-1} en utilisant la méthode *It-Update.*

Nous avons comparé la taille des documents estampillés Δ_i et Δ'_i pour i allant de 0 à 3. La différence de taille pour les documents estampillés Δ_0 et Δ'_0 est due aux identifiants explicites générés pour le besoin de la méthode *It-Comp*. Remarquons que la taille du document initial d_0 est égale à la taille de sa version estampillée Δ'_0 puisque, initialement, seule la racine de Δ'_0 est estampillée.

Les résultats présentés dans la Table 6.1 montrent clairement que la méthode *It-Update* est plus efficace en terme de stockage que la méthode *It-Comp*. Alors que la taille des documents Δ_i augmente à chaque étape, la taille des document Δ'_i reste constante. Cette différence est due au fait que *It-Comp* réplique des noeuds Cela montre que *It-Update* parvient à éviter la réplication des noeuds en utilisant l'information des mises à jour. La gain en terme d'espace obtenu en utilisant la méthode *It-Update* est significatif. Il est de 38.5 % pour la première mise à jour, et arrive jusqu'à 50.9% pour la dernière mise à jour.

Pour ces expérimentations, nous avons considéré un document abstrait de taille 4. L'amélioration de la taille des documents estampillés obtenue en utilisant la méthode *It-Update* relativement

d_0	**45.4MB**	**112.4MB**	**454.8MB**	**1126.8MB**
Δ_0	55.7	138.5	563.5	1351.7
Δ'_0	45.4	112.4	454.8	1126.8
Δ_1	76.5	190.5	774.7	1833.0
Δ'_1	45.4	112.4	454.8	1126.8
gain	40.7%	41.0%	41.3%	38.5%
Δ_2	84.5	210.1	853.6	2027.5
Δ'_2	45.6	112.9	456.4	1130.7
gain	46.0%	46.3%	46.5%	44.2%
Δ_3	91.7	228.6	929.0	2207.5
Δ'_3	45.6	112.9	456.4	1130.7
gain	50.3%	50.6%	50.9%	48.8%

TABLE 6.1 – Taille des documents construits par *It-Update* et *It-Comp*

à la taille des documents construits par la méthode *It-Comp* est susceptible de rester stable pour des scénarios plus réalistes où la taille des documents abstraits est plus grande. En effet, plus la taille (de la séquence) du document abstrait est importante, plus grande sera la probabilité pour *It-Comp* de répliquer des noeuds alors que la méthode *It-Update* est capable d'éviter cette réplication grâce à l'utilisation de l'information sur les mises à jour. Des expérimentations pour tester les deux méthodes pour des documents abstraits de plus grande taille sont en cours de réalisation.

Pour conclure, il est important de souligner que même si la méthode *It-Comp* n'est pas capable de produire des documents estampillés aussi compacts que ceux construits par *It-Update*, elle reste particulièrement intéressante car elle permet de traiter des documents volumineux quelconques. Notons que la méthode *It-Update* permet de traiter, en utilisant le moteur QizX, des documents atteignant 1Go alors même que la taille maximale des documents pouvant être traités par ce moteur est de 580 Mo.

6.4 Conclusion et perspectives

Dans ce chapitre, nous avons présenté deux méthodes qui permettent de générer et de maintenir des encodages pour les documents XML temporels. La première méthode traite le cas où aucune information n'est disponible sur le document abstrait. Bien que cette technique ne soit pas totalement satisfaisante en terme de stockage, elle permet de traiter et de maintenir des documents volumineux. La seconde méthode basée sur les mises à jour, est développée pour manipuler les historiques. Elle exploite la technique de projection présentée dans le Chapitre 4 pour permettre la mise à jour de documents temporels volumineux en utilisant des moteurs mémoire-centrale. L'évaluation expérimentale a permis de confirmer l'efficacité des deux méthodes concernant notre premier objectif, celui de permettre le traitement de documents de grande taille. Cette évaluation montre aussi que la méthode basée sur les mises à jour est plus efficace relativement au stockage que la méthode générale.

Les méthodes présentées ici peuvent bénéficier d'améliorations et d'extensions que nous présentons dans ce qui suit.

Extension pour la méthode générale Une extension intéressante consisterait à étudier la manière dont la description du changement peut être exploitée pour améliorer l'efficacité, en terme de stockage, des documents estampillés. Dans [BKTT04b], les auteurs suggèrent l'utilisation des deltas pour la construction de documents estampillés (cf. Section 5.3, page 138). Cette idée a simplement été mentionnée comme solution à un problème spécifique pouvant survenir lors de la fusion de noeuds ne possédant pas de clés sans être développée en détail.

D'une manière générale, l'idée d'exploiter les deltas pour la construction de documents estampillés mérite d'être explorée plus en profondeur afin de mesurer l'impact que cela pourrait avoir sur la réduction de la taille des documents estampillés générés. Une des pistes pouvant être explorée est la possibilité d'intégrer les évolutions spécifiées par les deltas directement dans les documents estampillés. Etant donnée la taille de ces documents, l'objectif est de permettre l'application des deltas en streaming. Dans la Section 4.6 à la page 122, nous avions mentionné une technique [CGM11] qui permet d'appliquer, en streaming, une PUL sur un document statique. Cette technique montre bien qu'il est possible d'envisager la solution discutée ici, à condition bien sûr de prendre en compte l'estampillage puisque le but est de générer des documents estampillés.

Extension pour la méthode basée sur les mises à jour Une des améliorations que nous envisageons pour cette méthode concerne l'accélération de l'étape de fusion temporelle $TreeUMerge$. Cette étape a pour but d'intégrer l'effet d'une mise à jour dans un document estampillé. Elle nécessite de parcourir un document estampillé volumineux même s'il n'y a qu'une petite partie du document qui est modifiée. Afin d'accélérer de cette étape, nous envisageons d'exploiter une technique de partitionnement. L'idée est de découper le document en plusieurs parties et d'exploiter ce partionnement pour paralléliser le traitement. Le problème à résoudre ici est la stratégie de partionnement.

Proposition d'une langage de requêtes temporelles L'interrogation temporelle est une perspective que nous comptons explorer. Actuellement, il existe peu de travaux qui proposent un langage d'interrogation temporel pour les documents XML. Dans [GS03a], une extension temporelle de XQuery est proposée. Cependant, le langage décrit offre des fonctionnalités très limitées et ne permet pas par exemple d'exprimer des conditions sur les estampilles. Ce langage permet essentiellement de retracer l'historique d'un élément et d'extraire un snapshot i.e. l'état du document ou d'une partie du document à un instant donné. Dans [WZ08], les auteurs proposent l'utilisation d'opérateurs d'une logique temporelle pour l'interrogation des documents XML. Leur proposition est décrite sur la base de quelques exemples et ni la syntaxe ni la sémantique du langage ne sont développés.

Notre objectif est de proposer un langage temporel basé sur une logique temporelle et de définir formellement sa syntaxe et sa sémantique. Ce travail pourra bien entendu s'inspirer d'approches existantes dans le cas relationnel [Tom98, SK95, BO07].

Annexe A

Extraction des paths pour les requêtes

Cette annexe est dédiée à la présentation de l'analyse statique pour les chemins esquissée dans le Chapitre 4.

L'analyse pour un chemin P vise à :

1. extraire tous les sous-chemins de P accédant à des textes, et
2. examiner les conditions utilisées dans P afin d'en extraire les chemins accédant soit à des textes soit à des noeuds éléments.

Cette analyse consiste à analyser chaque étape de P et nécessite de garder trace des étapes parcourues. C'est le rôle du *chemin contextuel* noté $P_{\mathtt{ctx}}$ qu'on introduit pour le besoin de cette analyse.

Le jugement **(QPathExt)** utilisé pour l'extraction des chemins à partir des requêtes est modifié pour tenir compte des besoins de l'analyse les chemins. Ce jugement devient :

$$(\Gamma, P_{\mathtt{ctx}}, q) \rightsquigarrow_{\mathtt{all}} \boldsymbol{P} \quad (\mathbf{QPathExt - bis})$$

où Γ, q, $\boldsymbol{P}$ et $\mathbf{all}$ sont comme pour le jugement introduit **(QPathExt)** initialement.

La Table ci-dessous rappelle les notations utilisées pour désigner les chemins extraits pour chaque catégorie.

catégorie	**jugement**
node returned	$(\Gamma, P_{\mathtt{ctx}}, q) \rightsquigarrow_{\mathtt{nr}} \boldsymbol{N}^{\mathtt{R}}$
string returned	$(\Gamma, P_{\mathtt{ctx}}, q) \rightsquigarrow_{\mathtt{sr}} \boldsymbol{S}^{\mathtt{R}}$

catégorie	**jugement**
node used	$(\Gamma, P_{\mathtt{ctx}}, q) \rightsquigarrow_{\mathtt{nu}} \boldsymbol{N}^{\mathtt{U}}$
string used	$(\Gamma, P_{\mathtt{ctx}}, q) \rightsquigarrow_{\mathtt{su}} \boldsymbol{S}^{\mathtt{U}}$
$\forall$ *below used*	$(\Gamma, P_{\mathtt{ctx}}, q) \rightsquigarrow_{\mathtt{ebu}} \boldsymbol{E}$

L'analyse pour les chemins a pour objectif l'examen des conditions dans ces chemins. Ces conditions peuvent être de simples expressions booléennes dont les chemins ciblent des noeuds utilisés seulement par l'expression comme c'est le cas pour la condition suivante $[\mathrm{c/f} \wedge /\mathrm{e/c/d}]$. Elles peuvent aussi consister en des expressions de comparaisons dont les chemins ciblent les

racines d'éléments devant être extraits pour le besoin de la comparaison comme c'est le cas des chemins $\mathrm{c/f}$ et $\mathrm{d/g}$ de la condition $\mathrm{c/f} = \mathrm{d/g}$. Cela suggère que l'on doit identifier, en plus des chemins pouvant cibler des textes (appelés chemins $string$), ceux qui ciblent des noeuds seulement accédés par les conditions (appelés chemins $node$) et ceux qui ciblent des éléments nécessaires à l'évaluation des conditions (appelés chemins $\forall$ below).

De la même façon que nous avons introduit un jugement pour spécifier l'analyse pour les chemins, nous introduisons un autre jugement pour spécifier l'analyse pour les conditions. Ce jugement est donné comme suit :

$$C, \Gamma \vdash_{\overline{cat}} (\mathcal{P}^{\mathtt{abs}}, \mathcal{P}^{\mathtt{rel}}) \quad (\mathbf{CondPathExt})$$

Il entend extraire, à partir de la condition C et en s'appuyant sur l'environnement statique Γ, l'ensemble des chemins absolus $\mathcal{P}^{\mathtt{abs}}$ et l'ensemble des chemins relatifs $\mathcal{P}^{\mathtt{rel}}$ pour la catégorie de chemins cat.
La Table suivante introduit les notations utilisées concernant les chemins extraits à partir des conditions.

category	judgement
node	$C, \Gamma \vdash_{\overline{n}} (\mathcal{N}^{\mathtt{abs}}, \mathcal{N}^{\mathtt{rel}})$
string	$C, \Gamma \vdash_{\overline{s}} (\mathcal{S}^{\mathtt{abs}}, \mathcal{S}^{\mathtt{rel}})$
$\forall$ *below*	$C, \Gamma \vdash_{\overline{eb}} (\mathcal{E}^{\mathtt{abs}}, \mathcal{E}^{\mathtt{rel}})$

Les règles d'analyse des chemins sont guidées par la syntaxe donnée dans le Chapitre 2. Rappelons que dans le but d'empêcher les conditions imbriquées, notre syntaxe est structurée en deux niveaux :

- un niveau pour définir les chemins *simples* qui sont utilisés dans la construction de conditions et qui ne peuvent pas contenir de condition ; ils sont générés à partir du symbole p de la syntaxe ;
- un autre niveau pour définir les chemins pouvant contenir des conditions ; ils sont générés à partir du symbole P.

Du point de vue de la syntaxe, les chemins simples sont construits à partir d'étapes simples, générées par le symbole $step$. La forme générale d'un chemin simple est comme suit :

$$[/]step_0/step_1/\ldots/step_n$$

Les chemins sont construits à partir d'étapes composées qui consistent en une étape simple $step$ suivie éventuellement d'une condition $cond$; ces étapes sont générées par le symbole $STEP$ de notre syntaxe. La forme générale d'un chemin est comme suit :

$$step_0[cond_0]/step_1[cond_1]/\ldots/step_n[cond_n]$$

où chaque $cond_i$ est exprimée à partir de chemins simples $p_i, p_i', \ldots$.

Bien qu'ils soient différents, l'analyse pour ces deux types de chemins s'effectue de manière analogue : chaque étape $step$ est examinée afin d'extraire les sous-chemins *string used* et éventuellement les sous-chemins *node used* et $\forall$ *below used* puis le chemin contextuel $P_{\mathtt{ctx}}$ est mis à jour pour être utilisé dans l'analyse de l'étape suivante.

Nous ne donnons les règles d'extraction que pour les cas suivants :

- étapes simples $step$
- étapes composées $STEP$ données sous la forme $step[cond]$
- chemins $STEP/P$
- conditions

Nous omettons les règles pour le cas $step/p$ qui se traite de manière analogue au cas $STEP/P$.

Dans la suite, étant donné un chemin P et un ensemble de chemins $\boldsymbol{P}$, nous notons par $P{\cdot}\boldsymbol{P}$ l'ensemble des chemins obtenus par concaténation de P avec chaque chemin P' appartenant à $\boldsymbol{P}$, i.e $P{\cdot}\boldsymbol{P} = \{P/P' \mid P' \in \boldsymbol{P}\}$.

Analyse des chemins simples L'analyse d'un chemin simple p a pour but d'extraire tous ses sous-chemins p' accédant à des textes. Elle consiste à examiner chaque étape de p et vérifier si elle cible exclusivement des noeuds ou peut éventuellement cibler des textes. Cette vérification est assurée par la fonction $\texttt{STR}_D(P)$ dont nous rappelons la définition ci-dessous.

$$\texttt{STR}_D(P) = \begin{cases} true & \text{si } P = P'/Axis\texttt{::text}() \\ & \text{ou } P = P'/Axis\texttt{::node}() \text{ avec } String \in \texttt{Type}(D, P) \\ false & sinon \end{cases}$$

doc	$\rightarrow$	$(a \mid e)*$
a	$\rightarrow$	*b*,c*,d ?*
b	$\rightarrow$	*String*
c,d	$\rightarrow$	$(f \mid g \mid String)*$
e	$\rightarrow$	*c*,d ?*

FIGURE A.1 – DTD D

Exemple 85. Soit la DTD donnée dans la Figure A.1. Soit le chemin $P_1 = self :: doc/\texttt{desc} :: \texttt{node}()$. Comme la dernière étape de P_1 est de la forme $Axis :: \texttt{node}()$, il est nécessaire d'inférer le type pour ce chemin pour déterminer la valeur de $\texttt{STR}_D(P_1)$. Le résultat de cette inférence indique que *String* $\in \texttt{Type}(D, P_1)$. Donc $\texttt{STR}_D(P_1) = true$. ■

Les règles d'extraction pour les chemins simples sont données dans la Figure A.2. Le principe utilisé par ces règles est simple : le (seul) chemin *returned* est obtenu par concaténation du chemin contextuel $P_{\texttt{ctx}}$ à $step$. Ce chemin est retourné dans la catégorie *node returned* ou *string returned* en fonction du résultat de $\texttt{STR}_D(P)$.

Analyse des étapes composées L'analyse des étapes composées $step[cond]$ vise à extraire les chemins exprimés dans les conditions. Elle s'appuie à la fois sur l'analyse des étapes simples $step$ présentées ci-dessus et sur l'analyse des conditions que nous présentons ultérieurement. La règle générale pour l'extraction des chemins *string used* et *node used* à partir de $step[cond]$ est comme suit : les chemins exprimés dans $cond$ sont d'abord extraits pour chacune des catégories *string*, *node*

$$(\textbf{step:used}) \quad \frac{}{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{use}} \emptyset}$$

$$(\textbf{step:nr}) \quad \frac{\texttt{STR}_D(P_{\texttt{ctx}}/step){=}false}{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{nr}} \{P_{\texttt{ctx}}/step\}}$$

$$(\textbf{step:nr-empty}) \quad \frac{\texttt{STR}_D(P_{\texttt{ctx}}/step){=}true}{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{nr}} \emptyset}$$

$$(\textbf{step:sr}) \quad \frac{\texttt{STR}_D(P_{\texttt{ctx}}/step){=}true}{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{sr}} \{P_{\texttt{ctx}}/step\}}$$

$$(\textbf{step:sr-empty}) \quad \frac{\texttt{STR}_D(P_{\texttt{ctx}}/step){=}false}{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{sr}} \emptyset}$$

FIGURE A.2 – Extraction pour les étapes simples.

$$(\textbf{STEP:nu}) \quad \frac{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{nr}} \boldsymbol{N}^{\texttt{R}} \quad (\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{sr}} \boldsymbol{S}^{\texttt{R}} \quad cond, \Gamma \vdash_{\overline{\boldsymbol{n}}} (\mathcal{N}^{\texttt{abs}}, \mathcal{N}^{\texttt{rel}})}{(\Gamma, P_{\texttt{ctx}}, step[cond]) \rightsquigarrow_{\textbf{nu}} \mathcal{N}^{\texttt{abs}} \cup (\boldsymbol{N}^{\texttt{R}}.\mathcal{N}^{\texttt{rel}}) \cup (\boldsymbol{S}^{\texttt{R}}.\mathcal{N}^{\texttt{rel}})}$$

$$(\textbf{STEP:su}) \quad \frac{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{nr}} \boldsymbol{N}^{\texttt{R}} \quad (\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{sr}} \boldsymbol{S}^{\texttt{R}} \quad cond, \Gamma \vdash_{\overline{\boldsymbol{s}}} (\mathcal{S}^{\texttt{abs}}, \mathcal{S}^{\texttt{rel}})}{(\Gamma, P_{\texttt{ctx}}, step[cond]) \rightsquigarrow_{\textbf{su}} \mathcal{S}^{\texttt{abs}} \cup (\boldsymbol{N}^{\texttt{R}}.\mathcal{S}^{\texttt{rel}}) \cup (\boldsymbol{S}^{\texttt{R}}.\mathcal{S}^{\texttt{rel}})}$$

$$(\textbf{STEP:ebu}) \quad \frac{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{nr}} \boldsymbol{N}^{\texttt{R}} \quad (\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\textbf{sr}} \boldsymbol{S}^{\texttt{R}} \quad cond, \Gamma \vdash_{\overline{\boldsymbol{eb}}} (\mathcal{E}^{\texttt{abs}}, \mathcal{E}^{\texttt{rel}})}{(\Gamma, P_{\texttt{ctx}}, step[cond]) \rightsquigarrow_{\textbf{ebu}} \mathcal{E}^{\texttt{abs}} \cup (\boldsymbol{N}^{\texttt{R}}.\mathcal{E}^{\texttt{rel}}) \cup (\boldsymbol{S}^{\texttt{R}}.\mathcal{E}^{\texttt{rel}})}$$

$$(\textbf{STEP:ret}) \quad \frac{(\Gamma, P_{\texttt{ctx}}, step) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\texttt{R}}}{(\Gamma, P_{\texttt{ctx}}, step[cond]) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\texttt{R}}} \ \alpha{\in}\{\textbf{nr}, \textbf{sr}\}$$

FIGURE A.3 – Extraction pour les étapes composées.

et $\forall$ *below*. S'il s'agit de chemins absolus, ils sont retournés directement. S'il s'agit de chemins relatifs, ils sont d'abord concaténés au préfixe $P_{\texttt{ctx}}/step$ avant d'être retournés.

Les règles d'extraction pour les étapes composées sont données dans la Figure A.3. Les règles (**STEP**:**nu**), (**STEP**:**su**) et (**STEP**:**ebu**) spécifient l'extraction des chemins *node used*, *string used* et $\forall$ *below used* respectivement suivant le principe que nous venons de présenter. Dans ces trois règles, le préfixe $P_{\texttt{ctx}}/step$ est extrait comme chemin *node returned* ou *string returned* de $step$ suivant les règles de la Figure A.2. Notez que $\boldsymbol{N}^{\texttt{R}}$ et $\boldsymbol{S}^{\texttt{R}}$ sont des singletons mutuellement exclusifs puisque, d'une part, il ne peut y avoir qu'un chemin *returned* extrait de $step$ qui est donné par $P_{\texttt{ctx}}/step$ d'après les règles de la Figure A.2, et d'autre part, ce chemin est soit *node returned* soit *string returned* d'après la définition de $\texttt{STR}_D(P_{\texttt{ctx}}/step)$. La règle (**STEP**:**ret**) spécifie l'extraction du chemin *node returned* respectivement *string returned* de $step[cond]$ qui n'est autre que le chemin *node returned* respectivement *string returned* extrait à partir de $step$.

Exemple 86. Considérons le chemin P_2 spécifié par :

$self :: doc/\texttt{desc::node}()[c/f \vee /\texttt{desc-or-self} :: doc/e = <e><c></c></e>]/\texttt{ancs::node}()$

qui contient une disjonction de conditions : la première condition est réduite à un chemin relatif c/f, la deuxième est une égalité structurelle visant à comparer le contenu du sous-élément enraciné en e retourné par le chemin absolu $/\texttt{desc-or-self} :: doc/e$. Considérons la DTD D de la Figure A.1. L'analyse de P_2 pour D se déroule comme suit :

Etape $self :: doc$

comme il s'agit d'une étape simple ciblant un noeud (doc), seul le chemin $\{self :: doc\}$ est extrait dans la catégorie *node returned* en utilisant la règle (**step:nr**) de la Figure A.2.

Etape $\texttt{desc::node}()[c/f \vee /\texttt{desc-or-self::}doc/e = <e><c></c></e>]$

Cette étape est de la forme $step[cond]$ où :

– $step = \texttt{desc::node}()$, et

– $cond = c/f \vee /\texttt{desc-or-self::}doc/e = <e><c></c></e>$

Le chemin contextuel utilisé pour l'analyse de cette étape est $self :: doc$. La Table ci-dessous donne les chemins de cette étape en indiquant la catégorie et règle d'extraction utilisée.

Catégorie	Chemins	Règle utilisée
node used	$\{self :: doc/\texttt{desc::node}()/c/f\}$	(**STEP:nu**)
string used	$\{self :: doc/\texttt{desc::node}()\}$	(**STEP:su**)
$\forall$ *below used*	$\{/\texttt{desc-or-self::}doc/e\}$	(**STEP:ebu**)
node returned	$\{self :: doc/\texttt{desc::node}()\}$	(**STEP:nr**)
string returned	$\emptyset$	(**STEP:sr**)

Etape $\texttt{ancs::node}()$

Pour cette étape, le chemin contextuel est donné par $self :: doc/\texttt{desc::node}()$. Seul le chemin $self :: doc/\texttt{desc::node}()/\texttt{ancs::node}()$ est extrait de l'analyse de cette étape dans la catégorie *node returned*.

■

Analyse des chemins L'analyse d'un chemin P s'effectue en examinant chacune de ses étapes composées $STEP$ de la forme $step[cond]$. Chaque étape est analysée relativement à un ensemble de chemins préfixes $\boldsymbol{P}_{\texttt{ctx}}$. Rappelons d'ailleurs la règle (**QPE:path**) qui initialise l'analyse des chemins et qui a été introduite dans la Figure 4.13.

(**QPE:path**) $\dfrac{(\Gamma, \{\epsilon\}, P) \rightsquigarrow_{\text{all}} \boldsymbol{P}_{\textbf{all}}}{(\Gamma, P) \rightsquigarrow_{\text{all}} \boldsymbol{P}_{\textbf{all}}}$ (ϵ chemin vide)

Les règles d'analyse des chemins utilisent donc les jugements $(\Gamma, \boldsymbol{P}_{\texttt{ctx}}, P) \rightsquigarrow_{\text{all}} \boldsymbol{P}_{\textbf{all}}$ et sont données dans la Figure A.4.

La règle (**context:vide:all**) traite le cas où tous les chemins de $\boldsymbol{P}_{\texttt{ctx}}$ ont déjà été passés en préfixe de P pour l'analyse relative à la catégorie **all**.

La règle (**STEP**:**all**) effectue l'extraction pour le cas de base où P est réduit à $STEP$. Donc cette règle a recours à l'analyse de $STEP$ avec chacun des chemins préfixes de $\boldsymbol{P}_{\mathtt{ctx}}$ successivement (itération sur l'ensemble $\boldsymbol{P}_{\mathtt{ctx}}$) en fonction de la catégorie **all**. Notons que les règles qui définissent le jugement $(\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathtt{all}} \boldsymbol{P}_S^{\mathbf{all}}$ sont définies dans les Figures A.3 et A.2.

Les règles restantes sont destinées à traiter le cas où le chemin à analyser est de la forme $STEP/P$ et où l'ensemble des chemins préfixes $\boldsymbol{P}_{\mathtt{ctx}}$ n'est pas vide. Donc ces règles (voir leur prémisse) sont toutes construites

1. sur la base d'une itération sur l'ensemble $\boldsymbol{P}_{\mathtt{ctx}}$ (les chemins de $\boldsymbol{P}_{\mathtt{ctx}}$ sont passés en préfixe de $STEP/P$ les uns après les autres), et
2. pour chaque chemin préfixe $P_{\mathtt{ctx}}$ de $\boldsymbol{P}_{\mathtt{ctx}}$ sur la base du traitement de $STEP$ qui produit les préfixes pour l'analyse de P.

Pour chaque catégorie de chemins, deux règles exclusives sont données qui correspondent chacune respectivement au cas où $STEP$ cible un noeud élément et au cas dual où $STEP$ cible potentiellement un texte. Dans le premier cas, $\boldsymbol{N}_S^{\mathtt{R}}$ est non vide impliquant que $\boldsymbol{S}^{\mathtt{R}}$ est vide. Dans le deuxième cas, c'est $\boldsymbol{S}_S^{\mathtt{R}}$ qui est non vide.

Dans la suite, pour chaque paire de règles (**xx**:**1**) et (**xx**:**2**), nous nous contentons de commenter ce qui est relatif au traitement de $STEP$ pour un chemin préfixe $P_{\mathtt{ctx}}$. Nous ne commentons pas l'itération sur l'ensemble des préfixes.

Les règles (**STEP**/**P**:**nu**−**ebu**:**1**) et (**STEP**/**P**:**nu**−**ebu**:**2**) spécifient l'extraction des chemins *node used* et $\forall$ *below used* suivant la valeur de α. Ces chemins sont obtenus à partir des chemins *node used* (resp. $\forall$ *below used*) extraits de $STEP$, des chemins *node used* (resp. $\forall$ *below used*) extraits de P modulo les préfixes constitués des chemins *node returned* extrait de $STEP$ pour la règle (**STEP**/**P**:**nu**−**ebu**:**1**) (respectivement constitués des chemins *string returned* extrait de $STEP$ pour la règle (**STEP**/**P**:**nu**−**ebu**:**2**)).

Les règles (**STEP**/**P**:**su**:**1**) et (**STEP**/**P**:**su**:**2**) spécifient l'extraction des chemins *string used* qui diffère de l'extraction des chemins *node used* et $\forall$ *below used* uniquement parce que le chemin *string returned* de $STEP$, si il existe, est considéré comme un chemin *string used* pour $STEP/P$. Ceci apparaît dans la conclusion de la règle (**STEP**/**P**:**su**:**2**).

Les règles (**STEP**/**P**:**ret**:**1**) et (**STEP**/**P**:**ret**:**2**) spécifient l'extraction du chemin *node returned* respectivement *string returned* qui consiste à extraire le chemin *node returned* respectivement *string returned* de P modulo les préfixes extraits de $STEP$, toujours suivant le même principe.

(**context**:**vide**:**all**)	$\dfrac{}{(\Gamma, \emptyset, P) \rightsquigarrow_{\mathtt{all}} \emptyset}$
(**STEP**:**all**)	$\dfrac{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathtt{all}} \boldsymbol{P}^{\mathbf{all}}_{\mathtt{ctx}} \quad (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathtt{all}} \boldsymbol{P}^{\mathbf{all}}_{S}}{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}} \cup \{P_{\mathtt{ctx}}\}, STEP) \rightsquigarrow_{\mathtt{all}} \boldsymbol{P}^{\mathbf{all}}_{\mathtt{ctx}} \cup \boldsymbol{P}^{\mathbf{all}}_{S}}$
(**STEP**/**P**:**nu**−**ebu**:**1**)	$\dfrac{\begin{array}{c}(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, STEP/P) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\alpha}_{\mathtt{ctx}} \quad (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\mathtt{U}}_{S} \\ (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{nr}} \boldsymbol{N}^{\mathtt{R}}_{S} \quad \boldsymbol{N}^{\mathtt{R}}_{S} \neq \emptyset \quad (\Gamma, \boldsymbol{N}^{\mathtt{R}}_{S}, P) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\mathtt{U}}_{P}\end{array}}{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}} \cup \{P_{\mathtt{ctx}}\}, STEP/P) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\alpha}_{\mathtt{ctx}} \cup \boldsymbol{P}^{\mathtt{U}}_{S} \cup \boldsymbol{P}^{\mathtt{U}}_{P}} \quad \alpha \in \{\mathbf{nu}, \mathbf{ebu}\}$
(**STEP**/**P**:**nu**−**ebu**:**2**)	$\dfrac{\begin{array}{c}(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, STEP/P) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\alpha}_{\mathtt{ctx}} \quad (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\mathtt{U}}_{S} \\ (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{sr}} \boldsymbol{S}^{\mathtt{R}}_{S} \quad \boldsymbol{S}^{\mathtt{R}}_{S} \neq \emptyset \quad (\Gamma, \boldsymbol{S}^{\mathtt{R}}_{S}, P) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\mathtt{U}}_{P}\end{array}}{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}} \cup \{P_{\mathtt{ctx}}\}, STEP/P) \rightsquigarrow_{\alpha} \boldsymbol{P}^{\alpha}_{\mathtt{ctx}} \cup \boldsymbol{P}^{\mathtt{U}}_{S} \cup \boldsymbol{P}^{\mathtt{U}}_{P}} \quad \alpha \in \{\mathbf{nu}, \mathbf{ebu}\}$
(**STEP**/**P**:**su**:**1**)	$\dfrac{\begin{array}{c}(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, STEP/P) \rightsquigarrow_{\mathbf{su}} \boldsymbol{P}^{\mathbf{su}}_{\mathtt{ctx}} \quad (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{su}} \boldsymbol{S}^{\mathtt{U}}_{S} \\ (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{nr}} \boldsymbol{N}^{\mathtt{R}}_{S} \quad \boldsymbol{N}^{\mathtt{R}}_{S} \neq \emptyset \quad (\Gamma, \boldsymbol{N}^{\mathtt{R}}_{S}, P) \rightsquigarrow_{\mathbf{su}} \boldsymbol{S}^{\mathtt{U}}_{P}\end{array}}{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}} \cup \{P_{\mathtt{ctx}}\}, STEP/P) \rightsquigarrow_{\mathbf{su}} \boldsymbol{P}^{\mathbf{su}}_{\mathtt{ctx}} \cup \boldsymbol{S}^{\mathtt{U}}_{S} \cup \boldsymbol{S}^{\mathtt{U}}_{P}}$
(**STEP**/**P**:**su**:**2**)	$\dfrac{\begin{array}{c}(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, STEP/P) \rightsquigarrow_{\mathbf{su}} \boldsymbol{P}^{\mathbf{su}}_{\mathtt{ctx}} \quad (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{su}} \boldsymbol{S}^{\mathtt{U}}_{S} \\ (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{sr}} \boldsymbol{S}^{\mathtt{R}}_{S} \quad \boldsymbol{S}^{\mathtt{R}}_{S} \neq \emptyset \quad (\Gamma, \boldsymbol{S}^{\mathtt{R}}_{S}, P) \rightsquigarrow_{\mathbf{su}} \boldsymbol{S}^{\mathtt{U}}_{P}\end{array}}{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}} \cup \{P_{\mathtt{ctx}}\}, STEP/P) \rightsquigarrow_{\mathbf{su}} \boldsymbol{P}^{\mathbf{su}}_{\mathtt{ctx}} \cup \boldsymbol{S}^{\mathtt{U}}_{S} \cup \boldsymbol{S}^{\mathtt{R}}_{S} \cup \boldsymbol{S}^{\mathtt{U}}_{P}}$
(**STEP**/**P**:**ret**:**1**)	$\dfrac{\begin{array}{c}(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, STEP/P) \rightsquigarrow_{\mathbf{ret}} \boldsymbol{P}^{\mathtt{R}}_{\mathtt{ctx}} \\ (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{nr}} \boldsymbol{N}^{\mathtt{R}}_{S} \quad \boldsymbol{N}^{\mathtt{R}}_{S} \neq \emptyset \quad (\Gamma, \boldsymbol{N}^{\mathtt{R}}_{S}, P) \rightsquigarrow_{\mathbf{ret}} \boldsymbol{P}^{\mathtt{R}}_{P}\end{array}}{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}} \cup \{P_{\mathtt{ctx}}\}, STEP/P) \rightsquigarrow_{\mathbf{ret}} \boldsymbol{P}^{\mathtt{R}}_{\mathtt{ctx}} \cup \boldsymbol{P}^{\mathtt{R}}_{P}}$
(**STEP**/**P**:**ret**:**2**)	$\dfrac{\begin{array}{c}(\Gamma, \boldsymbol{P}_{\mathtt{ctx}}, STEP/P) \rightsquigarrow_{\mathbf{ret}} \boldsymbol{P}^{\mathtt{R}}_{\mathtt{ctx}} \\ (\Gamma, P_{\mathtt{ctx}}, STEP) \rightsquigarrow_{\mathbf{sr}} \boldsymbol{S}^{\mathtt{R}}_{S} \quad \boldsymbol{S}^{\mathtt{R}}_{S} \neq \emptyset \quad (\Gamma, \boldsymbol{S}^{\mathtt{R}}_{S}, P) \rightsquigarrow_{\mathbf{ret}} \boldsymbol{P}^{\mathtt{R}}_{P}\end{array}}{(\Gamma, \boldsymbol{P}_{\mathtt{ctx}} \cup \{P_{\mathtt{ctx}}\}, STEP/P) \rightsquigarrow_{\mathbf{ret}} \boldsymbol{P}^{\mathtt{R}}_{\mathtt{ctx}} \cup \boldsymbol{P}^{\mathtt{R}}_{P}}$

FIGURE A.4 – Extraction pour les chemins composés.

Extraction pour les conditions. L'analyse des conditions permet d'identifier les chemins des catégories suivantes :

- les chemins *node* qui ciblent des noeuds qui sont seulement consultés,
- les chemins *string* qui ciblent des textes,
- les chemins $\forall$ *below* qui ciblent des noeuds dont les sous-éléments sont accédés.

Cette analyse s'appuie à la fois sur l'analyse des chemins simples présentée ci-dessus et sur l'analyse des expressions dans les conditions,

$$(\textbf{cond:bool}) \quad \frac{cond_i, \Gamma \vdash_{\overline{\textbf{all}}} (\mathcal{P}_i^{\texttt{abs}}, \mathcal{P}_i^{\texttt{rel}}) \quad i{=}1,2}{cond_1\ ct\ cond_2, \Gamma \vdash_{\overline{\textbf{all}}} (\mathcal{P}_1^{\texttt{abs}} \cup \mathcal{P}_2^{\texttt{abs}}, \mathcal{P}_1^{\texttt{rel}} \cup \mathcal{P}_2^{\texttt{rel}})} \ \textbf{all} \in \{n, s, eb\}$$

$$(\textbf{cond:exp}) \quad \frac{Exp_i, \Gamma \vdash_{\overline{\textbf{all}}} (\mathcal{P}_i^{\texttt{abs}}, \mathcal{P}_i^{\texttt{rel}}) \quad i{=}1,2}{Exp_1\ cp\ Exp_2, \Gamma \vdash_{\overline{\textbf{all}}} (\mathcal{P}_1^{\texttt{abs}} \cup \mathcal{P}_2^{\texttt{abs}}, \mathcal{P}_1^{\texttt{rel}} \cup \mathcal{P}_2^{\texttt{rel}})} \ \textbf{all} \in \{n, s, eb\}$$

$$(\textbf{cond:arit}) \quad \frac{Arit_i, \Gamma \vdash_{\overline{\textbf{all}}} (\mathcal{P}_i^{\texttt{abs}}, \mathcal{P}_i^{\texttt{rel}}) \quad i{=}1,2}{Arit_1\ op\ Arit_2, \Gamma \vdash_{\overline{\textbf{all}}} (\mathcal{P}_1^{\texttt{abs}} \cup \mathcal{P}_2^{\texttt{abs}}, \mathcal{P}_1^{\texttt{rel}} \cup \mathcal{P}_2^{\texttt{rel}})} \ \textbf{all} \in \{n, s, eb\}$$

$$(\textbf{cond:bool:node}) \quad \frac{p, \Gamma \vdash_{\overline{n}} (\mathcal{N}_p^{\texttt{abs}}, \mathcal{N}_p^{\texttt{rel}}) \quad cond, \Gamma \vdash_{\overline{n}} (\mathcal{N}_c^{\texttt{abs}}, \mathcal{N}_c^{\texttt{rel}})}{p\ ct\ cond, \Gamma \vdash_{\overline{n}} (\mathcal{N}_p^{\texttt{abs}} \cup \mathcal{N}_c^{\texttt{abs}}, \mathcal{N}_p^{\texttt{rel}} \cup \mathcal{N}_c^{\texttt{rel}})}$$

$$(\textbf{cond:bool:string}) \quad \frac{p, \Gamma \vdash_{\overline{s}} (\mathcal{S}_p^{\texttt{abs}}, \mathcal{S}_p^{\texttt{rel}}) \quad cond, \Gamma \vdash_{\overline{s}} (\mathcal{S}_c^{\texttt{abs}}, \mathcal{S}_c^{\texttt{rel}})}{p\ ct\ cond, \Gamma \vdash_{\overline{s}} (\mathcal{S}_p^{\texttt{abs}} \cup \mathcal{S}_c^{\texttt{abs}}, \mathcal{S}_p^{\texttt{rel}} \cup \mathcal{S}_c^{\texttt{rel}})}$$

$$(\textbf{cond:bool:}\forall\textbf{below}) \quad \frac{cond, \Gamma \vdash_{\overline{eb}} (\mathcal{S}_c^{\texttt{abs}}, \mathcal{S}_c^{\texttt{rel}})}{p\ ct\ cond, \Gamma \vdash_{\overline{eb}} (\mathcal{E}_c^{\texttt{abs}}, \mathcal{E}_c^{\texttt{rel}})}$$

$$(\textbf{cond:exp:node}) \quad \frac{Exp, \Gamma \vdash_{\overline{n}} (\mathcal{N}_e^{\texttt{abs}}, \mathcal{N}_e^{\texttt{rel}})}{p\ ct\ Exp, \Gamma \vdash_{\overline{n}} (\mathcal{N}_e^{\texttt{abs}}, \mathcal{N}_e^{\texttt{rel}})}$$

$$(\textbf{cond:exp:string}) \quad \frac{p, \Gamma \vdash_{\overline{s}} (\mathcal{S}_p^{\texttt{abs}}, \mathcal{S}_p^{\texttt{rel}}) \quad Exp, \Gamma \vdash_{\overline{s}} (\mathcal{S}_e^{\texttt{abs}}, \mathcal{S}_e^{\texttt{rel}})}{p\ ct\ Exp, \Gamma \vdash_{\overline{s}} (\mathcal{S}_p^{\texttt{abs}} \cup \mathcal{S}_e^{\texttt{abs}}, \mathcal{S}_p^{\texttt{rel}} \cup \mathcal{S}_e^{\texttt{rel}})}$$

$$(\textbf{cond:exp:}\forall\textbf{below}) \quad \frac{p, \Gamma \vdash_{\overline{n}} (\mathcal{N}_p^{\texttt{abs}}, \mathcal{N}_p^{\texttt{rel}}) \quad Exp, \Gamma \vdash_{\overline{eb}} (\mathcal{S}_e^{\texttt{abs}}, \mathcal{S}_e^{\texttt{rel}})}{p\ ct\ Exp, \Gamma \vdash_{\overline{eb}} (\mathcal{N}_p^{\texttt{abs}} \cup \mathcal{E}_e^{\texttt{abs}}, \mathcal{N}_p^{\texttt{rel}} \cup \mathcal{E}_e^{\texttt{rel}})}$$

FIGURE A.5 – Extraction pour les expressions (des conditions).

La Figure A.5 donne les règles d'extraction pour les expressions qui peuvent être de trois types différents :

- les expressions booléennes dont la forme générale est donnée par : $cond_1\ ct\ cond_2$,
- les expressions de comparaisons dont la forme générale est donnée par : $Exp_1\ cp\ Exp_2$,
- les expression arithmétiques dont la forme générale est donnée par : $Arit_1\ op\ Arit_2$.

Rappelons que la syntaxe des chemins permet de construire les expressions booléennes à partir d'autres expressions booléennes mais aussi à partir d'expressions de comparaisons. De même,

les expressions de comparaison sont construites à partir d'expressions de comparaisons ainsi qu'à partir d'expressions arithmétiques. Enfin les expressions arithmétiques sont construites à partir des chemins et des littéraux. Par conséquent, l'extraction des chemins à partir des expressions doit considérer les chemins extraits de chaque sous-expression.

Les règles (**cond**:**bool**), (**cond**:**exp**) et (**cond**:**arit**) spécifient que les trois types de chemins sont obtenus, pour chaque type d'expression, à partir de ses deux sous-expressions. Les règles (**cond**:**bool**:**node**), (**cond**:**bool**:**string**) et (**cond**:**bool**:∀**below**) raffinent l'extraction des chemins pour le cas d'une expression booléenne de la forme $p\ ct\ cond$. Elles spécifient que les chemins *node* extraits de p sont retournés comme chemins *node* de l'expression $p\ ct\ cond$. En effet, il n'est pas nécessaire de projeter le sous-éléments des noeuds ciblés par p pour pouvoir évaluer l'expression booléenne $p\ ct\ cond$.

Les règles (**cond**:**exp**:**node**), (**cond**:**exp**:**string**) et (**cond**:**exp**:∀**below**) permettent de raffiner l'extraction des chemins pour le cas $p\ cp\ Exp$. Elles spécifient que les chemins *node* de p qui ciblent des noeuds dont les sous-éléments nécessaires pour l'évaluation de l'expression, doivent être considérés comme chemin ∀ *below*. Ces mêmes règles s'appliquent pour le cas $p\ op\ Arit$.

(**valeur**) $$\frac{}{\nu, \Gamma \vdash_{\mathbf{all}} (\emptyset, \emptyset)}\ \mathbf{all}{\in}\{n, s, eb\}$$

(**path**:**abs**) $$\frac{(\Gamma, \epsilon, p) \rightsquigarrow_{\beta} \boldsymbol{P}^{\mathtt{R}} \quad (\alpha, \beta){\in}\{(n, nr), (s, sr)\}}{p, \Gamma \vdash_{\alpha} (\boldsymbol{P}^{\mathtt{R}}, \emptyset)}\ \mathtt{Abs(p)}{=}true$$

(**path**:**rel**) $$\frac{(\Gamma, \epsilon, p) \rightsquigarrow_{\beta} \boldsymbol{P}^{\mathtt{R}} \quad (\alpha, \beta){\in}\{(n, nr), (s, sr)\}}{p, \Gamma \vdash_{\alpha} (\emptyset, \boldsymbol{P}^{\mathtt{R}})}\ \mathtt{Abs(p)}{=}false$$

(**path**:**eb**) $$\frac{}{p, \Gamma \vdash_{eb} (\emptyset, \emptyset)}$$

(**function**:**nu**) $$\frac{Exp_i, \Gamma \vdash_{n} (\mathcal{N}_i^{\mathtt{abs}}, \mathcal{N}_i^{\mathtt{rel}}) \quad i{=}1, n}{f(Exp_1, \ldots, Exp_n), \Gamma \vdash_{n} (\cup_{i=1}^{n} \mathcal{N}_i^{\mathtt{abs}}, \cup_{i=1}^{n} \mathcal{N}_i^{\mathtt{rel}})}$$

(**function**:**su**) $$\frac{Exp_i, \Gamma \vdash_{s} (\mathcal{S}_i^{\mathtt{abs}}, \mathcal{S}_i^{\mathtt{rel}}) \quad i{=}1, n}{f(Exp_1, \ldots, Exp_n), \Gamma \vdash_{s} (\cup_{i=1}^{n} \mathcal{S}_i^{\mathtt{abs}}, \cup_{i=1}^{n} \mathcal{S}_i^{\mathtt{rel}})}$$

$$f{\in}\{count, not, position\}$$

FIGURE A.6 – Extraction pour les chemins simples (des conditions).

Les règles (**path**:**abs**) et (**path**:**rel**) identifient d'abord si le chemin p cible des noeuds ou des textes en s'appuyant sur l'extraction des chemins pour les requêtes, puis identifient s'il est relatif ou absolu, à l'aide de la fonction `Abs(p)`. La règle (**path**:**eb**) spécifie qu'aucun chemin n'est extrait de p de la catégorie ∀ below. Les règles (**function**:**nu**) et (**function**:**su**) spécifient l'extraction des chemins des fonctions pré-définies dont nous considérons uniquement les fonctions $count, not$ et $position$. Ces fonctions considèrent uniquement les chemins *node* et *string* qui sont extraits à partir de leurs expressions Exp_i.

Annexe B

Preuves de la correction de l'inférence des paths pour les mises à jour

La preuve de la correction du tri-projecteur pour la pré-évaluation de mises à jour XML (Théorème 3) repose essentiellement sur la preuve du Lemme 4 qui exprime la correction de l'extraction des chemins pour la pré-évaluation de mises à jour XML.

B.1 Preuve du Lemme 4

La preuve du Lemme 4 repose sur la correction de l'extraction des chemins pour les requêtes. Cette dernière est exprimée pour les deux formes de correction :

- correction pour **l'évaluation** des requêtes énoncée par le Lemme 2, et
- correction pour la **pré-évaluation** des requêtes énoncée par le Lemme 3.

Nous rappelons les notations relatives aux id-sets définis les dans ces trois Lemmes.

évaluation des requêtes	$\mathbf{eval}$-$\boldsymbol{K}^q$	Lemme 2, page 95
pré-évaluation des requêtes	$\mathbf{pre}$-$\boldsymbol{K}^q$	Lemme 3, page 98
pré-évaluation des mises à jour	$\boldsymbol{K}^u$	Lemme 4, page 105

Nous rappelons ci-dessous l'énoncé du Lemme 4 :

Lemme 4 (Correction de l'inférence des chemins pour la pré-évaluation des mises à jour).
Soit une DTD D, un document $t \in D$ et une mise à jour u. Alors, l'id-set $\boldsymbol{K}^u$ défini ci-dessous est un id-projecteur correct pour la pré-évaluation de u sur t.

$$\boldsymbol{K}^u = \bigcup_{\alpha \in \{\mathbf{no},\mathbf{olb},\mathbf{eb}\}} \boldsymbol{K}^u_\alpha$$

où pour $\alpha \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$, on pose $UExt_\alpha(/, u) = \{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$

et pour i=1..k_α on a :
- $\mathbf{pi}^i_\alpha$ le path-projecteur pour P^i_α et D,
- KT^i_α=$PEval(t, P^i_\alpha)$,
- KN^i_α=$KPath(t, \mathbf{pi}^i_\alpha)$,
- $\boldsymbol{K}^u_{\mathbf{no}}$=$\bigcup_{i=1}^{k_{\mathbf{no}}}[KN^i_{\mathbf{no}}]$,
- $\boldsymbol{K}^u_{\mathbf{olb}}$=$\bigcup_{i=1}^{k_{\mathbf{olb}}}[KN^i_{\mathbf{olb}} \cup \mathit{Child}(t, KT^i_{\mathbf{olb}})]$,
- $\boldsymbol{K}^u_{\mathbf{eb}}$=$\bigcup_{i=1}^{k_{\mathbf{eb}}}[KN^i_{\mathbf{eb}} \cup \mathit{Desc}(t, KT^i_{\mathbf{eb}})]$.

■

Notre objectif va être de montrer que la pré-évaluation de u sur t est équivalente à la pré-évaluation de u sur $\Pi_{\boldsymbol{K}^u}(t)$ i.e

$$(\dagger) \quad (\sigma_t \cdot \sigma_\omega, \omega) \sim (\Pi_{\boldsymbol{K}^u}(\sigma_t) \cdot \widehat{\sigma_\omega}, \widehat{\omega})$$

où $\boldsymbol{K}^u$ est défini dans le Lemme 4 ci-dessus et

$(\dagger_a)$ $(\sigma_\omega, \omega) = PUL(t, u)$ et $(\dagger_b)$ $(\widehat{\sigma_\omega}, \widehat{\omega}) = PUL(\Pi_{\boldsymbol{K}^u}(t), u)$

Bien évidemment, la preuve du Lemme 4 est effectuée par induction structurelle relativement à la mise à jour u. Avant d'effectuer cette preuve, nous allons en modifier l'objectif présenté ci-dessus pour des raisons techniques. Dans l'induction, les mises à jour par exemple atomiques ($\mathbf{delete}\ q_{\texttt{cib}}, \mathbf{insert}\ q_{\texttt{src}}\ \delta\ q_{\texttt{cib}}$, etc) vont devoir être traitées dans le cas de base (le cas où u est atomique) et dans le cas où elles sont imbriquées dans une mise à jour complexe (le cas où u est par exemple de la forme $\mathbf{for}\ x\ \texttt{in}\ q_{\texttt{ctxt}}\ \mathbf{return}\ u$). Afin de permettre un traitement unique de ces deux cas, nous allons considérer que la mise à jour u n'est pas close (cf. Section 2.4 page 26). Pour rappel, $VarLib(u)$ désigne l'ensemble des variables libres de u.

L'hypothèse que $VarLib(u)$ peut ne pas être vide, a tout d'abord un impact sur la définition de l'id-set $\boldsymbol{K}^u$ dans le Lemme 4. En effet, l'id-set $\boldsymbol{K}^u$ est construit à partir de l'analyse statique de u qui est l'extraction des chemins. L'extraction des chemins à partir de u doit donc prendre en compte le fait que la mise à jour u n'est pas close et fournir via l'environnement statique en entrée de cette analyse des liaisons pour les variables libres de $VarLib(u)$. Ces liaisons associent à chaque variable un ensemble de chemins.

[Environnement statique fermé pour X] Soit X un ensemble de variables, un environnement statique Γ est fermé pour X si $X \subseteq Var(\Gamma)$.

Dans la suite, dans le lemme 4, on pose :

$(\dagger\dagger)$ $UExt_\alpha(\Gamma, u)$=$\{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$ pour Γ un environnement statique fermé pour $VarLib(u)$.

Notons ici que la pré-évaluation de u que ce soit dans $\dagger_a$ ou $\dagger_b$ est réalisée relativement à un contexte dynamique vide. Rappelons qu'un contexte dynamique Υ est un n-uplet $(\sigma_\epsilon, d, \gamma)$ tel que,

- σ_ϵ est un store auxiliaire décrivant les éléments matérialisés lors de l'évaluation de la requête contextuelle de la mise à jour composée,
- d est un ensemble d'identifiants que la pré-évaluation de u doit s'interdire d'utiliser puisqu'ils ont été utilisés lors d'éventuelles matérialisations d'éléments survenues lors de l'évaluation de la requête contextuelle, et
- γ est un environnement dynamique des variables qui garde trace des liaisons variables-identifiants obtenues lors de la pré-évaluation des requêtes composées.

L'hypothèse que $VarLib(u)$ peut ne pas être vide impose donc de considérer la pré-évaluation de u dans $(\dagger_a)$ ou $(\dagger_b)$ relativement à un contexte dynamique non vide, en particulier relativement à un environnement dynamique qui va fournir des liaisons pour les variables libres de u. Donc $(\dagger_a)$ ou $(\dagger_b)$ deviennent :

$$(\dagger_{a1}) \quad (\sigma_\omega, \omega) = \mathit{PUL}_\Upsilon(t, u) \quad \text{et} \quad (\dagger_{b1}) \quad (\widehat{\sigma_\omega}, \widehat{\omega}) = \mathit{PUL}_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^u}(t), u)$$

Bien sûr, les contextes dynamiques Υ et $\widehat{\Upsilon}$ ci-dessus ne peuvent pas être quelconques.

– Ce sont des environnements dynamiques liant toutes les variables libres de u.

[Contexte dynamique fermé pour X] Soit X un ensemble de variables, un contexte dynamique $\Upsilon{=}(\sigma_\epsilon, d, \gamma)$ est fermé pour X si $X \subseteq Var(\gamma)$.

– Ce sont des environnements dynamiques équivalents ie dont les composants respectifs sont isomorphes.

[Contextes dynamiques équivalents] Deux contextes dynamiques $\Upsilon{=}(\sigma_\epsilon, d, \gamma)$ et $\widehat{\Upsilon}{=}(\widehat{\sigma_\epsilon}, \widehat{d}, \widehat{\gamma})$ fermés pour X et définis sur t et $\widehat{t}$ respectivement sont équivalents, noté $\Upsilon{\simeq}\widehat{\Upsilon}$ si il existe une bijection ζ : $\mathit{dom}(\Upsilon) \rightarrow \mathit{dom}(\widehat{\Upsilon})$ telle que :

i) $\zeta(\sigma_\epsilon){=}\widehat{\sigma_\epsilon}$
ii) $\zeta(d){\in}\widehat{d}$ et
iii) $Var(\gamma){=}Var(\widehat{\gamma})$ et pour tout $x \in Var(\gamma)$ x=I $\in \gamma$ implique que $x{=}\zeta(\mathrm{I}){\in}\widehat{\gamma}$.

– Enfin, ce sont des environnements dynamiques qui doivent être consistants avec l'environnement statique Γ qui est utilisé pour la construction de $\boldsymbol{K}^u$. Intuitivement, pour que l'environnement dynamique γ soit consistant avec l'environnement statique Γ, il faut que pour chaque variable $x{\in}VarLib(u)$, la liaison sémantique x=I corresponde à la liaison statique $x{\mapsto}\boldsymbol{P}$ où la notion de correspondance s'exprime simplement par le fait que l'évaluation des chemins de $\boldsymbol{P}$ sur t produit I. Rappelons que d'après l'hypothèse donnée dans le Chapitre 2 en page 36, les liaisons sémantiques ciblent exclusivement des noeuds du store initial σ_t.

[Consistance des environnements statique et dynamique] Soit t un document. L'environnement statique Γ et l'environnement dynamique $\Upsilon{=}(\sigma_\epsilon, d, \gamma)$ sont consistants si $Var(\gamma){=}Var(\Gamma)$ et $\mathrm{I}{=}\bigcup_{P\in\boldsymbol{P}} \mathit{PEval}(\sigma_t, P)$ où x=I$\in\gamma$ et $x{\mapsto}\boldsymbol{P} \in \Gamma$.

Cette discussion préliminaire nous conduit à généraliser le Lemme 4 comme suit :

Lemme 4-bis (Correction généralisée de l'inférence des chemins pour la prévaluation des mises à jour).
Soit une DTD D, un document $t \in D$ et une mise à jour u.

Soit un environnement statique Γ fermé pour $VarLib(u)$.
Soit l'id-set $\boldsymbol{K}^u$ spécifié par :

$$\boldsymbol{K}^u = \bigcup_{\alpha \in \{\mathbf{no},\mathbf{olb},\mathbf{eb}\}} \boldsymbol{K}^u_\alpha$$

où pour $\alpha \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$, on pose $UExt_\alpha(\Gamma, u) = \{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$ et pour $i = 1..k_\alpha$ on a :

- $\mathbf{pi}^i_\alpha$ le path-projecteur pour P^i_α et D,
- $KT^i_\alpha = PEval(t, P^i_\alpha)$,
- $KN^i_\alpha = KPath(t, \mathbf{pi}^i_\alpha)$,

(lem3-no) $\boldsymbol{K}^u_{\mathbf{no}} = \bigcup_{i=1}^{k_{\mathbf{no}}} [KN^i_{\mathbf{no}}]$,
(lem3-olb) $\boldsymbol{K}^u_{\mathbf{olb}} = \bigcup_{i=1}^{k_{\mathbf{olb}}} [KN^i_{\mathbf{olb}} \cup \mathit{Child}(t, KT^i_{\mathbf{olb}})]$,
(lem3-eb) $\boldsymbol{K}^u_{\mathbf{eb}} = \bigcup_{i=1}^{k_{\mathbf{eb}}} [KN^i_{\mathbf{eb}} \cup \mathit{Desc}(t, KT^i_{\mathbf{eb}})]$.

Alors, on a : (†-lem3) $(\sigma_t \cdot \sigma_\omega, \omega) \sim (\Pi_{\boldsymbol{K}^u}(\sigma_t) \cdot \widehat{\sigma_\omega}, \widehat{\omega})$ où

- $(\sigma_\omega, \omega) = PUL_\Upsilon(t, u)$ sachant que Υ est un contexte dynamique fermé pour $VarLib(u)$ et consistant avec Γ,
- $(\widehat{\sigma_\omega}, \widehat{\omega}) = PUL_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^u}(t), u)$ sachant que $\widehat{\Upsilon}$ est un contexte dynamique fermé pour $VarLib(u)$ équivalent à Υ.

■

Il est trivial de noter que le Lemme 4 est un cas particulier du Lemme 4-bis correspondant à une mise à jour u telle que $VarLib(u) = \emptyset$.

Comme déjà annoncé en début de l'annexe, la preuve du Lemme 4 utilise les résultats de correction pour les requêtes (Lemme 2 pour l'évaluation des requêtes et Lemme 3 pour leur pré-évaluation). Ci-dessous, nous généralisons ces deux lemmes pour prendre en compte des requêtes non closes exactement dans le même esprit que pour le Lemme 4 et pour la preuve de celui-ci.

Lemme 3-bis (Correction de l'inférence des chemins pour la pré-évaluation des requêtes).
Soit une DTD D, un document $t \in D$ et une requête q.
Soit un environnement statique Γ fermé pour $VarLib(q)$.
Soit l'id-set $\mathbf{pre}\text{-}\boldsymbol{K}^q$ spécifié par :

$$\mathbf{pre}\text{-}\boldsymbol{K}^q = \bigcup_{\alpha \in \{\mathbf{nu},\mathbf{su},\mathbf{ebu},\mathbf{nr},\mathbf{sr}\}} \mathbf{pre}\text{-}\boldsymbol{K}^q_\alpha$$

où, on pose :

- pour $\alpha \in \{\mathbf{nu}, \mathbf{ebu}, \mathbf{nr}\}$, $QExt_\alpha(\Gamma, q) = \{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$
- pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$, $\{P^1_\alpha, .., P^{k_\alpha}_\alpha\} = \{P/\texttt{parent::node()} \mid P \in QExt_\alpha(\Gamma, q)\}$

et pour $\alpha \in \{\mathbf{nu}, \mathbf{su}, \mathbf{ebu}, \mathbf{nr}, \mathbf{sr}\}$ et $i = 1..k_\alpha$ on définit

- $\mathbf{pi}^i_\alpha$ le path-projecteur pour P^i_α et D
- $pre\text{-}T^i_\alpha = PEval(t, P^i_\alpha)$
- $pre\text{-}N^i_\alpha = KPath(t, \mathbf{pi}^i_\alpha)$

(lem2-n) pour $\alpha \in \{\mathbf{nu}, \mathbf{nr}\}$ $\mathbf{pre\text{-}K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [pre\text{-}N^i_\alpha]$

(lem2-str) pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$ $\mathbf{pre\text{-}K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [pre\text{-}N^i_\alpha \cup \mathit{Child}(t, pre\text{-}T^i_\alpha)]$

(lem2-ebu) $\mathbf{pre\text{-}K}^q_{\mathbf{ebu}} = \bigcup_{i=1}^{k_{\mathbf{ebu}}} [pre\text{-}N^i_{\mathbf{ebu}} \cup \mathit{Desc}(t, pre\text{-}T^i_{\mathbf{ebu}})]$

Alors si $\mathrm{I}_q \subseteq \mathit{dom}(\sigma_t)$ on a : $\mathrm{I}_q = \widehat{\mathrm{I}_q}$

- $\mathrm{I}_q = QEvalId_\Upsilon(t, q)$ sachant que Υ est un contexte dynamique fermé pour $VarLib(u)$ et consistant avec Γ,
- $\widehat{\mathrm{I}_q} = QEvalId_{\widehat{\Upsilon}}(\Pi_{\mathbf{pre\text{-}K}^q}(t), q)$, sachant que $\widehat{\Upsilon}$ est un contexte dynamique fermé pour $VarLib(u)$ et équivalent à Υ.

■

Lemme 2-bis (Correction généralisée de l'inférence des chemins pour l'évaluation des requêtes).
Soit une DTD D, un document $t \in D$ et une requête q.
Soit un environnement statique Γ fermé pour $VarLib(q)$.
Soit l'id-set $\mathbf{eval\text{-}K}^q$ spécifié par :

$$\mathbf{eval\text{-}K}^q = \bigcup_{\alpha \in \{\mathbf{nu},\mathbf{su},\mathbf{ebu},\mathbf{nr},\mathbf{sr}\}} \mathbf{eval\text{-}K}^q_\alpha$$

où, on pose :

- pour $\alpha \in \{\mathbf{nu}, \mathbf{ebu}, \mathbf{nr}\}$, $QExt_\alpha(\Gamma, q) = \{P^1_\alpha, .., P^{k_\alpha}_\alpha\}$
- pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$, $\{P^1_\alpha, .., P^{k_\alpha}_\alpha\} = \{P/\texttt{parent::node()} \mid P \in QExt_\alpha(\Gamma, q)\}$

et pour $\alpha \in \{\mathbf{nu}, \mathbf{su}, \mathbf{ebu}, \mathbf{nr}, \mathbf{sr}\}$ et $i = 1..k_\alpha$, on définit :

- $\mathbf{pi}^i_\alpha$ le path-projecteur pour P^i_α et D
- $ev\text{-}T^i_\alpha = PEval(t, P^i_\alpha)$
- $ev\text{-}N^i_\alpha = KPath(t, \mathbf{pi}^i_\alpha)$

(lem1-n) $\mathbf{eval\text{-}K}^q_{\mathbf{nu}} = \bigcup_{i=1}^{k_{\mathbf{nu}}} [ev\text{-}N^i_{\mathbf{nu}}]$

(lem1-str) pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$ $\mathbf{eval\text{-}K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [ev\text{-}N^i_\alpha \cup \mathit{Child}(t, ev\text{-}T^i_\alpha)]$

(lem1-ebu) pour $\alpha \in \{\mathbf{nr}, \mathbf{ebu}\}$, $\mathbf{eval\text{-}K}^q_\alpha = \bigcup_{i=1}^{k_\alpha} [ev\text{-}N^i_\alpha \cup \mathit{Desc}(t, ev\text{-}T^i_\alpha)]$

Alors on a : $(\sigma_t \cdot \sigma_q, \mathrm{I}_q) \simeq (\Pi_{\textbf{eval-}K^q}(t) \cdot \widehat{\sigma_q}, \widehat{\mathrm{I}_q})$ où

- $(\sigma_q, \mathrm{I}_q) = QEval_{\Upsilon}(t, q)$ sachant que Υ est un contexte dynamique fermé pour $VarLib(u)$ et consistant avec Γ,
- $(\widehat{\sigma_q}, \widehat{\mathrm{I}_q}) = QEval_{\widehat{\Upsilon}}(\Pi_{\textbf{eval-}K^q}(t), q)$, sachant que $\widehat{\Upsilon}$ est un contexte dynamique fermé pour $VarLib(u)$ et équivalent à Υ.

■

Remarque 1. Il est important de noter pour la suite que, si Γ est un environnement statique fermé pour la mise à jour u, alors c'est un environnement statique fermé pour toute sous-requête q de u. De manière similaire, si Υ est un contexte dynamique fermé pour la mise à jour u et consistant avec Γ alors c'est un contexte dynamique fermé pour toute sous-requête q de u et consistant avec Γ.

En préparation de la preuve du lemme 4-bis, nous allons nous intéresser maintenant aux requêtes **contextuelles** des mises à jour. Ces requêtes sont utilisées dans les mises à jour composées i.e **for** x `in` $q_{\texttt{ctxt}}$ **return** u, **let** x = $q_{\texttt{ctxt}}$ **return** u et **if** $q_{\texttt{ctxt}}$ **then** u_1 **else** u_2 pour assurer la navigation ou pour exprimer une condition. Ces requêtes occupent une place importante dans la pré-évaluation des mises à jour puisqu'elles sont à l'origine de l'évolution du contexte dynamique. Plus exactement, la pré-évaluation des mises à jour composées passe par la pré-évaluation des requêtes contextuelles dont le résultat est stocké dans le contexte dynamique.

Dans le Chapitre 2, nous avons introduits quelques notations concernant la syntaxe des mises à jour, notations que nous rappelons ici pour faciliter la lecture.

- $AtomUp$ désigne l'ensemble des mises à jour atomiques, i.e les mises à jour de la forme **delete** $q_{\texttt{cib}}$, **insert** $q_{\texttt{src}}$ δ $q_{\texttt{cib}}$, **replace** $q_{\texttt{cib}}$ `with` $q_{\texttt{src}}$ et **rename** $q_{\texttt{cib}}$ `as` a.
- $CompUp_1$ désigne des mises à jour d'itération et les mises à jour de binding i.e les mises à jour de la forme **for** x `in` $q_{\texttt{ctxt}}$ **return** u et **let** x = $q_{\texttt{ctxt}}$ **return** u, et
- $CompUp_2$ désigne les mises à jour conditionnelles **if** $q_{\texttt{ctxt}}$ **then** u_1 **else** u_2 et la composition de mises à jour u_1, u_2

Définition 31 (Requête contextuelle d'une mise à jour composée)**.**
L'ensemble des requêtes contextuelles d'une mise à jour u noté *CtxtQ*(u) est défini comme suit :

$$\textit{CtxtQ}(u) = \begin{cases} \emptyset & \text{si } u \in AtomUp \\ \{q_{\texttt{ctxt}}\} \cup \textit{CtxtQ}(u_1) & \text{si } u \in CompUp_1 \\ \textit{CtxtQ}(u_1) \cup \textit{CtxtQ}(u_2) & \text{si } u \in CompUp_2 \end{cases}$$

Le Lemme ci-dessous exprime que l'id-projecteur spécifié dans le lemme 4-bis est un projecteur correct pour la pré-évaluation des requêtes contextuelles de la mise à jour u. Formellement :

Lemme 5 (Correction de l'extraction des chemins pour la pré-évaluation des requêtes contextuelles)**.**

Sous les hypothèses du lemme 4-bis, pour toute requête $q \in$ *CtxtQ*(u) on a :

$$\mathrm{I}_q = \widehat{\mathrm{I}_q} \quad \text{avec} \tag{B.1}$$

- $\mathrm{I}_q = QEval_{\Upsilon}(t, q)$ sachant que Υ est un contexte dynamique fermé pour $VarLib(u)$ et consistant avec Γ (voir remarque 1),
- $\widehat{\mathrm{I}_q} = QEval_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^u}(t), q)$, sachant que $\widehat{\Upsilon}$ est un contexte dynamique fermé pour $VarLib(u)$ et équivalent à Υ (voir remarque 1).

■

Démonstration. Nous procédons par induction sur la taille de *CtxtQ*(u).

Cas de base : |*CtxtQ*(u)|=0. Immédiat puisque u ne contient pas de requête contextuelle.

Cas d'induction : Supposons maintenant la propriété (B.1) vraie pour toute mise à jour u telle que |*CtxtQ*(u)|$<n$. Montrons (B.1) pour u contenant n requêtes contextuelles. D'après la définition 31, |*CtxtQ*(u)|$=n$ correspond à l'un des cas suivants :

(i) $u \in \{\mathbf{for}\ x\ \mathtt{in}\ q_0\ \mathbf{return}\ u_1, \mathbf{let}\ x = q_0\ \mathbf{return}\ u_1\}$, avec |*CtxtQ*$(u_1)$|$=n-1$,

(ii) $u = \mathbf{if}\ q_0\ \mathbf{then}\ u_1\ \mathbf{else}\ u_2$, avec |*CtxtQ*$(u_1)$|+|*CtxtQ*$(u_2)$|$=n-1$

(iii) $u = u_1, u_2$ avec |*CtxtQ*(u_1)|+|*CtxtQ*(u_2)|$=n$.

Cas (i) $u \in \{\mathbf{for}\ x\ \mathtt{in}\ q_0\ \mathbf{return}\ u_1, \mathbf{let}\ x = q_0\ \mathbf{return}\ u_1\}$ avec |*CtxtQ*(u_1)|$=n-1$:
Par hypothèse d'induction, on sait que (B.1) est vérifié pour la mise à jour u_1 (avec $\boldsymbol{K}^{u_1}$). Pour démontrer (B.1) pour la mise à jour u, il faut montrer que $\boldsymbol{K}^u$ est correct pour la pré-évaluation de q_0 sur t. Pour cela, nous allons considérer l'id-projecteur $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}$ construit à partir des chemins extraits de q_0 comme défini dans le lemme 3-bis. Etant donné que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}$ est correct pour la pré-évaluation de q_0 sur t, il suffit de démontrer que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0} \subseteq \boldsymbol{K}^u$ pour déduire, par monotonie de la correction de la pré-évaluation des requêtes (propriété 8), que $\boldsymbol{K}^u$ est correct pour la pré-évaluation de q_0 sur t.

Pour montrer que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0} \subseteq \boldsymbol{K}^u$, nous analysons et comparons les constructions des deux id-projecteurs présentées dans les lemmes 3 et 4 version bis. Cette analyse est schématisée dans la Figure B.1 dont la partie centrale montre l'extraction des chemins pour la mise à jour u et la partie gauche (resp. droite) montre la construction de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}$ (resp. $\boldsymbol{K}^u$) à partir des chemins extraits de q_0 (resp. u). Les chemins qui interviennent dans la construction de chaque composant pour les deux id-projecteurs sont mis en évidence en utilisant des flèches en pointillés pour les chemins extraits de q_0 et des flèches en traits pleins pour ceux extraits de u. Noter pour certains chemins, l'ajout de l'étape `parent::node()` pour permettre l'accès au noeud dont tous les fils doivent être projetés. Dans ce cas, la flèche entre les chemins et les id-projecteurs concernés est étiquetée par **Par()**. Rappelons que l'étape `parent::node()` est ajoutée au moment de l'extraction des chemins pour les mises à jour tel que c'est indiqué par la règle (UPE:for:olb) donnée en Figure 4.21 page 104 d'où la présence de **Par()** dans la colonne du milieu dans la Figure B.1. L'étape `parent::node()` est ajoutée aux chemins extraits de q_0 au moment de la construction de l'id-projecteur $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}$ d'où l'indication de **Par()** dans la colonne de gauche. Enfin, nous indiquons en bas de la figure comment sont formés les chemins P^i_α à partir des id-sets KT^i_α et KN^i_α.

L'analyse de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}$ et de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}(u)$ montre qu'ils sont égaux, donc on peut écrire :

$\boldsymbol{K}^{u} = \mathbf{pre}\text{-}\boldsymbol{K}^{q_0} \cup \boldsymbol{K}^{u_1}$.

On en déduit que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0} \subseteq \boldsymbol{K}^{u}$.

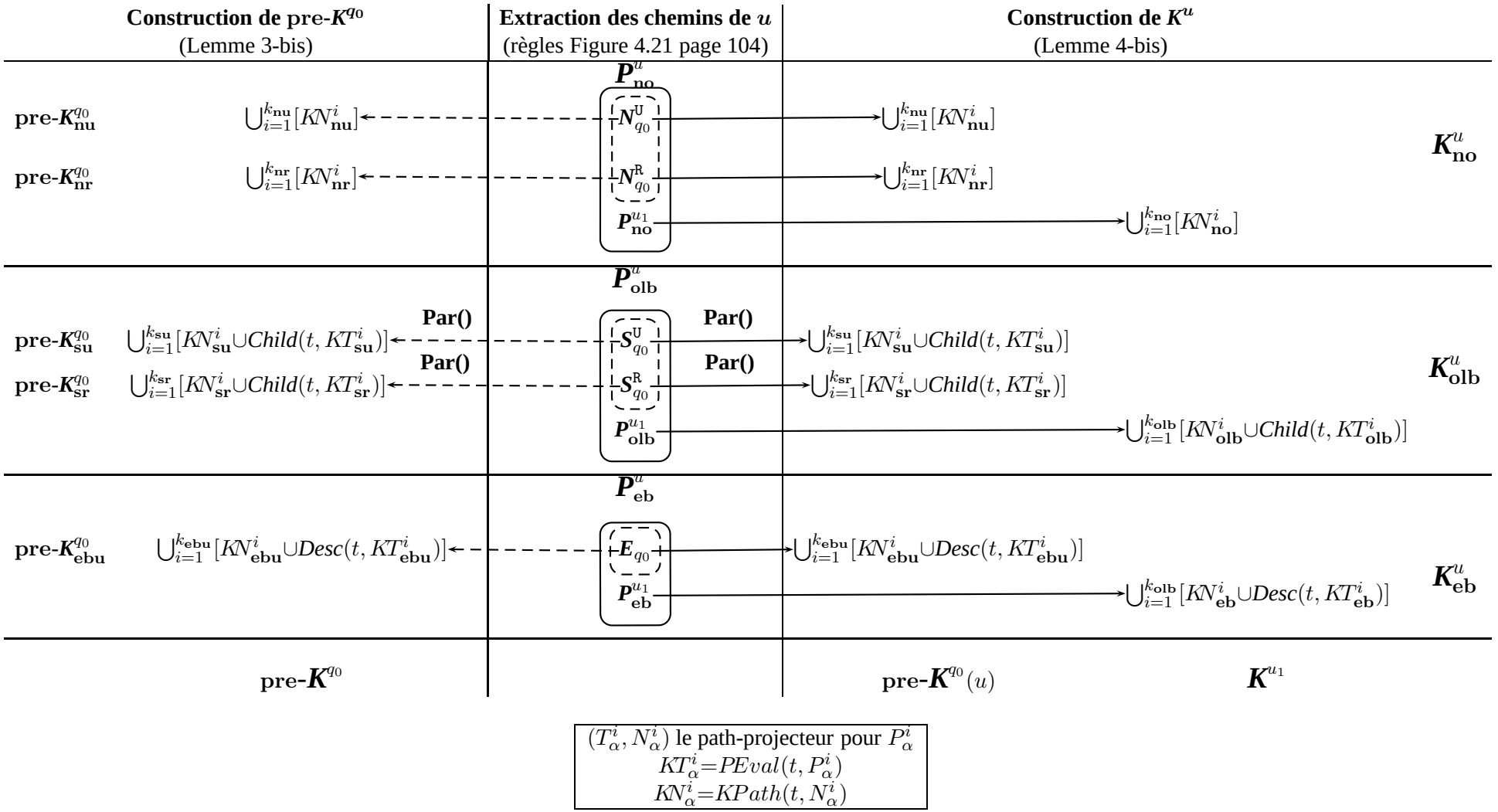


FIGURE B.1 – Illustration de la preuve du lemme 5 pour les mises à jour de la forme **for** x `in` q **return** u.

Cas (ii) u=**if** q_0 **then** u_1 **else** u_2 :
Dans ce cas $\boldsymbol{K}^u = \mathbf{pre}\text{-}\boldsymbol{K}^{q_0} \cup \boldsymbol{K}^{u_1} \cup \boldsymbol{K}^{u_2}$ où :
$-$ $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}$ est construit à partir de q_0 comme défini dans la lemme 3-bis, et
$-$ $\boldsymbol{K}^{u_1}$ et $\boldsymbol{K}^{u_2}$ sont construits à partir de u_1 et u_2 respectivement comme défini dans la lemme 4-bis. D'après l'hypothèse d'induction, pour toute requête $\boldsymbol{K}^{u_i}$ est correct pour la pré-évaluation de toute requête q∈*CtxtQ*(u_i). Comme $\boldsymbol{K}^{u_1} \cup \boldsymbol{K}^{u_2} \subseteq \boldsymbol{K}^u$ et d'après la monotonie de la correction des id-projecteurs pour la pré-évaluation des requêtes, on a que $\boldsymbol{K}^u$ est correct pour la pré-évaluation de toute requête q∈*CtxtQ*(u_i). Il reste à démontrer que $\boldsymbol{K}$ est correct pour la pré-évaluation de q_0 sur t. Pour cela, nous procédons comme dans le cas (i) en démontrant que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0} \subseteq \boldsymbol{K}^u$ et utilisons à nouveau la monotonie pour déduire que $\boldsymbol{K}^u$ est correct pour la pré-évaluation de q_0.

La démonstration de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0} \subseteq \boldsymbol{K}$ suit le même principe que celui utilisé pour le traitement du cas (i) et qui consiste à examiner la construction des id-projecteurs $\boldsymbol{K}^u$ et $\mathbf{pre}\text{-}\boldsymbol{K}^{q_0}$. Etant donné que l'extraction des chemins pour les mises à jour conditionnelles est analogue à l'extraction des chemins pour les mises à jour for et let, il n'est pas nécessaire de développer davantage la preuve qui se trouve être semblable au cas précédent.

Cas (iii) u=u_1, u_2
Dans ce cas $\boldsymbol{K}^u = \boldsymbol{K}^{u_1} \cup \boldsymbol{K}^{u_2}$ où $\boldsymbol{K}^{u_1}$ et $\boldsymbol{K}^{u_2}$ sont construits à partir u_1 et u_2 respectivement comme défini dans la lemme 4-bis. D'après l'hypothèse d'induction, $\boldsymbol{K}^{u_i}$ est correct pour la pré-évaluation des requêtes contextuelles de u_i sur t. La monotonie de la correction permet de conclure : $\boldsymbol{K}^u$ est correct pour toute requête contextuelle de u. □

B.1.1 Démonstration du Lemme 4-bis

Cas de base : $|u|$=1, i.e u est une mise à jour atomique (u∈$AtomUp$). Afin de faciliter la compréhension de la preuve, nous identifions trois groupes de mises à jour qui se traitent de la même manière parce qu'elles présentent des similarités du point de vue de la sémantique et de l'extraction des chemins. Il sera ainsi possible de démontrer $(\dagger\text{-lem3})$ pour une seule mise à jour atomique par groupe. Ces groupes sont comme suit :

(I) **delete** $q_{\texttt{cib}}$ et **rename** $q_{\texttt{cib}}$ **as** a
(II) **replace** $q_{\texttt{cib}}$ **with** $q_{\texttt{src}}$ et **insert** $[\,\delta\ \delta_{\texttt{sib}}]q_{\texttt{src}}q_{\texttt{cib}}$,
(III) **insert** $[\,\delta\ \delta_{\texttt{in}}]q_{\texttt{src}}q_{\texttt{cib}}$.

(I) Cas où u=**delete** $q_{\texttt{cib}}$:
La sémantique de cette mise à jour est capturée par la règle (**Up:delete**) que nous rappelons ci-dessous.

$$(\mathbf{Up{:}delete})\quad \frac{\sigma_t, \sigma_\epsilon, d, \gamma \models q_{\texttt{cib}} \Rightarrow \sigma_{\texttt{cib}}, \mathrm{I}_{\texttt{cib}} \quad |\mathrm{I}_{\texttt{cib}}| \leq 1 \quad \mathrm{I}_{\texttt{cib}} \subseteq \mathit{dom}(\sigma_t)}{\sigma_t, \sigma_\epsilon, d, \gamma \models \mathbf{delete}\ q_{\texttt{cib}} \Rightarrow \varnothing, \mathrm{del}(\mathrm{I}_{\texttt{cib}})}$$

D'après cette règle, $PUL_\Upsilon(t, \mathbf{delete}\ q_{\texttt{cib}}) = (\varnothing, \mathrm{del}(\mathbf{i}_{\texttt{cib}}))$ et $PUL_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^u}(t), \mathbf{delete}\ q_{\texttt{cib}}) = (\varnothing, \mathrm{del}(\widehat{\mathbf{i}_{\texttt{cib}}}))$.

D'après la définition 21 de l'équivalence des PULs, pour démontrer $(\dagger\text{-lem3})$ il faut démontrer que :

$\mathbf{i}_{\texttt{cib}} = \widehat{\mathbf{i}_{\texttt{cib}}}$.

Or, on sait d'après la définition 19 que :

$\mathbf{i}_{\texttt{cib}} = \widehat{\mathbf{i}_{\texttt{cib}}}$ ssi $(\dagger\dagger\text{-lem3})$ $\boldsymbol{K}^u$ est un id-projecteur correct pour la pré-évaluation de $q_{\texttt{cib}}$ relativement à t. Notre but est donc de démontrer $(\dagger\dagger\text{-lem3})$.

Pour démontrer $(\dagger\dagger\text{-lem3})$, nous suivons une méthode analogue à celle utilisée dans la preuve du Lemme 5 : nous considérons un id-projecteur $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}$ construit pour $q_{\texttt{cib}}$, t et D comme défini dans le lemme 3-bis et qui est donc correct pour la pré-évaluation de $q_{\texttt{cib}}$ relativement à t, nous démontrons que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}} \subseteq \boldsymbol{K}^u$ et en nous appuyant sur la propriété 8 de la monotonie de la correction des id-projecteurs pour la pré-évaluation des requêtes, nous déduisons $(\dagger\dagger\text{-lem3})$.

La démonstration de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}} \subseteq \boldsymbol{K}^u$ repose, bien évidemment, sur l'examen de la construction de chacun de ces id-projecteurs définie respectivement dans le Lemme 3-bis et le Lemme 4-bis. Cette construction se déroule en deux étapes :

> **Etape 1** : extraction des chemins à partir de $q_{\texttt{cib}}$ et de u et ajustement de ceux extraits de $q_{\texttt{cib}}$ afin qu'ils ciblent les noeuds devant servir pour la projection.
> Cet ajustement concerne uniquement les chemins extraits dans les catégories $\mathbf{su}$ et $\mathbf{sr}$ et consiste à étendre ces derniers par l'étape $\texttt{parent::node()}$. En posant pour $\alpha \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$ $\boldsymbol{P}_\alpha^{q_{\texttt{cib}}} = \mathit{QExt}_\alpha(\Gamma, q_{\texttt{cib}})$, et en notant $\mathbf{pre}\text{-}\boldsymbol{P}_\alpha^{q_{\texttt{cib}}}$ les ensembles des chemins après ajustement, alors
>
> – pour $\alpha \in \{\mathbf{nu}, \mathbf{ebu}, \mathbf{nr}\}$ $\mathbf{pre}\text{-}\boldsymbol{P}_\alpha^{q_{\texttt{cib}}} = \boldsymbol{P}_\alpha^{q_{\texttt{cib}}}$, et
>
> – pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$, $\mathbf{pre}\text{-}\boldsymbol{P}_\alpha^{q_{\texttt{cib}}} = Par(\boldsymbol{P}_\alpha^{q_{\texttt{cib}}})$.
>
> **Etape 2** : construction des id-projecteurs $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}$ et $\boldsymbol{K}^u$ à partir des chemins obtenus lors de la première étape. Cette construction consiste à calculer la trace de chaque chemin extrait lors de la première étape et de lui rajouter les identifiants des fils ou des descendants de ses cibles en fonction de sa catégorie. Cette étape est détaillée dans les Lemme 3-bis et Lemme 4-bis, nous ne la reprenons pas ici.

Comparer $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}$ et $\boldsymbol{K}^u$ nécessite donc d'examiner les résultats obtenus à l'issue de chacune de ces étapes.

Nous comparons d'abord les chemins extraits lors de l'étape 1. D'après les règles d'extraction des chemins pour u données en Figure 4.20, page 103, nous avons :

– $\boldsymbol{P}^u_{\mathbf{no}} = \boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{nu}} \cup \boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{nr}}$,

– $\boldsymbol{P}^u_{\mathbf{olb}} = Par(\boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{su}} \cup \boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{sr}})$, et

– $\boldsymbol{P}^u_{\mathbf{eb}} = \boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{ebu}}$,

donc, on a :

(I-Pno) $\boldsymbol{P}^u_{\mathbf{no}} = \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{nu}} \cup \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{nr}}$,

(I-Polb) $\boldsymbol{P}^u_{\mathbf{olb}} = \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{su}} \cup \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{sr}}$, et

(I-Peb) $\boldsymbol{P}^u_{\mathbf{eb}} = \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\texttt{cib}}}_{\mathbf{ebu}}$.

Nous procédons maintenant à l'examen des id-projecteurs $\boldsymbol{K}^u$ et $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{src}}}$ qui repose sur l'analyse du lien entre les chemins extraits pour u et ceux extraits pour $q_{\texttt{src}}$ et ajustés comme spécifié dans le Lemme 3-bis. Cette analyse synthétisée par les points (I-Pno), (I-Polb) et (I-Peb)

ci-dessus. L'examen de ces id-projecteurs repose aussi sur les hypothèses $(\text{lem2-n}), (\text{lem2-str})$ et (lem2-ebu) du Lemme 3-bis concernant la construction de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}$ ainsi que les hypothèses $(\text{lem3-no}), (\text{lem3-olb})$ et (lem3-eb) du lemme 4-bis concernant la construction de $\boldsymbol{K}^u$. Nous détaillons l'examen de chaque sous-enesemble $\boldsymbol{K}^u_\alpha$ dans ce qui suit.

Examen de $\boldsymbol{K}^u_{\mathbf{no}}$

D'après l'hypothèse (lem3-no) de l'énoncé du Lemme 4 donné en page 105, on a :

$\boldsymbol{K}^u_{\mathbf{no}} = \bigcup_{i=1}^{k_{\mathbf{no}}} [KN^i_{\mathbf{no}}]$.

Or d'après le point (I-Pno), on sait que :

$\boldsymbol{P}^u_{\mathbf{no}} = \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\text{cib}}}_{\mathbf{nu}} \cup \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\text{cib}}}_{\mathbf{nr}}$.

Par ailleurs, d'après lemme 3-bis, on sait que $pre\text{-}N^i_{\mathbf{nr}}$ resp. $pre\text{-}N^i_{\mathbf{nr}}$ sont obtenus à partir de $\mathbf{pre}\text{-}\boldsymbol{P}^{q_{\text{cib}}}_{\mathbf{nu}}$ resp. $\mathbf{pre}\text{-}\boldsymbol{P}^{q_{\text{cib}}}_{\mathbf{nr}}$. On en déduit que :

$\bigcup_{i=1}^{k_{\mathbf{no}}} [KN^i_{\mathbf{no}}] = \bigcup_{i=1}^{k_{\mathbf{nu}}} [pre\text{-}N^i_{\mathbf{nu}}] \cup \bigcup_{i=1}^{k_{\mathbf{nr}}} [pre\text{-}N^i_{\mathbf{nr}}]$,

D'après l'hypothèse (lem2-n) de l'énoncé de ce même Lemme, on a :

$\bigcup_{i=1}^{k_{\mathbf{nu}}} [pre\text{-}N^i_{\mathbf{nu}}] = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{nu}}$ et $\bigcup_{i=1}^{k_{\mathbf{nr}}} [pre\text{-}N^i_{\mathbf{nr}}] = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{nr}}$.

Donc,

$\boldsymbol{K}^u_{\mathbf{no}} = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{nu}} \cup \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{nr}}$. (R1-a)

Examen de $\boldsymbol{K}^u_{\mathbf{olb}}$

Nous procédons de la même manière que pour le cas précédent. Cette fois, on utilise l'hypothèse (lem3-olb) qui nous donne :

$\boldsymbol{K}^u_{\mathbf{olb}} = \bigcup_{i=1}^{k_{\mathbf{olb}}} [KN^i_{\mathbf{olb}} \cup \mathit{Child}(t, KT^i_{\mathbf{olb}})]$.

D'après le point (I-Polb), on a :

$\bigcup_{i=1}^{k_{\mathbf{olb}}} [KN^i_{\mathbf{olb}}] = \bigcup_{i=1}^{k_{\mathbf{su}}} [pre\text{-}N^i_{\mathbf{su}}] \cup \bigcup_{i=1}^{k_{\mathbf{sr}}} [pre\text{-}N^i_{\mathbf{sr}}]$ et

$\bigcup_{i=1}^{k_{\mathbf{olb}}} [\mathit{Child}(t, KT^i_{\mathbf{olb}})] = \bigcup_{i=1}^{k_{\mathbf{su}}} [\mathit{Child}(t, pre\text{-}T^i_{\mathbf{su}})] \cup \bigcup_{i=1}^{k_{\mathbf{sr}}} [\mathit{Child}(t, pre\text{-}T^i_{\mathbf{sr}})]$.

Or, d'après le point (lem2-str) du Lemme 3-bis, on sait que :

$\bigcup_{i=1}^{k_{\mathbf{su}}} [pre\text{-}N^i_{\mathbf{su}} \cup \mathit{Child}(t, pre\text{-}T^i_{\mathbf{su}})] = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{su}}$ et

$\bigcup_{i=1}^{k_{\mathbf{sr}}} [pre\text{-}N^i_{\mathbf{sr}} \cup \mathit{Child}(t, pre\text{-}T^i_{\mathbf{sr}})] = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{sr}}$.

Donc,

$\boldsymbol{K}^u_{\mathbf{olb}} = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{su}} \cup \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\text{cib}}}_{\mathbf{sr}}$. (R1-b)

Examen de $\boldsymbol{K}^u_{\mathbf{eb}}$

Pour ce cas, on utilise l'hypothèse (lem3-eb) qui nous donne :

$\boldsymbol{K}^u_{\mathbf{eb}} = \bigcup_{i=1}^{k_{\mathbf{eb}}} [KN^i_{\mathbf{eb}} \cup \mathit{Desc}(t, KT^i_{\mathbf{eb}})]$.

D'après le point (I-Peb), on a :

$\bigcup_{i=1}^{k_{\mathbf{eb}}} [KN^i_{\mathbf{eb}} \cup \mathit{Desc}(t, KT^i_{\mathbf{eb}})] = \bigcup_{i=1}^{k_{\mathbf{ebu}}} [pre\text{-}N^i_{\mathbf{ebu}} \cup \mathit{Desc}(t, pre\text{-}T^i_{\mathbf{ebu}})]$

D'après le point $(\mathrm{lem2\text{-}ebu})$, on déduit que :

$$\boldsymbol{K}^{u}_{\mathbf{eb}} = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\mathtt{cib}}}_{\mathbf{ebu}}. \qquad \text{(R1-c)}$$

D'après (R1-a), (R1-b) et (R1-c) il est facile de constater que :

$\bigcup_{\beta\in\{\mathbf{no},\mathbf{olb},\mathbf{eb}\}} \boldsymbol{K}^{u}_{\beta} = \bigcup_{\alpha\in\{\mathbf{nu},\mathbf{su},\mathbf{ebu},\mathbf{nr},\mathbf{sr}\}} \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\mathtt{cib}}}_{\alpha}$.

On en déduit alors $(\dagger\dagger\text{-}\mathrm{lem3})$.

(II) Cas où u=**insert** $q_{\mathtt{src}}$ δ $q_{\mathtt{cib}}$ avec $\delta\in\{\leftarrow,\rightarrow\}$
La sémantique de cette mise à jour est capturée par la règle $(\mathbf{Up{:}insert})$ que nous rappelons ci-dessous.

$$(\mathbf{Up{:}insert})\ \frac{\sigma_t,\sigma_\epsilon,d,\gamma\models q_{\mathtt{src}} \overset{\mathsf{copie}}{\Longrightarrow} \sigma_{\mathtt{src}},\mathrm{I}_{\mathtt{src}} \quad \sigma_t,\sigma_\epsilon,d,\gamma\models q_{\mathtt{cib}}\Rightarrow\sigma_{\mathtt{cib}},\mathrm{I}_{\mathtt{cib}} \quad |\mathrm{I}_{\mathtt{cib}}|\leq 1 \quad \mathrm{I}_{\mathtt{cib}}\subseteq \mathit{dom}(\sigma_t)}{\sigma_t,\sigma_\epsilon,d,\gamma\models \mathbf{insert}\ q_{\mathtt{src}}\ \delta\ q_{\mathtt{cib}} \Rightarrow \sigma_{\mathtt{src}},\mathrm{ins}(\mathrm{I}_{\mathtt{src}},\delta,\mathrm{I}_{\mathtt{cib}})}$$

D'après cette règle, on a :

- $\mathit{PUL}_{\Upsilon}(t, \mathbf{insert}\ q_{\mathtt{src}}\ \delta\ q_{\mathtt{cib}}) = (\sigma_{\mathtt{src}}, \mathrm{ins}(\mathrm{I}_{\mathtt{src}},\delta,\mathbf{i}_{\mathtt{cib}}))$
- $\mathit{PUL}_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^u}(t), \mathbf{insert}\ q_{\mathtt{src}}\ \delta\ q_{\mathtt{cib}}) = (\widehat{\sigma_{\mathtt{src}}}, \mathrm{ins}(\widehat{\mathrm{I}_{\mathtt{src}}},\delta,\widehat{\mathbf{i}_{\mathtt{cib}}}))$;

où,

$\mathbf{i}_{\mathtt{cib}} = QEvalId_{\Upsilon}(t, q_{\mathtt{cib}})$ (resp. $\widehat{\mathbf{i}_{\mathtt{cib}}} = QEvalId_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^u}(t), q_{\mathtt{cib}})$) est obtenu de la pré-évaluation de la requête cible $q_{\mathtt{cib}}$ sur t (resp. sur $\Pi_{\boldsymbol{K}^u}(t)$) , et

$(\sigma_{\mathtt{src}},\mathrm{I}_{\mathtt{src}}) = QCopy_{\Upsilon}(t, q_{\mathtt{src}})$ (resp. $(\widehat{\sigma_{\mathtt{src}}},\widehat{\mathrm{I}_{\mathtt{src}}}) = QCopy_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^u}(t), q_{\mathtt{src}})$) est obtenu de la matérialisation des réponses de la requête source $q_{\mathtt{src}}$ à partir de t (resp. sur $\Pi_{\boldsymbol{K}^u}(t)$).

D'après la définition 21 de l'équivalence des PULs, pour démontrer $(\dagger\text{-}\mathrm{lem3})$ il faut démontrer que :

$\mathbf{i}_{\mathtt{cib}} = \widehat{\mathbf{i}_{\mathtt{cib}}}$ et $(\sigma_{\mathtt{src}},\mathrm{I}_{\mathtt{src}}) \simeq (\widehat{\sigma_{\mathtt{src}}},\widehat{\mathrm{I}_{\mathtt{src}}})$.

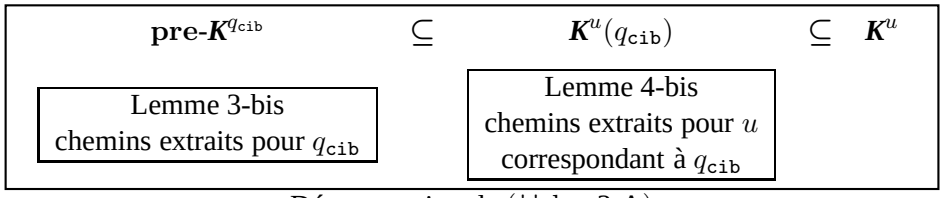
Or, on sait d'après la définition 19 que :

$\mathbf{i}_{\mathtt{cib}} = \widehat{\mathbf{i}_{\mathtt{cib}}}$ ssi $(\dagger\dagger\text{-}\mathrm{lem3\text{-}A})$ $\boldsymbol{K}^u$ est un id-projecteur correct pour la pré-évaluation de $q_{\mathtt{cib}}$ relativement à t, et
d'après la définition 19 que :

$(\sigma_{\mathtt{src}},\mathrm{I}_{\mathtt{src}}) \simeq (\widehat{\sigma_{\mathtt{src}}},\widehat{\mathrm{I}_{\mathtt{src}}})$ ssi $(\dagger\dagger\text{-}\mathrm{lem3\text{-}B})$ $\boldsymbol{K}^u$ est un id-projecteur correct pour la matérialisation des réponses de $q_{\mathtt{src}}$ relativement à t.

Pour la démonstration de $(\dagger\text{-}\mathrm{lem3})$, nous considérons que $\boldsymbol{K}^u$ s'écrit comme l'union de :
$-$ $\boldsymbol{K}^u(q_{\mathtt{cib}})$ construit comme défini dans Lemme 4 pour les chemins extraits pour u et correspondant à $q_{\mathtt{cib}}$;
$-$ $\boldsymbol{K}^u(q_{\mathtt{src}})$ construit comme défini dans Lemme 4 pour les chemins extraits pour u et correspondant à $q_{\mathtt{src}}$,
étant donné que $\boldsymbol{K}^u$ est construit à partir des chemins extraits de $q_{\mathtt{cib}}$ ou $q_{\mathtt{src}}$.

La démonstration de $(\dagger\dagger\text{-}\mathrm{lem3\text{-}A})$ est illustrée ci-dessous.



Démonstration de $(\dagger\dagger\text{-lem3-A})$

Elle consiste à considérer id-projecteur $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}$ construit pour $q_{\texttt{cib}}$ comme défini dans le Lemme 3 et qui est donc correct pour la pré-évaluation de $q_{\texttt{cib}}$ relativement à t, et à démontrer que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}} \subseteq \boldsymbol{K}^u(q_{\texttt{cib}})$. En nous appuyant sur la propriété 8 de monotonie de la correction des id-projecteurs pour la pré-évaluation des requêtes, nous déduisons $(\dagger\dagger\text{-lem3-A})$.

La démonstration de $(\dagger\dagger\text{-lem3-B})$ suit le même principe. Dans ce cas, nous considérons l'id-projecteur $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}}$ construit pour $q_{\texttt{src}}$ comme défini dans Lemme 2-bis et qui est donc correct pour l'évaluation de $q_{\texttt{src}}$ sur t. D'après la propriété 9, $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}}$ est correct pour la matérialisation des réponses de $q_{\texttt{src}}$ relativement à t et D. Donc, il nous suffit de montrer que $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}} \subseteq \boldsymbol{K}^u(q_{\texttt{src}})$ pour déduire $(\dagger\dagger\text{-lem3-B})$.

Dans la suite de la preuve, nous notons, pour $\alpha \in \{\mathbf{nu}, \mathbf{su}, \mathbf{ebu}, \mathbf{nr}, \mathbf{sr}\}$, $\beta \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$ et pour $q_x \in \{q_{\texttt{cib}}, q_{\texttt{src}}\}$:

- $\boldsymbol{P}_\alpha^{q_x}$ l'ensemble des chemins de la catégorie α extraits de q_x i.e $\boldsymbol{P}_\alpha^{q_x} = QExt_\alpha(\Gamma, q_x)$,
- $\boldsymbol{P}_\beta^u$ l'ensemble des chemins de la catégorie β extraits de u i.e $\boldsymbol{P}_\alpha^u = UExt_\beta(\Gamma, u)$,
- $\boldsymbol{P}_\beta^u(q_x)$ l'ensemble des chemins de la catégorie β extraits de u et correspondant à la requête q_x.

Preuve de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}} \subseteq \boldsymbol{K}^u(q_{\texttt{cib}})$

Cette démonstration suit le principe utilisé dans la preuve du cas $u = \mathbf{delete}\ q_{\texttt{cib}}$. Le but des d'écrire $\boldsymbol{K}^u(q_{\texttt{cib}})$ en fonction de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}$.

Posons $\mathbf{pre}\text{-}\boldsymbol{P}_\alpha^{q_{\texttt{cib}}}$ l'ensemble des chemins de la catégorie α obtenus de $q_{\texttt{cib}}$ après l'ajustement effectué dans Lemme 3-bis (voir traitement du cas précédent).

D'après les règles d'extraction des chemins pour u données en Figure 4.19, page 103, nous avons :

- $\boldsymbol{P}_{\mathbf{no}}^u(q_{\texttt{cib}}) = \boldsymbol{P}_{\mathbf{nu}}^{q_{\texttt{cib}}}$,
- $\boldsymbol{P}_{\mathbf{olb}}^u(q_{\texttt{cib}}) = Par(\boldsymbol{P}_{\mathbf{su}}^{q_{\texttt{cib}}} \cup \boldsymbol{P}_{\mathbf{sr}}^{q_{\texttt{cib}}} \cup \boldsymbol{P}_{\mathbf{nr}}^{q_{\texttt{cib}}})$, et
- $\boldsymbol{P}_{\mathbf{eb}}^u(q_{\texttt{cib}}) = \boldsymbol{P}_{\mathbf{ebu}}^{q_{\texttt{cib}}}$,

donc, on a :

$(\text{II-Pno-}\texttt{cib})$ $\boldsymbol{P}_{\mathbf{no}}^u(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{P}_{\mathbf{nu}}^{q_{\texttt{cib}}}$,

$(\text{II-Polb-}\texttt{cib})$ $\boldsymbol{P}_{\mathbf{olb}}^u(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{P}_{\mathbf{su}}^{q_{\texttt{cib}}} \cup \mathbf{pre}\text{-}\boldsymbol{P}_{\mathbf{sr}}^{q_{\texttt{cib}}} \cup Par(\mathbf{pre}\text{-}\boldsymbol{P}_{\mathbf{nr}}^{q_{\texttt{cib}}})$, et

$(\text{II-Peb-}\texttt{cib})$ $\boldsymbol{P}_{\mathbf{eb}}^u(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{P}_{\mathbf{ebu}}^{q_{\texttt{cib}}}$.

Nous constatons d'après le point (II-Polb) ci-dessus que les chemins $\mathbf{pre}\text{-}\boldsymbol{P}_{\mathbf{nr}}^{q_{\texttt{cib}}}$ utilisés pour générer $\boldsymbol{K}_{\mathbf{olb}}^u(q_{\texttt{cib}})$ sont d'abord étendus avec l'étape `parent::node()` du fait de l'extraction des chemins pour le cas $u = \mathbf{insert}\ q_{\texttt{src}}\ \delta\ q_{\texttt{cib}}$ pour $\delta \in \{\leftarrow, \rightarrow\}$. Comme notre but est d'écrire chaque

id-projecteur $\boldsymbol{K}^u_\beta(q_{\texttt{cib}})$ en fonction de certains id-projecteurs $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_\alpha$, notons pour $P^i_{\mathbf{nr}}$ extrait à partir de $q_{\texttt{cib}}$, $par\text{-}pre\text{-}T^i_{\mathbf{nr}}$ et $par\text{-}pre\text{-}N^i_{\mathbf{nr}}$ les id-sets générés à partir de $P^i_{\mathbf{nr}}/\texttt{parent::node}()$ comme défini dans le Lemme 3-bis.

Nous procédons maintenant à la comparaison de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}$ et $\boldsymbol{K}^u(q_{\texttt{cib}})$ qui consiste à développer chaque $\boldsymbol{K}^u_\beta(q_{\texttt{cib}})$ en fonction des points (II-Pno-`cib`), (II-Polb-`cib`) et (II-Peb-`cib`).

Examen de $\boldsymbol{K}^u_{\mathbf{no}}(q_{\texttt{cib}})$

D'après l'hypothèse (lem3-no) on a :

$\boldsymbol{K}^u_{\mathbf{no}}(q_{\texttt{cib}}) = \bigcup_{i=1}^{k_{\mathbf{no}}}[KN^i_{\mathbf{no}}]$.

Or, d'après le point (II-Pno), on a :

$\bigcup_{i=1}^{k_{\mathbf{no}}}[KN^i_{\mathbf{no}}] = \bigcup_{i=1}^{k_{\mathbf{nu}}}[pre\text{-}N^i_{\mathbf{nu}}]$.

Enfin, d'après l'hypothèse (lem2-n) du lemme 3-bis, on conclut que :

$$\boldsymbol{K}^u_{\mathbf{no}}(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nu}}. \tag{R2-a}$$

Examen de $\boldsymbol{K}^u_{\mathbf{olb}}(q_{\texttt{cib}})$

D'après l'hypothèse (lem3-olb) :

$\boldsymbol{K}^u_{\mathbf{olb}}(q_{\texttt{cib}}) = \bigcup_{i=1}^{k_{\mathbf{olb}}}[KN^i_{\mathbf{olb}} \cup \mathit{Child}(t, KT^i_{\mathbf{olb}})]$

Or, d'après le point (II-Polb), on a :

$\bigcup_{i=1}^{k_{\mathbf{olb}}}[pre\text{-}N^i_{\mathbf{olb}}] = \bigcup_{i=1}^{k_{\mathbf{su}}}[pre\text{-}N^i_{\mathbf{su}}] \cup \bigcup_{i=1}^{k_{\mathbf{sr}}}[pre\text{-}N^i_{\mathbf{sr}}] \cup \bigcup_{i=1}^{k_{\mathbf{nr}}}[par\text{-}pre\text{-}N^i_{\mathbf{nr}}]$

$$\bigcup_{i=1}^{k_{\mathbf{olb}}}[\mathit{Child}(t, pre\text{-}T^i_{\mathbf{olb}})] = \bigcup_{i=1}^{k_{\mathbf{su}}}[\mathit{Child}(t, pre\text{-}T^i_{\mathbf{su}})] \cup \bigcup_{i=1}^{k_{\mathbf{sr}}}[\mathit{Child}(t, pre\text{-}T^i_{\mathbf{sr}})] \cup \bigcup_{i=1}^{k_{\mathbf{nr}}}[\mathit{Child}(t, par\text{-}pre\text{-}T^i_{\mathbf{nr}})]$$

En posant, on a :

– $par\text{-}\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} = \bigcup_{i=1}^{k_{\mathbf{nr}}}[par\text{-}pre\text{-}N^i_{\mathbf{nr}} \cup \mathit{Child}(t, par\text{-}pre\text{-}T^i_{\mathbf{nr}})]$

Donc, d'après l'hypothèse (lem2-str), on conclut que :

$$\boldsymbol{K}^u_{\mathbf{olb}}(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{su}} \cup \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{sr}} \cup par\text{-}\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} \tag{R2-b}$$

Examen de $\boldsymbol{K}^u_{\mathbf{eb}}(q_{\texttt{cib}})$

Pour ce cas, d'après l'hypothèse (lem3-eb) on a :

$\boldsymbol{K}^u_{\mathbf{eb}}(q_{\texttt{cib}}) = \bigcup_{i=1}^{k_{\mathbf{eb}}}[KN^i_{\mathbf{eb}} \cup \mathit{Desc}(t, KT^i_{\mathbf{eb}})]$

D'après (II-Peb), on a :

$\bigcup_{i=1}^{k_{\mathbf{eb}}}[KN^i_{\mathbf{eb}} \cup \mathit{Desc}(t, KT^i_{\mathbf{eb}})] = \bigcup_{i=1}^{k_{\mathbf{ebu}}}[pre\text{-}N^i_{\mathbf{ebu}} \cup \mathit{Desc}(t, pre\text{-}T^i_{\mathbf{ebu}})]$.

On en déduit que :

$$\boldsymbol{K}^u_{\mathbf{eb}}(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{ebu}}. \tag{R2-c}$$

D'après (R2-a), (R2-b) et (R2-c), on a :
$\boldsymbol{K}^u(q_{\texttt{cib}}) = \bigcup_{\alpha\in\{\mathbf{nu},\mathbf{su},\mathbf{ebu},\mathbf{sr}\}} \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_\alpha \cup par\text{-}\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}}$.

Donc, afin de démontrer :

$\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}} \subseteq \boldsymbol{K}^u(q_{\texttt{cib}})$,
il suffit de démontrer que :
$\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} \subseteq par\text{-}\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}}$.

Preuve de $(\ddagger)$ $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} \subseteq par\text{-}\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}}$

Rappelons que d'après l'hypothèse $(\mathrm{lem2\text{-}n})$ du Lemme 3-bis :

$\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} = \bigcup_{i=1}^{k_{\mathbf{nr}}} [pre\text{-}N^i_{\mathbf{nr}}]$.

Rappelons également que :

$par\text{-}\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} = \bigcup_{i=1}^{k_{\mathbf{nr}}} [par\text{-}pre\text{-}N^i_{\mathbf{nr}} \cup \mathit{Child}(t, par\text{-}pre\text{-}T^i_{\mathbf{nr}})]$.

Pour démontrer $(\ddagger)$, il faut donc démontrer que pour $i = 1 \ldots k_{\mathbf{nr}}$:

$pre\text{-}N^i_{\mathbf{nr}} \subseteq par\text{-}pre\text{-}N^i_{\mathbf{nr}} \cup \mathit{Child}(t, par\text{-}pre\text{-}T^i_{\mathbf{nr}})$.

Cette preuve repose sur le résultat de [BCCN06] énoncé comme suit. Etant donné P un chemin, (T, N) le path-projecteur de P, $(par\text{-}T, par\text{-}N)$ le path-projecteur de $P/\texttt{parent::node()}$ alors : $N \subseteq par\text{-}N$.
Ce résultat est exploité dans notre preuve puisque $pre\text{-}N^i_{\mathbf{nr}}$ resp. $par\text{-}pre\text{-}N^i_{\mathbf{nr}}$ est construit à partir de $N^i_{\mathbf{nr}}$ inféré pour $P^i_{\mathbf{nr}}$ resp. $par\text{-}N^i_{\mathbf{nr}}$ inféré pour $P^i_{\mathbf{nr}}/\texttt{parent::node()}$ en utilisant la fonction $KPath$ qui est monotone. Cela nous permet de déduire $(\ddagger)$.

Preuve de $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}} \subseteq \boldsymbol{K}^u(q_{\texttt{src}})$

La preuve de ce cas reprend la méthode utilisée pour la preuve des deux cas précédents. Dans ce cas, nous devons écrire $\boldsymbol{K}^u(q_{\texttt{src}})$ en fonction de $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}}$.

Posons $\boldsymbol{P}^{q_{\texttt{src}}}_\alpha$ l'ensemble des chemins de la catégorie α obtenus de $q_{\texttt{src}}$ i.e $\boldsymbol{P}^{q_{\texttt{src}}}_\alpha = QExt_\alpha(\Gamma, q_{\texttt{src}})$. Notons $\mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_\alpha$ l'ensemble des chemins de la catégorie α obtenus de $q_{\texttt{src}}$ après l'ajustement effectué dans Lemme 2-bis que nous rappelons ci-dessous :

- pour $\alpha \in \{\mathbf{nu}, \mathbf{ebu}, \mathbf{nr}\}$, $\mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_\alpha = \boldsymbol{P}^{q_{\texttt{src}}}_\alpha$, et
- pour $\alpha \in \{\mathbf{su}, \mathbf{sr}\}$, $\mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_\alpha = Par(\boldsymbol{P}^{q_{\texttt{src}}}_\alpha)$.

D'après les règles d'extraction des chemins pour u données en Figure 4.19, page 103, nous avons :

- $\boldsymbol{P}^u_{\mathbf{no}}(q_{\texttt{src}}) = \boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{nu}}$,
- $\boldsymbol{P}^u_{\mathbf{olb}}(q_{\texttt{src}}) = Par(\boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{su}} \cup \boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{sr}})$, et
- $\boldsymbol{P}^u_{\mathbf{eb}}(q_{\texttt{src}}) = \boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{ebu}} \cup \boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{nr}}$,

donc, on a :

$(\text{II-Pno-}\texttt{src})$ $\boldsymbol{P}^u_{\mathbf{no}}(q_{\texttt{src}}) = \mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{nu}}$,

$(\text{II-Polb-}\texttt{src})$ $\boldsymbol{P}^u_{\mathbf{olb}}(q_{\texttt{src}}) = \mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{su}} \cup \mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{sr}}$, et

$(\text{II-Peb-}\texttt{src})$ $\boldsymbol{P}^u_{\mathbf{eb}}(q_{\texttt{src}}) = \mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{ebu}} \cup \mathbf{eval}\text{-}\boldsymbol{P}^{q_{\texttt{src}}}_{\mathbf{nr}}$.

L'étape suivante consiste à comparer les id-sets $\boldsymbol{K}^u(q_{\texttt{src}})$ et $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}}$ en se basant d'abord sur les points (III-Pno), (III-Polb) et (III-Peb) qui exhibent le lien entre les chemins extraits pour u et correspondant à $q_{\texttt{src}}$ et des chemins extraits de $q_{\texttt{src}}$ et ajustés tel que spécifié dans le Lemme 2-bis. L'analyse des id-sets $\boldsymbol{K}^u(q_{\texttt{src}})$ et $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}}$ se base aussi sur les hypothèses $(\mathrm{lem3\text{-}no})$, $(\mathrm{lem3\text{-}olb})$

et (lem3-eb) du Lemme 4-bis spécifiant la construction de $\boldsymbol{K}^u(q_{\mathtt{src}})$ et des hypothèses (lem1-n), (lem1-str) et (lem1-ebu) du Lemme 2-bis spécifiant la construction de $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}$. Cette analyse conclut que :

$$\boldsymbol{K}^u_{\mathbf{no}}(q_{\mathtt{src}}) = \mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}_{\mathbf{nu}}. \tag{R3-a}$$

$$\boldsymbol{K}^u_{\mathbf{olb}}(q_{\mathtt{src}}) = \mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}_{\mathbf{su}} \cup \mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}_{\mathbf{sr}}. \tag{R3-b}$$

$$\boldsymbol{K}^u_{\mathbf{eb}}(q_{\mathtt{src}}) = \mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}_{\mathbf{ebu}} \cup \mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}_{\mathbf{nr}}. \tag{R3-c}$$

et permet de déduire d'après (R3-a), (R3-b) et (R3-c) que $\boldsymbol{K}^u(q_{\mathtt{src}}) = \mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}$.

(III) Cas où u=**insert** $q_{\mathtt{src}}$ δ $q_{\mathtt{cib}}$ avec $\delta \in \{\swarrow, \downarrow, \searrow\}$
La sémantique de cette mise à jour est la même que pour le cas où $\delta \in \{\leftarrow, \rightarrow\}$. Par conséquent, la preuve de (†-lem3) pour ce cas est analogue à la preuve de (†-lem3) pour le cas précédent et consiste à démonter que :

(††-lem3-C) $\boldsymbol{K}^u$ est correct pour la pré-évaluation de $q_{\mathtt{src}}$ relativement à t, et

(††-lem3-D) $\boldsymbol{K}^u$ est correct pour la matérialisation des réponses de $q_{\mathtt{src}}$ relativement à t.

Nous procédons comme pour la preuve du cas précédent en considérons $\boldsymbol{K}^u = \boldsymbol{K}^u(q_{\mathtt{cib}}) \cup \boldsymbol{K}^u(q_{\mathtt{src}})$ où $\boldsymbol{K}^u(q_{\mathtt{cib}})$ (resp. $\boldsymbol{K}^u(q_{\mathtt{src}})$) est construit tel que défini dans Lemme **??**-bis à partir des chemins extraits de u et correspondant à $q_{\mathtt{cib}}$ (resp. $q_{\mathtt{src}}$).

Pour démontrer (††-lem3-C), nous considérons l'id-projecteur $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\mathtt{cib}}}$ construit pour $q_{\mathtt{cib}}$ comme défini dans Lemme 3-bis et démontrons que $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\mathtt{cib}}} \subseteq \boldsymbol{K}^u(q_{\mathtt{cib}})$.

Pour démontrer (††-lem3-D), nous procédons de manière analogue en considérant l'id-projecteur $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}}$ construit, cette fois, pour la requête $q_{\mathtt{src}}$ comme défini dans Lemme **??**-bis, nous démontrons que $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\mathtt{src}}} \subseteq \boldsymbol{K}^u(q_{\mathtt{src}})$ pour déduire (††-lem3-D).

Preuve de $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\mathtt{cib}}} \subseteq \boldsymbol{K}^u(q_{\mathtt{cib}})$

Le traitement de ce cas est bien évidemment similaire au traitement du cas (II) puisque la sémantique des deux mises à jour est identique. La seule différence réside dans l'extraction des chemins de la catégorie **olb** qui est définie de manière différente pour le cas où $\delta \in \{\swarrow, \downarrow, \searrow\}$, qui est la règle ($\mathbf{UPE}$:$\mathbf{ins}$:$\mathbf{olb-bis}$) de la Figure 4.19. Nous procédons de la même manière que pour le cas (II) en identifiant d'abord le lien entre les chemins $\boldsymbol{P}(q_{\mathtt{cib}})$ extraits pour u et correspondant à $q_{\mathtt{cib}}$ et les chemins $\mathbf{pre}\text{-}\boldsymbol{P}^{q_{\mathtt{cib}}}$ obtenus de $q_{\mathtt{cib}}$ après ajustements. D'après les règles de la Figure 4.19 donnée en page 103, on a :

- $\boldsymbol{P}^u_{\mathbf{no}}(q_{\mathtt{cib}}) = \boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{nu}}$,
- $\boldsymbol{P}^u_{\mathbf{olb}}(q_{\mathtt{cib}}) = Par(\boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{su}} \cup \boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{sr}}) \cup \boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{nr}}$, et
- $\boldsymbol{P}^u_{\mathbf{eb}}(q_{\mathtt{cib}}) = \boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{ebu}}$.

D'après l'hypothèse du Lemme 3-bis, on déduit que :

(III-Pno-cib) $\boldsymbol{P}^u_{\mathbf{no}}(q_{\mathtt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{nu}}$,

(III-Polb-cib) $\boldsymbol{P}^u_{\mathbf{olb}}(q_{\mathtt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{su}} \cup \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{sr}} \cup \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{nr}}$, et

(III-Peb-cib) $\boldsymbol{P}^u_{\mathbf{eb}}(q_{\mathtt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{P}^{q_{\mathtt{cib}}}_{\mathbf{ebu}}$.

L'étape suivante qui consiste à examiner $\boldsymbol{K}^u(q_{\mathtt{cib}})$ et $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\mathtt{cib}}}$ est synthétisée comme suit.

$$\boldsymbol{K}^u_{\mathbf{no}}(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nu}}. \quad \text{(R4-a)}$$

$$\boldsymbol{K}^u_{\mathbf{olb}}(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{su}} \cup \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{sr}} \cup \boldsymbol{L}^{q_{\texttt{cib}}}_{\mathbf{nr}}. \quad \text{(R4-b)}$$

où, $\boldsymbol{L}^{q_{\texttt{cib}}}_{\mathbf{nr}} = \bigcup_{i=1}^{k_{\mathbf{nr}}} [pre\text{-}N^i_{\mathbf{nr}} \cup \mathit{Child}(t, pre\text{-}T^i_{\mathbf{nr}})]$,

$$\boldsymbol{K}^u_{\mathbf{eb}}(q_{\texttt{cib}}) = \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{ebu}}. \quad \text{(R4-c)}$$

D'après (R4-a), (R4-b) et (R4-c), on a :
$\boldsymbol{K}^u(q_{\texttt{cib}}) = \bigcup_{\alpha\in\{\mathbf{nu},\mathbf{ebu},\mathbf{su},\mathbf{sr}\}} \mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\alpha} \cup \boldsymbol{L}^{q_{\texttt{cib}}}_{\mathbf{nr}}$.

Or, $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} \subseteq \boldsymbol{L}^{q_{\texttt{cib}}}_{\mathbf{nr}}$ puisque $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}}_{\mathbf{nr}} = \bigcup_{i=1}^{k_{\mathbf{nr}}} [pre\text{-}N^i_{\mathbf{nr}}]$.

Donc, $\mathbf{pre}\text{-}\boldsymbol{K}^{q_{\texttt{cib}}} \subseteq \boldsymbol{K}^u(q_{\texttt{cib}})$.

Preuve de $\mathbf{eval}\text{-}\boldsymbol{K}^{q_{\texttt{src}}} \subseteq \boldsymbol{K}^u(q_{\texttt{src}})$

Cette preuve est identique à celle du (II) puisque les règles d'extraction des chemins à partir de la requête $q_{\texttt{src}}$ sont les mêmes pour les deux cas.

Cas d'induction

Supposons maintenant que pour toute mise à jour U de taille inférieure à n, étant donné un contexte dynamique Υ et un environnement statique Γ consistant par rapport à Υ, alors l'id-projecteur $\boldsymbol{K}^U$ construit à partir des chemins inférés pour U par rapport à Γ est correct pour la pré-évaluation de U par rapport à Υ. Montrons que cette hypothèse reste vérifiée pour les mises à jour de taille n. Nous devons considérer les cas suivants :

(I) $U = \mathbf{let}\ x$ = $q\ \mathbf{return}\ u$ avec $|u|{=}n-1$,
(II) $U = \mathbf{for}\ x\ \texttt{in}\ q\ \mathbf{return}\ u$ avec $|u|{=}n-1$,
(III) $U = \mathbf{if}\ q\ \mathbf{then}\ u_1\ \mathbf{else}\ u_2$ avec $|u_i| < n-1$ pour $i{=}1,2$ et
(IV) $U = u_1, u_2$ avec $|u_i| < n$ pour $i{=}1,2$.

(I) [$U{=}\mathbf{let}\ x$ = $q\ \mathbf{return}\ u$, $|u|{=}n-1$] : La sémantique de U est donnée par la règle suivante :

$$(\mathbf{Up{:}Let})\ \frac{\overset{\textcircled{1}}{(\sigma_t, \sigma_\epsilon, d, \gamma \models q \Rightarrow \sigma_q, \mathrm{I})} \quad \overset{\textcircled{2}}{(\sigma_t, \sigma_\epsilon{\cdot}\sigma_q, d, ext(\gamma, x, \mathrm{I}) \models u \Rightarrow \sigma_\omega, \omega)}}{\sigma_t, \sigma_\epsilon, d, \gamma \models \mathbf{let}\ x\ \texttt{=}\ q\ \mathbf{return}\ u \Rightarrow \sigma_\omega, \omega}$$

Cette règle spécifie que la pré-évaluation de U partir d'un document t et d'un contexte dynamique $\Upsilon{=}(\sigma_\epsilon, d, \gamma)$ se déroule en deux étapes :

1. pré-évaluation de la requête contextuelle q à partir de Υ qui produit un nouveau contexte dynamique $\Upsilon_u{=}\Upsilon{\cdot}(\sigma_q, \emptyset, x{=}\mathrm{I})$ obtenu en étendant Υ avec les résultats de cette pré-évaluation,
2. pré-évaluation de la mise à jour u à partir de Υ_u qui produit la PUL (σ_ω, ω) qui génère le résultat de la pré-évaluation de la mise à jour composée U relativement à t et Υ_u.

Par conséquent, démontrer que $\boldsymbol{K}^U$ est correct pour la pré-évaluation de U à partir de Υ est équivalent à démontrer que $\boldsymbol{K}^U$ est correct pour la pré-évaluation de u à partir de Υ_u. Or, on sait que $|u|{=}n-1$ et d'après l'hypothèse d'induction on sait que $\boldsymbol{K}^U$ est un id-projecteur correct pour la pré-évaluation de u relativement à t et Υ_u ssi :

(i) les contextes dynamiques Υ_u et $\widehat{\Upsilon_u}$ servant à la pré-évaluation de u sur le document initial t et sur la projection $\Pi_K(t)$ respectivement soient équivalents et,
(ii) l'environnement statique Γ_u (utilisé pour l'inférence des chemins de u) soit consistent par rapport à l'environnement dynamique γ_u.

Les points (i) et (ii) sont des conséquences de la correction de l'inférence des chemins pour les requêtes contextuelles énoncée par le lemme 5. En effet, pour (i), il nous faut démontrer $\mathrm{I}{=}\widehat{\mathrm{I}}$ et $\sigma_q{\simeq}\widehat{\sigma_q}$. Pour (ii), il est suffisant de montrer que $\mathrm{I}{=}\widehat{\mathrm{I}}$.

(II) $[U{=}\mathbf{for}\ x\ \mathtt{in}\ q\ \mathbf{return}\ u]$: Le traitement de ce cas suit le même principe que le traitement du cas précédent puisque les deux requêtes ont une sémantique similaire et puisque l'extraction des chemins à partir de ces requêtes est identique.

Nous rappelons d'abord la sémantique de la mise à jour $\mathbf{for}\ x\ \mathtt{in}\ q\ \mathbf{return}\ u$ qui est capturée par le trois règles suivantes :

$$(\mathbf{Up{:}For})\ \frac{\overset{\textbf{1}}{\sigma_t, \sigma_\epsilon, d, \gamma \models q \Rightarrow \sigma_q, \mathrm{I}} \quad \overset{\textbf{2}}{\sigma_t, \sigma_\epsilon{\cdot}\sigma_q, d, \gamma, x{\in}\mathrm{I} \models^{\star} u \Rightarrow \sigma_\omega, \omega}}{\sigma_t, \sigma_\epsilon, d, \gamma \models \mathbf{for}\ x\ \mathtt{in}\ q\ \mathbf{return}\ u \Rightarrow \sigma_\omega, \omega}$$

$$(\mathbf{Up{:}For{:}stop})\ \frac{}{\sigma_t, \sigma_\epsilon, d, \gamma, x{\in}() \models^{\star} u \Rightarrow \emptyset, \epsilon}$$

$$(\mathbf{Up{:}For{:}iter})\ \frac{\overset{\textbf{2-a}}{\sigma_t, \sigma_\epsilon, d, \gamma, x{\in}\mathrm{I} \models^{\star} u \Rightarrow \sigma'_\omega, \omega'} \quad \overset{\textbf{2-b}}{\sigma_t, \sigma_\epsilon, d \cup \mathit{dom}(\sigma'_\omega), ext(\gamma, x, \mathbf{i}) \models u \Rightarrow \sigma''_\omega, \omega''}}{\sigma_t, \sigma_\epsilon, d, \gamma, x{\in}\mathrm{I}{\cdot}\mathbf{i} \models^{\star} u \Rightarrow \sigma'_\omega \cdot \sigma''_\omega, \omega'; \omega''}$$

La règle $(\mathbf{Up{:}For})$ capture le fait que la pré-évaluation de cette mise à jour s'effectue en deux étapes : pré-évaluation de q suivie de la pré-évaluation itérative de u. Cette dernière est capturée par la règle $(\mathbf{Up{:}For{:}stop})$ qui spécifie le cas terminal et la règle $(\mathbf{Up{:}For{:}iter})$ qui spécifie que le résultat $(\sigma''_\omega, \omega'')$ de chaque nouvelle itération est concaténé au résultat en cours $(\sigma'_\omega, \omega')$. Lors de chaque nouvelle itération, le domaine des identifiants interdits est enrichi avec les identifiants de σ'_ω et l'environnement des variables est étendu avec un lien allant de la variable x à l'identifiant $\mathbf{i}$.

Nous dressons une table permettant de mettre en évidence la pré-évaluation de u d'une part, sur t à partir de Υ, et d'autre part sur $\Pi_{\boldsymbol{K}}(t)$ à partir de $\widehat{\Upsilon}$. Rappelons que $\mathrm{PULIter}_{\Upsilon,x}(\mathrm{I}, \sigma_t, u)$ est une fonction introduite dans le Chapitre 2 pour capturer la pré-évaluation itérative de u à partir d'un store σ_t, d'un contexte dynamique Υ pour chaque $\mathbf{i} \in \mathrm{I}$.

$\underline{\mathit{PUL}_{\Upsilon}(\sigma_t, \mathbf{for}\ x\ \mathtt{in}\ q\ \mathbf{return}\ u)}$	$\underline{\mathit{PUL}_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^U}(\sigma_t), \mathbf{for}\ x\ \mathtt{in}\ q\ \mathbf{return}\ u)}$
$(1)\mathit{QEval}_{\Upsilon}(\sigma_t, q){=}(\sigma_q, \mathrm{J})$	$(1)\mathit{QEval}_{\widehat{\Upsilon}}(\Pi_{\boldsymbol{K}^U}(\sigma_t), q){=}(\widehat{\sigma_q}, \widehat{\mathrm{J}})$
$\Upsilon_1{=}\Upsilon{\cdot}(\sigma_q, \emptyset, ())$	$\widehat{\Upsilon_1}{=}\widehat{\Upsilon}{\cdot}(\widehat{\sigma_q}, \emptyset, ())$
$(2)\mathrm{PULIter}_{\Upsilon_1,x}(\mathrm{J}, \sigma_t, u){=}(\sigma_\omega, \omega)$	$(2)\mathrm{PULIter}_{\widehat{\Upsilon_1},x}(\widehat{\mathrm{J}}, \Pi_{\boldsymbol{K}^U}(\sigma_t), u){=}(\widehat{\sigma_\omega}, \widehat{\omega})$

Notre but est de démontrer que

$$(\sigma \cdot \sigma_u, \omega) \sim (\Pi_{\boldsymbol{K}^U}(\sigma_t) \cdot \widehat{\sigma_u}, \widehat{\omega}) \tag{B.2}$$

Pour ce faire, nous procédons par double inclusion. Remarquez que $\boldsymbol{K}^U$ est correct pour la pré-évaluation de q relativement à t et Υ, entraîne $\mathrm{J} = \widehat{\mathrm{J}}$ et $\sigma_q \simeq \widehat{\sigma_q}$. Rappelons que d'après la règle (**Up:For:iter**) nous avons $|\mathrm{PULIter}_{\Upsilon_u,x}(\mathrm{J}, t, u)| = |\mathrm{J}|$ et $|\mathrm{PULIter}_{\widehat{\Upsilon_u},x}(\widehat{\mathrm{J}}, \Pi_{\boldsymbol{K}^U}(\sigma_t), u)| = |\widehat{\mathrm{J}}|$. Ce qui nous permet de procéder par induction sur $|\mathrm{J}|$.

Sens $\sqsubseteq$ Nous voulons démontrer

$$(\sigma \cdot \sigma_u, \omega) \sqsubseteq (\Pi_{\boldsymbol{K}^U}(\sigma_t) \cdot \widehat{\sigma_u}, \widehat{\omega}) \tag{B.3}$$

[Cas de base] : $|\mathrm{J}|=0$. Dans ce cas, on applique la règle (**Up:For:stop**).

On aura d'une part,

$\mathrm{PULIter}_{\Upsilon_u,x}(\mathrm{J}, \sigma_t, u) = (\varnothing, \emptyset)$

et d'autre part,

$\mathrm{PULIter}_{\widehat{\Upsilon_u},x}(\mathrm{J}, \Pi_{\boldsymbol{K}^U}(\sigma_t), u) = (\varnothing, \emptyset)$.

Donc (B.3) est vraie.

[Cas induction] : nous supposons maintenant que (B.3) est vraie pour J tel que $|\mathrm{J}|<n$ et nous montrons qu'elle reste vraie pour J tel que $|\mathrm{J}|=n$. Posons $\mathrm{J} = \mathrm{I} \cdot \mathbf{i}$ avec $|\mathrm{I}| = n-1$. Nous dressons un tableau pour mettre en évidence les résultats obtenus de la pré-évaluation sur t et sur $\Pi_{\boldsymbol{K}^U}(\sigma_t)$.

$(\sigma_\omega, \omega) = (\sigma'_\omega \cdot \sigma''_\omega, \omega' \cdot \omega'')$	$(\widehat{\sigma_u}, \widehat{\omega}) = (\widehat{\sigma'_\omega} \cdot \widehat{\sigma''_\omega}, \widehat{\omega'} \cdot \widehat{\omega''})$
(2-a) $\mathrm{PULIter}_{\Upsilon_1,x}(\mathrm{I}, t, u) = (\sigma'_\omega, \omega')$	(2-a) $\mathrm{PULIter}_{\widehat{\Upsilon_1},x}(\mathrm{I}, t, u) = (\widehat{\sigma'_\omega}, \widehat{\omega'})$
$\Upsilon_n = \Upsilon_1 \cdot (\varnothing, \mathit{dom}(\sigma'_\omega), x{=}\mathbf{i})$	$\widehat{\Upsilon_n} = \widehat{\Upsilon_1} \cdot (\varnothing, \mathit{dom}(\widehat{\sigma'_\omega}), x{=}\mathbf{i})$
(2-b) $\mathit{PUL}_{\Upsilon_n}(\sigma_t, u) = (\sigma''_\omega, \omega'')$	(2-b) $\mathit{PUL}_{\widehat{\Upsilon_n}}(\Pi_{\boldsymbol{K}^U}(\sigma_t), u) = (\widehat{\sigma''_\omega}, \widehat{\omega''})$

D'après le lemme 5-bis, $\boldsymbol{K}^U$ est correct pour la pré-évaluation de q et donc que $\Upsilon_1 \simeq \widehat{\Upsilon_1}$. Par hypothèse d'induction on déduit que :

$(\sigma \cdot \sigma'_\omega, \omega') \sqsubseteq (\Pi_{\boldsymbol{K}^U}(\sigma_t) \cdot \widehat{\sigma'_\omega}, \widehat{\omega'})$.

Il nous reste à démontrer que :

$(\sigma \cdot \sigma''_\omega, \omega'') \sqsubseteq (\Pi_{\boldsymbol{K}^U}(\sigma_t) \cdot \widehat{\sigma''_\omega}, \widehat{\omega''})$ pour déduire (B.3).

Il est facile de voir que $\Upsilon_n \simeq \widehat{\Upsilon_n}$ puisque :

$\Upsilon_1 \simeq \widehat{\Upsilon_1}$ et $\sigma'_\omega \simeq \widehat{\sigma'_\omega}$ ($\boldsymbol{K}^U$ correct pour la pré-évaluation de q).

Donc, on a bien (B.3).

Sens $\sqsupseteq$ Cette preuve se fait de manière identique à la preuve du cas précédent.

Il n'est pas nécessaire de traiter les cas $U{=}\mathbf{if}\ q\ \mathbf{then}\ u_1\ \mathbf{else}\ u_2$, $|u_1|{+}|u_2|{=}n{-}1$ et $U = u_1, u_2$ puisque on utilise le même principe que les cas déjà traités.

□

Rappelons que notre but était de démontrer la correction l'inférence des chemins pour les mises à jour dans le cas général où les contextes dynamiques et l'environnement statique de départ sont vides. Le corollaire permet d'énoncer ce résultat étant donné que les contextes dynamiques vides sont équivalents et que l'environnement statique vide est consistent par rapport à un contexte dynamique vide.

Corollaire 2.
Soit $\boldsymbol{K}$ un id-projecteur construit comme défini dans la lemme 4, alors $\boldsymbol{K}$ est correct pour la pré-évaluation de u relativement à t.

B.2 Preuve du Théorème 3

La preuve du Théorème 3 s'appuie sur la correction de l'inférence des chemins pour les mises à jour énoncée par le lemme 4. Etant donné l'id-projecteur $\boldsymbol{K}^u$ défini dans ce lemme, il suffit de démontrer que $\boldsymbol{K}^u \subseteq K(t,\pi)$ et d'utiliser la propriété de monotonie 9 pour déduire que $K(t,\pi)$ est correct pour u relativement à t.

Afin de démontrer $\boldsymbol{K}^u \subseteq K(t,\pi)$, nous définissons $L(t,\mathbf{pi})$ une projection pour les mises à jour à base des path-projecteurs. Cela permet de faire le lien, d'une part avec $\boldsymbol{K}^u$ qui est défini à partir de path-projecteurs et d'autre part avec $K(t,\pi)$ extrait du tri-projecteur. Nous montrons que $\boldsymbol{K}^u \subseteq L(t,\mathbf{pi})$ puis que $L(t,\mathbf{pi}) = K(t,\pi)$ pour déduire que $\boldsymbol{K}^u \subseteq K(t,\pi)$. Afin de faciliter la preuve de $L(t,\mathbf{pi}) = K(t,\pi)$, nous considérons une étape supplémentaire qui consiste à démontrer que $L(t,\mathbf{pi}) = K^*(t,\pi^*)$ où π^* est un tri-projecteur relaxé pour lequel la propriété de disjonction n'est pas requise et $K^*(t,\pi^*)$ est une relaxation de la projection $K(t,\pi)$. Il suffit ensuite de montrer $K(t,\pi) = K^*(t,\pi^*)$ pour déduire $\boldsymbol{K}^u \subseteq K(t,\pi)$.

Les étapes de la preuve de $\boldsymbol{K}^u \subseteq K(t,\pi)$ sont récapitulées dans l'encadré ci-dessous.

$\boldsymbol{K}^u$	$\subseteq$	$L(t,\mathbf{pi})$	$=$	$K^*(t,\pi^*)$	$=$	$K(t,\pi)$
Lemme 4						Définition 18

Nous définissons formellement les path-projecteurs pour les mises à jour.

Définition 32 (Path-projecteur pour les mises à jour)**.**
Soit D une DTD et u une mise à jour.
Soient $\boldsymbol{P}_{\mathbf{no}}, \boldsymbol{P}_{\mathbf{olb}}$ et $\boldsymbol{P}_{\mathbf{eb}}$ les ensembles de chemins inférés de u et D. Comme précédemment, on suppose que $\boldsymbol{P}_\alpha = \{P_\alpha^1, \ldots, P_\alpha^{k_\alpha}\}$.
Soit $\mathbf{pi}_\alpha^i$=(T_α^i, N_α^i) le path-projecteur inféré pour P_α^i et D.
Le path-projecteur pour u est donné par :

$\mathbf{pi} = (\mathbf{pi}_{\mathbf{no}}, \mathbf{pi}_{\mathbf{olb}}, \mathbf{pi}_{\mathbf{eb}})$ où $\mathbf{pi}_\alpha$=$(\bigcup_{i=1}^{k_\alpha} T_\alpha^i, \bigcup_{i=1}^{k_\alpha} N_\alpha^i)$.

Définition 33 (Path-projection pour les mises à jour)**.**
Soit D une DTD, $t \in D$ un document et u une mise à jour.
Soit $\mathbf{pi}$ le path-projecteur inféré de u et D.
La path-projection pour t relativement à $\mathbf{pi}$, notée $L(t, \mathbf{pi})$, est donnée par l'arbre $\Pi_{L(t,\mathbf{pi})}(t)$ où $L(t, \mathbf{pi})$=$\bigcup_{\alpha\in\{\mathbf{no},\mathbf{olb},\mathbf{eb}\}} L_\alpha(t, \mathbf{pi})$ et :

- $L_{\mathbf{no}}(t, \mathbf{pi})$=$KPath(t, \mathbf{pi}_{\mathbf{no}}) \cup KPath(t, \mathbf{pi}_{\mathbf{olb}}) \cup KPath(t, \mathbf{pi}_{\mathbf{eb}})$,
- $L_{\mathbf{olb}}(t, \mathbf{pi})$=*Child*$(t, KNode(t, \mathbf{pi}_{\mathbf{olb}}))$,
- $L_{\mathbf{eb}}(t, \mathbf{pi})$=*Desc*$(t, KNode(t, \mathbf{pi}_{\mathbf{eb}}))$.

Propriété 13.
Soit D une DTD, $t \in D$ un document et u une mise à jour.
Soit $\mathbf{pi}$ le path-projecteur tel que défini dans Définition 32.
Soit $\boldsymbol{K}^u$ l'id-projecteur obtenu de u tel que spécifié dans lemme 4.
Alors on a : $\boldsymbol{K}^u \subseteq L(t, \mathbf{pi})$.

Démonstration. On procède suivant le même principe que celui utilisé pour la preuve du lemme 4-bis c'est à dire, en comparant les composants respectifs de $\boldsymbol{K}^u$ et $L(t, \mathbf{pi})$.

On considère $\boldsymbol{P}_{\mathbf{no}}, \boldsymbol{P}_{\mathbf{olb}}, \boldsymbol{P}_{\mathbf{eb}}$, $\mathbf{pi}_\alpha^i$, $\mathbf{pi}$ de la Définition 32.
On pose,

- KT_α^i=$PEval(t, P_\alpha^i)$
- KN_α^i=$KPath(t, \mathbf{pi}_\alpha^i)$

D'après le lemme 4, $\boldsymbol{K}^u = \boldsymbol{K}^u_{\mathbf{no}} \cup \boldsymbol{K}^u_{\mathbf{olb}} \cup \boldsymbol{K}^u_{\mathbf{eb}}$ où,

- $\boldsymbol{K}^u_{\mathbf{no}}$=$\bigcup_{i=1}^{k_{\mathbf{no}}}[KN^i_{\mathbf{no}}]$,
- $\boldsymbol{K}^u_{\mathbf{olb}}$=$\bigcup_{i=1}^{k_{\mathbf{olb}}}[KN^i_{\mathbf{olb}} \cup$ *Child*$(t, KT^i_{\mathbf{olb}})]$,
- $\boldsymbol{K}^u_{\mathbf{eb}}$=$\bigcup_{i=1}^{k_{\mathbf{eb}}}[KN^i_{\mathbf{eb}} \cup$ *Desc*$(t, KT^i_{\mathbf{eb}})]$

Posons :

· $\boldsymbol{K}^u_{\mathbf{olb}} = \boldsymbol{K}^u_{\mathbf{olb}}$-$N \cup \boldsymbol{K}^u_{\mathbf{olb}}$-$T$, avec

$\boldsymbol{K}^u_{\mathbf{olb}}$-$N = \bigcup_{i=1}^{k_{\mathbf{olb}}}[KN^i_{\mathbf{olb}}]$ et $\boldsymbol{K}^u_{\mathbf{olb}}$-$T = \bigcup_{i=1}^{k_{\mathbf{olb}}}[$*Child*$(t, KT^i_{\mathbf{olb}})]$.

· $\boldsymbol{K}^u_{\mathbf{eb}}$=$\boldsymbol{K}^u_{\mathbf{eb}}$-$N \cup \boldsymbol{K}^u_{\mathbf{eb}}$-$T$, avec

$\boldsymbol{K}^u_{\mathbf{eb}}$-$N = \bigcup_{i=1}^{k_{\mathbf{eb}}}[KN^i_{\mathbf{eb}}]$ et $\boldsymbol{K}^u_{\mathbf{eb}}$-$T = \bigcup_{i=1}^{k_{\mathbf{eb}}}[$*Desc*$(t, KT^i_{\mathbf{eb}})]$.

D'après la définition 33 on a :

$$
\begin{aligned}
-L_{\mathbf{no}}(t,\mathbf{pi}) &= KPath(t,\mathbf{pi}_{\mathbf{no}}) \cup KPath(t,\mathbf{pi}_{\mathbf{olb}}) \cup KPath(t,\mathbf{pi}_{\mathbf{eb}}) \\
&= \bigcup_{i=1}^{k_{\mathbf{no}}} [KPath(t,\mathbf{pi}^i_{\mathbf{no}})] \cup \bigcup_{i=1}^{k_{\mathbf{olb}}} [KPath(t,\mathbf{pi}^i_{\mathbf{olb}})] \cup \bigcup_{i=1}^{k_{\mathbf{eb}}} [KPath(t,\mathbf{pi}^i_{\mathbf{eb}})] \\
&= \bigcup_{i=1}^{k_{\mathbf{no}}} [KN^i_{\mathbf{no}}] \cup \bigcup_{i=1}^{k_{\mathbf{olb}}} [KN^i_{\mathbf{olb}}] \cup \bigcup_{i=1}^{k_{\mathbf{eb}}} [KN^i_{\mathbf{eb}}] \\
&= \quad \boldsymbol{K}^u_{\mathbf{no}} \quad \cup \quad \boldsymbol{K}^u_{\mathbf{olb}}\text{-}N \quad \cup \quad \boldsymbol{K}^u_{\mathbf{eb}}\text{-}N \qquad (\dagger\text{-}1) \\
-L_{\mathbf{olb}}(t,\mathbf{pi}) &= \mathit{Child}(t, KNode(t,\mathbf{pi}_{\mathbf{olb}})) \\
&= \bigcup_{i=1}^{k_{\mathbf{olb}}} [\mathit{Child}(t, KNode(t,\mathbf{pi}^i_{\mathbf{olb}}))] \\
-L_{\mathbf{eb}}(t,\mathbf{pi}) &= \mathit{Desc}(t, KNode(t,\mathbf{pi}_{\mathbf{eb}})) \\
&= \bigcup_{i=1}^{k_{\mathbf{eb}}} [\mathit{Desc}(t, KNode(t,\mathbf{pi}^i_{\mathbf{eb}}))]
\end{aligned}
$$

D'après $(\dagger\text{-}1)$, on a :

$(\dagger\text{-}1')$ $\boldsymbol{K}^u = L_{\mathbf{no}}(t,\mathbf{pi}) \cup \boldsymbol{K}^u_{\mathbf{olb}}\text{-}T \cup \boldsymbol{K}^u_{\mathbf{eb}}\text{-}T$.

Or, d'après la propriété 5 donnée en page 68 on a :

$KT^i_\alpha = PEval(t, P^i_\alpha) \subseteq KNode(t,\mathbf{pi}^i_\alpha)$.

Donc,

$\mathit{Child}(t, KT^i_\alpha) \subseteq \mathit{Child}(t, KNode(t,\mathbf{pi}^i_\alpha))$ et

$\mathit{Desc}(t, KT^i_\alpha) \subseteq \mathit{Desc}(t, KNode(t,\mathbf{pi}^i_\alpha))$.

Cela implique que

$(\dagger\text{-}2)$ $\boldsymbol{K}^u_{\mathbf{olb}}\text{-}T \subseteq L_{\mathbf{olb}}(t,\mathbf{pi})$ et

$(\dagger\text{-}3)$ $\boldsymbol{K}^u_{\mathbf{eb}}\text{-}T \subseteq L_{\mathbf{eb}}(t,\mathbf{pi})$.

D'après $(\dagger\text{-}1')$, $(\dagger\text{-}2)$ et $(\dagger\text{-}3)$, on déduit que $\boldsymbol{K}^u \subseteq L(t,\mathbf{pi})$ □

Nous définissons le tri-projecteur relaxé π^* de la manière suivante.

Définition 34 (Tri-projecteur relaxé)**.**
Soit D une DTD, $t \in D$ un document et u une mise à jour.

Soient $\boldsymbol{P}_{\mathbf{no}}, \boldsymbol{P}_{\mathbf{olb}}$ et $\boldsymbol{P}_{\mathbf{eb}}$ les chemins inféré de u et D avec $\boldsymbol{P}_\alpha = \{P^1_\alpha, \ldots, P^{k_\alpha}_\alpha\}$.

Soit (T^i_α, N^i_α) le path-projecteur inféré pour le chemin P^i_α pour $\alpha \in \{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$ et $i \in \{1, .., k_\alpha\}$.

Posons $\tau_\alpha = \bigcup_{i=1}^{k_\alpha} T^i_\alpha$ et $\kappa_\alpha = \bigcup_{i=1}^{k_\alpha} N^i_\alpha$.

Le tri-projecteur $\pi^* = (\pi^*_{\mathbf{no}}, \pi^*_{\mathbf{olb}}, \pi^*_{\mathbf{eb}})$ pour u est donné par :

$$\begin{array}{lcl} \pi^*_{\mathbf{no}} & = & \kappa_{\mathbf{no}} \cup \kappa_{\mathbf{olb}} \cup \kappa_{\mathbf{eb}}, \\ \pi^*_{\mathbf{olb}} & = & \tau_{\mathbf{olb}}, et \\ \pi^*_{\mathbf{eb}} & = & \tau_{\mathbf{eb}}. \end{array}$$

La projection associée au tri-projecteur relaxé π^* est similaire à la projection associée au tri-projecteur π donnée par la définition 18. Seule la construction de $K^*(t, \pi^*)$ à partir des id-sets $K^*_\alpha(t, \pi^*)$ diffère de celle de $K(t, \pi)$. Comme les composants du tri-projecteur relaxé π^* ne sont pas disjoints, il est nécessaire de récupérer les identifiants générés pour toutes les étiquettes dans π^*. Cela est effectué en utilisant l'union (point b. de la définition ci-dessou).

Définition 35 (Tri-projection relaxée)**.**
Considérons une DTD (D, s_D), un tri-projecteur relaxé $\pi^*{=}(\pi^*_{\mathbf{no}}, \pi^*_{\mathbf{olb}}, \pi^*_{\mathbf{eb}})$ et un document $t{\in}D$. On pose *roots*$(t){=}\{r_t\}$ et *subfor*$(t){=}F$.
La projection de t suivant π^*, notée $\pi^*(t)$, est le document $\Pi_{K^*(t,\pi^*)}(t)$ où $K^*(t, \pi^*)$ est l'ensemble d'identifants défini récursivement par :

a′. si *lab*$(r_t){\notin}\pi^*$ alors $K^*(t, \pi^*){=}\emptyset$,

b′. si *lab*$(r_t){\in}\pi^*_\alpha$, où $\alpha{\in}\{\mathbf{no}, \mathbf{olb}, \mathbf{eb}\}$, alors $K^*(t, \pi^*){=}\{r_t\}{\cup}\bigcup_{\alpha\in\{\mathbf{no},\mathbf{olb},\mathbf{eb}\}} K^*_\alpha(F, \pi^*)$ avec :

1′. $K^*_\alpha(F, \pi^*){=}\emptyset$ si $F{=}()$ et sinon, en supposant que $F{=}t' \cdot F'$ et *roots*$(t'){=}\{r_{t'}\}$,

2′. $K^*_{\mathbf{no}}(F, \pi^*){=}K(t', \pi^*){\cup}K^*_{\mathbf{no}}(F', \pi^*)$,

3′. $K^*_{\mathbf{olb}}(F, \pi^*){=}K(t', \pi^*){\cup}K^*_{\mathbf{olb}}(F', \pi^*)$ si *lab*$(r_{t'}){\in}\pi^*$,

4′. $K^*_{\mathbf{olb}}(F, \pi^*){=}\{r_{t'}\}{\cup}K^*_{\mathbf{olb}}(F', \pi^*)$ si *lab*$(r_{t'}){\notin}\pi^*$,

5′. $K_{\mathbf{eb}}(F, \pi^*){=}$*dom*$(F)$.

Propriété 14.
Soit D une DTD, $t \in D$ un document et u une mise à jour. Soit $\mathbf{pi}$ le path-projecteur inféré de u et D suivant la définition 32. Soit π^* le tri-projecteur inféré de u et D suivant la Définition 34. Alors on a : $L(t, \mathbf{pi}) = K^*(t, \pi^*)$.

Démonstration. Cette preuve est directe puisqu'il suffit d'examiner la construction de ces deux id-projecteurs à partir des chemins extraits de la mise à jour u, et ce en utilisant les définitions 32 et 34. □

Pour conclure la preuve du Théorème 3 il suffit de montrer que la projection relaxée en utilisant le tri-projecteur relaxé π^* produit le même résultat que la projection "classique" qui utilise le tri-projecteur π dont les composants sont disjoints deux à deux.

Cette démonstration est immédiate, il suffit de voir que chaque ligne de la définition 35 est équivalente à la ligne correspondante dans la définition 18.

Annexe C

Correction du tri-projecteur

La preuve du Théorème 5 repose sur le résultat suivant qui découle de la correction du tri-projecteur inféré pour les mises à jour. Etant donné un document t, une mise à jour u et la PUL (σ_ω, ω) générée pour u sur t, alors *dom*(ω) a été introduit dans le Chapitre 2 pour désigner les identifiants des cibles des mises à jour primitives de ω et le domaine du store σ_ω produit lors de la génération de (σ_ω, ω) par pré-évaluation de u sur t.

Rappelons que l'ensemble des cibles des primitives des mises à jour est défini par :
$Cible(\text{ins}(\mathbf{i}_{\texttt{cib}}, \delta, \text{I}_{\texttt{src}})) = Cible(\text{repl}(\text{I}_{\texttt{src}}, \mathbf{i}_{\texttt{cib}})) = Cible(\text{del}(\mathbf{i}_{\texttt{cib}})) = Cible(\text{ren}(\mathbf{i}_{\texttt{cib}}, a)) = \{\mathbf{i}_{\texttt{cib}}\}$.

Rappelons également que σ_ω correspond aux noeuds devant être insérés par les primitives d'insertions (ins$(\mathbf{i}_{\texttt{cib}}, \delta, \text{I}_{\texttt{src}})$) ou de remplacements (repl$(\text{I}_{\texttt{src}}, \mathbf{i}_{\texttt{cib}})$) appartenant à ω.

Lorsqu'un noeud **i** est cible d'une mise à jour primitive de ω i.e **i** $\in Cible(\omega)$ alors ce noeud à forcément été projeté par π i.e **i** $\in K(t, \pi)$. Lorsqu'un noeud **i** appartient au store σ_ω produit par pré-évaluation de u sur t, alors il existe un noeud $\widehat{\mathbf{i}}$ appartenant au store $\widehat{\sigma}_\omega$ produit par pré-évaluation de u sur $\Pi_K(t)$.
Formellement,

Corollaire 3.
Soit une DTD D, un document t, une mise à jour u et le tri-projecteur π inféré pour u et D. L'équivalence $(\sigma_t \cdot \sigma_\omega, \omega) \sim (\Pi_K(t) \cdot \widehat{\sigma_\omega}, \widehat{\omega})$ induit naturellement un isomorphisme
ρ : *dom*$(\sigma_\omega) \rightarrow$ *dom*$(\widehat{\sigma_\omega})$, où on a :

$$\text{si } \mathbf{i} \in \textit{dom}(\omega) \cap \textit{dom}(\sigma_t) \text{ alors } \rho(\mathbf{i}) = \mathbf{i} \text{ et } \mathbf{i} \in K(t, \pi) \tag{C.1}$$

$$\text{si } \mathbf{i} \in \textit{dom}(\omega) - \textit{dom}(t) \quad \text{alors } (\sigma_\omega, \mathbf{i}) \simeq (\widehat{\sigma_\omega}, \rho(\mathbf{i})) \tag{C.2}$$

Démonstration. Soit une mise à jour u, une DTD D, un document $t \in D$ et le type-projecteur π inféré à partir de u et D.
Soit $(\sigma_\omega, \omega) = PUL(t, u)$ la PUL obtenue en pré-évaluant u sur le document initial t.
Soit $(\widehat{\sigma_\omega}, \widehat{\omega}) = PUL(\pi(t), u)$ la PUL obtenue en pré-évaluant u sur le document projeté $\pi(t)$.
D'après le théorème 3 on a :

$$(\dagger)\ (\sigma_t \cdot \sigma_\omega, \omega) \sim (\pi(t) \cdot \widehat{\sigma_\omega}, \widehat{\omega})$$

et on peut poser :
$\omega{=}\mu_1, \ldots, \mu_n$, et $\widehat{\omega}{=}\widehat{\mu_1}, \ldots, \widehat{\mu_n}$.

Montrons que pour $i = 0 \ldots n$ on a :

$$\widehat{\Lambda_i} \simeq \Pi_{\mathrm{J}_i}(\Lambda_i) \text{ avec } \mathrm{J}_i = \mathit{dom}(\Lambda_i) - [\mathit{dom}(t) - K(t,\pi)] \quad \text{(C.3a)}$$

$$\mathbf{i}{\in}[\mathit{dom}(\Lambda_i) - \mathrm{J}_i] \text{ implique que } \Lambda_i(\mathbf{i}){=}t(\mathbf{i}) \quad \text{(C.3b)}$$

où,

- $\Lambda_0{=}t$, et pour $i = 1..n$, $\Lambda_i = \mu_i(\Lambda_{i-1})$ est le résultat de l'application de la primitive de mise à jour μ_i sur Λ_{i-1}, et de manière similaire,
- $\widehat{\Lambda_0}{=}\pi(t)$, et pour $i = 1..n$, $\widehat{\Lambda_i} = \widehat{\mu_i}(\widehat{\Lambda_{i-1}})$ est le résultat de l'application de la primitive de mise à jour $\widehat{\mu_i}$ sur $\widehat{\Lambda_{i-1}}$.

Remarque 1 : La projection de Λ_i sur J_i est bien définie, c'est un arbre.

Remarque 2 : La définition ci-dessus de Λ_i, $\widehat{\Lambda_i}$ et J_i implique que :
$\Lambda_n{=}u(t)$, $\widehat{\Lambda_n}{=}u(\pi(t))$ et aussi $\mathrm{J}_n{=}\mathrm{J}$
donc la démonstration de C.3a et C.3b pour $i = 0 \ldots n$ constitue une démonstration du Théorème 5.

[Cas de base] $i{=}0$. Immédiat en remarquant que :
$\mathrm{J}_0{=}\mathit{dom}(\Lambda_0){-}[\mathit{dom}(t){-}K(t,\pi)]{=}\mathit{dom}(t){\cap}K(t,\pi){=}K(t,\pi)$.

[Cas d'induction] Supposons que (C.3a) et (C.3b) sont vérifiés pour $i{=}0, \ldots, k-1$, avec $k{\leq}n$. La démonstration de (C.3a) et (C.3b) pour k consiste en une analyse de cas portant sur la nature de la primitive de mise à jour μ_k.

$\boxed{\mu_k{=}\mathrm{ins}(\mathrm{I}_{\mathtt{new}}, \downarrow, \mathbf{i})}$
Rappelons ci-dessous la règle définissant l'application de la primitive de mise à jour μ_k sur le document Λ_{k-1} en présence du store auxiliaire σ_ω.

$$(\mathbf{ins{-}into})\frac{\Lambda_{k-1}(\mathbf{i}) = a[\mathrm{I}_1 \cdot \mathrm{I}_2]}{\Lambda_{k-1}, \sigma_\omega \models \mathrm{ins}(\mathrm{I}_{\mathtt{new}}, \downarrow, \mathbf{i}) \rightsquigarrow \mathit{SubStore}(\Lambda_{k-1}, \mathbf{i}, a[\mathrm{I}_1 \cdot \mathrm{I}_{\mathtt{new}} \cdot \mathrm{I}_2], \sigma_\omega)}$$

Cette règle implique qu'en posant $\Lambda_{k-1}(\mathbf{i}){=}a[\mathrm{I}_1 \cdot \mathrm{I}_2]$, on a $\Lambda_k(\mathbf{i}){=}a[\mathrm{I}_1 \cdot \mathrm{I}_{\mathtt{new}} \cdot \mathrm{I}_2]$.

D'après les propriétés (C.1) et (C.2) du corollaire 3 donné en page 217, et $(\dagger)$ on a :
$\mathbf{i} \in K(t,\pi)$ et $\widehat{\mu_k} = \mathrm{ins}(\widehat{\mathrm{I}}_{\mathtt{new}}, \downarrow, \mathbf{i})$ avec $(\sigma_\omega, \mathrm{I}_{\mathtt{new}}) \simeq (\widehat{\sigma_\omega}, \widehat{\mathrm{I}}_{\mathtt{new}})$.

Donc on peut poser $\widehat{\Lambda_{k-1}}(\mathbf{i}){=}a[\widehat{\mathrm{I}}_1 \cdot \widehat{\mathrm{I}}_2]$, et alors $\widehat{\Lambda_k}(\mathbf{i}){=}a[\widehat{\mathrm{I}}_1 \cdot \widehat{\mathrm{I}}_{\mathtt{new}} \cdot \widehat{\mathrm{I}}_2]$.

Pour montrer (C.3a) pour k, i.e. pour montrer que
$\widehat{\Lambda_k} \simeq \Pi_{\mathrm{J}_k}(\Lambda_k)$ avec $\mathrm{J}_k = \mathit{dom}(\Lambda_i) - [\mathit{dom}(t) - K(t,\pi)]$
il suffit de montrer que

$(\widehat{\Lambda_k}, \mathbf{i}) \simeq (\Pi_{\mathrm{J}_k}(\Lambda_k), \mathbf{i})$
ou encore que :
$(\widehat{\Lambda_k}, \widehat{\mathrm{I}}_1 \cdot \widehat{\mathrm{I}}_{\mathtt{new}} \cdot \widehat{\mathrm{I}}_2) \simeq (\Pi_{\mathrm{J}_k}(\Lambda_k), \mathrm{I}_1 \cdot \mathrm{I}_{\mathtt{new}} \cdot \mathrm{I}_2)$.
Ceci se déduit de ce qui précède, en utilisant l'hypothèse d'induction qui assure que
$(\widehat{\Lambda_k}, \widehat{\mathrm{I}}_{\mathtt{new}}) \simeq (\Lambda_k, \mathrm{I}_{\mathtt{new}})$ et
$(\widehat{\Lambda_{k-1}}, \widehat{\mathrm{I}}_1 \cdot \widehat{\mathrm{I}}_2) \simeq (\Pi_{\mathrm{J}_{k-1}}(\Lambda_{k-1}), \mathrm{I}_1 \cdot \mathrm{I}_2)$,
sachant que J_k=$\mathrm{J}_{k-1} \cup \mathrm{I}_{\mathtt{new}} \cup \mathit{Desc}(\sigma_\omega, \mathrm{I}_{\mathtt{new}})$

Pour montrer (C.3b) pour k, i.e. pour montrer que
si $\mathbf{i}$∈[$\mathit{dom}(\Lambda_k) - \mathrm{J}_k$] alors $\Lambda_k(\mathbf{i})$=$t(\mathbf{i})$
il suffit de noter que
$\mathit{dom}(\Lambda_k) - \mathrm{J}_k = \mathit{dom}(\Lambda_{k-1}) - \mathrm{J}_{k-1}$
sachant que J_k=$\mathrm{J}_{k-1} \cup \mathrm{I}_{\mathtt{new}} \cup \mathit{Desc}(\sigma_\omega, \mathrm{I}_{\mathtt{new}})$
et d'utiliser l'hypothèse d'induction qui permet d'assurer que
si $\mathbf{i}$∈[$\mathit{dom}(\Lambda_{k-1}) - \mathrm{J}_{k-1}$] alors $\Lambda_{k-1}(\mathbf{i})$=$t(\mathbf{i})$
sachant que
si $\mathbf{i}$∈[$\mathit{dom}(\Lambda_k) - \mathrm{J}_k$] alors $\Lambda_k(\mathbf{i})$=$\Lambda_{k-1}(\mathbf{i})$.

$\boxed{\mu_k = \mathrm{del}(\mathbf{i})}$
Rappelons ci-dessous la règle définissant l'application de la primitive de mise à jour μ_k sur le document Λ_{k-1} en présence du store auxiliaire σ_ω.

$$(\mathbf{delete})\ \frac{\Lambda_{k-1}(\mathbf{j}) = a[\mathrm{I}_1 \cdot \mathbf{i} \cdot \mathrm{I}_2]}{\Lambda_{k-1}, \sigma_\omega \models \mathrm{del}(\mathbf{i}) \rightsquigarrow SubStore(\Lambda_{k-1}, \mathbf{j}, a[\mathrm{I}_1 \cdot \mathrm{I}_2], \sigma_\omega)}$$

Cette règle implique qu'en posant $\Lambda_{k-1}(\mathbf{j})$=$a[\mathrm{I}_1 \cdot \mathbf{i} \cdot \mathrm{I}_2]$, avec $\mathbf{j}$=$\mathit{Parent}(t, \{\mathbf{i}\})$ on a $\Lambda_k(\mathbf{j})$=$a[\mathrm{I}_1 \cdot \mathrm{I}_2]$.

D'après la propriété (C.1) du corollaire 3 et d'après $(\dagger)$ on a :
$\mathbf{i}, \mathbf{j} \in K(t, \pi)$ et $\widehat{\mu_k} = \mathrm{del}(\mathbf{i})$.

Donc on peut poser $\widehat{\Lambda_{k-1}}(\mathbf{j})$=$a[\widehat{\mathrm{I}}_1 \cdot \mathbf{i} \cdot \widehat{\mathrm{I}}_2]$ et $\widehat{\Lambda_{k-1}}(\mathbf{j})$=$a[\widehat{\mathrm{I}}_1 \cdot \widehat{\mathrm{I}}_2]$

Pour montrer (C.3a) pour k, i.e. pour montrer que
$\widehat{\Lambda_k} \simeq \Pi_{\mathrm{J}_k}(\Lambda_k)$ avec $\mathrm{J}_k = \mathit{dom}(\Lambda_i) - [\mathit{dom}(t) - K(t, \pi)]$
il suffit de montrer que
$(\widehat{\Lambda_k}, \mathbf{j}) \simeq (\Pi_{\mathrm{J}_k}(\Lambda_k), \mathbf{j})$
ou encore que :
$(\widehat{\Lambda_k}, \widehat{\mathrm{I}}_1 \cdot \widehat{\mathrm{I}}_2) \simeq (\Pi_{\mathrm{J}_k}(\Lambda_k), \mathrm{I}_1 \cdot \mathrm{I}_2)$.
Ceci de déduit de ce qui précède, en utilisant l'hypothèse d'induction qui assure que
$(\widehat{\Lambda_{k-1}}, \widehat{\mathrm{I}}_1 \cdot \widehat{\mathrm{I}}_2) \simeq (\Pi_{\mathrm{J}_{k-1}}(\Lambda_{k-1}), \mathrm{I}_1 \cdot \mathrm{I}_2)$,
sachant que J_k=$\mathrm{J}_{k-1} - (\{\mathbf{i}\} \cup \mathit{Desc}(\Lambda_{k-1}, \{\mathbf{i}\}))$

Pour montrer (C.3a) pour k, i.e. pour montrer que
si $\mathbf{i}$∈[$\mathit{dom}(\Lambda_k) - \mathrm{J}_k$] alors $\Lambda_k(\mathbf{i})$=$t(\mathbf{i})$
il suffit de noter que

$dom(\Lambda_k) - \mathrm{J}_k = dom(\Lambda_{k-1}) - \mathrm{J}_{k-1} - (\{\mathbf{i}\} \cup Desc(\Lambda_{k-1}, \mathbf{i}))$
sachant que $\mathrm{J}_k = \mathrm{J}_{k-1} - (\{\mathbf{i}\} \cup Desc(\Lambda_{k-1}, \{\mathbf{i}\}))$
et d'utiliser l'hypothèse d'induction qui permet d'assurer que
si $\mathbf{i} \in [dom(\Lambda_{k-1}) - \mathrm{J}_{k-1}]$ alors $\Lambda_{k-1}(\mathbf{i}) = t(\mathbf{i})$
sachant que
si $\mathbf{i} \in [dom(\Lambda_k) - \mathrm{J}_k]$ alors $\Lambda_k(\mathbf{i}) = \Lambda_{k-1}(\mathbf{i})$.

– $\boxed{\mu_k = \mathrm{repl}(\mathbf{i}, \mathrm{I}_{\texttt{new}})}$
Ce cas n'est pas développé car il se traite sur la base des deux cas précédents sachant qu'une primitive de type `replace` peut s'exprimer à l'aide d'une suppression suivie d'une insertion.

– $\boxed{\mu_k = \mathrm{ren}(\mathbf{i}, a)}$
Ce cas n'est pas développé. Il se traite de manière similaire au cas `delete`.

□

Bibliographie

[BBC+11] Mohamed-Amine Baazizi, Nicole Bidoit, Dario Colazzo, Noor Malla, and Marina Sahakyan, *Projection for XML update optimization*, EDBT'11, 2011.

[BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom, *Models and issues in data stream systems*, PODS (Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, eds.), ACM, 2002, pp. 1–16.

[BBFV05] Michael Benedikt, Angela Bonifati, Sergio Flesca, and Avinash Vyas, *Adding updates to xquery : Semantics, optimization, and static analysis*, XIME-P (Daniela Florescu and Hamid Pirahesh, eds.), 2005.

[BBTC11] Mohamed Amine Baazizi, Nicole Bidoit-Tollu, and Dario Colazzo, *Efficient encoding of temporal xml documents*, TIME (Carlo Combi, Martin Leucker, and Frank Wolter, eds.), IEEE, 2011, pp. 15–22.

[BC09] Michael Benedikt and James Cheney, *Semantics, types and effects for xml updates*, DBPL (Philippa Gardner and Floris Geerts, eds.), Lecture Notes in Computer Science, vol. 5708, Springer, 2009, pp. 1–17.

[BC10] ———, *Destabilizers and independence of xml updates*, PVLDB **3** (2010), no. 1, 906–917.

[BCCN06] Véronique Benzaken, Giuseppe Castagna, Dario Colazzo, and Kim Nguyen, *Type-based XML projection*, VLDB, 2006.

[BCCN11] ———, *Optimizing xml querying using type-based document projection*, CoRR **abs/1104.2079** (2011).

[BCF+07] Béatrice Bouchou, Ahmed Cheriat, Mírian Halfeld Ferrari, Dominique Laurent, Maria Adriana Lima, and Martin A. Musicante, *Efficient constraint validation for updated xml database*, Informatica (Slovenia) **31** (2007), no. 3, 285–309.

[BCL+05] S. Bressan, B. Catania, Z. Lacroix, Y-G Li, and A. Maddalena, *Accelerating queries by pruning XML documents.*, Data Knowl. Eng. **54** (2005), no. 2.

[BDF+01] Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan, *Keys for xml*, WWW, 2001, pp. 201–210.

[BDF+03] ———, *Reasoning about keys for xml*, Inf. Syst. **28** (2003), no. 8, 1037–1063.

[BGvK+06] Peter A. Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner, *Monetdb/xquery : a fast xquery processor powered by a relational engine*, SIGMOD Conference (Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, eds.), ACM, 2006, pp. 479–490.

[BK08] Michael Benedikt and Christoph Koch, *Xpath leashed*, ACM Comput. Surv. **41** (2008), no. 1.

[BK09] ———, *From xquery to relational logics*, ACM Trans. Database Syst. **34** (2009), no. 4.

[BKTT04a] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang-Chiew Tan, *Archiving scientific data*, ACM Transactions on Database Systems (2004).

[BKTT04b] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang Chiew Tan, *Archiving scientific data*, ACM Trans. Database Syst. **29** (2004), 2–42.

[BO07] Nicole Bidoit and Matthieu Objois, *Sqtl : A preliminary proposal for a temporal-to-temporal query language*, TIME, IEEE Computer Society, 2007, pp. 35–46.

[BSS96] Michael H. Böhlen, Richard T. Snodgrass, and Michael D. Soo, *Coalescing in temporal databases*, VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India (T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, eds.), Morgan Kaufmann, 1996, pp. 180–191.

[BYFJ04] Ziv Bar-Yossef, Marcus Fontoura, and Vanja Josifovski, *On the memory requirements of XPath evaluation over XML streams*, PODS (Alin Deutsch, ed.), ACM, 2004, pp. 177–188.

[CAH02] Grégory Cobéna, Talel Abdessalem, and Yassine Hinnach, *A comparative study for XML change detection*, Tech. report, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France, July 2002.

[CAM02] Gregory Cobena, Serge Abiteboul, and Amélie Marian, *Detecting changes in xml documents*, ICDE (Rakesh Agrawal and Klaus R. Dittrich, eds.), IEEE Computer Society, 2002, pp. 41–52.

[CAW99] Sudarshan S. Chawathe, Serge Abiteboul, and Jennifer Widom, *Managing historical semistructured data*, Theory and Practice of Object Systems **5** (1999), no. 3, 143–162.

[CGM11] Federico Cavalieri, Giovanna Guerrini, and Marco Mesiti, *Dynamic reasoning on xml updates*, EDBT (Anastasia Ailamaki, Sihem Amer-Yahia, Jignesh M. Patel, Tore Risch, Pierre Senellart, and Julia Stoyanovich, eds.), ACM, 2011, pp. 165–176.

[COQ04] Carlo Combi, Barbara Oliboni, and Elisa Quintarelli, *A graph-based data model to represent transaction time in semistructured data*, DEXA (Fernando Galindo, Makoto Takizawa, and Roland Traunmüller, eds.), Lecture Notes in Computer Science, vol. 3180, Springer, 2004, pp. 559–568.

[CT05] Jan Chomicki and David Toman, *Time in database systems*, Elsevier, 2005.

[CTZ00] Shu-Yao Chien, Vassilis J. Tsotras, and Carlo Zaniolo, *Version management of xml documents*, WebDB (Selected Papers) (Dan Suciu and Gottfried Vossen, eds.), Lecture Notes in Computer Science, vol. 1997, Springer, 2000, pp. 184–200.

[DBJ99] Curtis E. Dyreson, Michael H. Böhlen, and Christian S. Jensen, *Capturing and querying multiple aspects of semistructured data*, VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK (Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, eds.), Morgan Kaufmann, 1999, pp. 290–301.

[dom] *DOM*, http ://www.w3.org/DOM/.

[DTD] DTD, *Html 4.01 specification*.

[exi] *eXist*, `http://exist.sourceforge.net/`.

[FCB07] Wenfei Fan, Gao Cong, and Philip Bohannon, *Querying XML with update syntax*, SIGMOD Conference, 2007.

[Feg10] Leonidas Fegaras, *A schema-based translation of XQuery updates*, XSym, 2010.

[FHM+05] Mary F. Fernández, Jan Hidders, Philippe Michiels, Jérôme Siméon, and Roel Vercammen, *Optimizing sorting and duplicate elimination in xquery path expressions*, DEXA (Kim Viborg Andersen, John K. Debenham, and Roland Wagner, eds.), Lecture Notes in Computer Science, vol. 3588, Springer, 2005, pp. 554–563.

[gal] *Galax*, `http://www.galaxquery.org`.

[Gau09] Olivier Gauwin, *Flux xml, requêtes xpath et automates*, Ph.D. thesis, INRIA Lille - Nord Europe - MOSTRARE, 2009.

[GKP02] Georg Gottlob, Christoph Koch, and Reinhard Pichler, *Efficient algorithms for processing xpath queries*, VLDB, Morgan Kaufmann, 2002, pp. 95–106.

[GKPS05] Georg Gottlob, Christoph Koch, Reinhard Pichler, and Luc Segoufin, *The complexity of xpath query evaluation and xml typing*, J. ACM **52** (2005), no. 2, 284–335.

[GMN07] Wouter Gelade, Wim Martens, and Frank Neven, *Optimizing schema languages for XML : Numerical constraints and interleaving*, ICDT, 2007.

[GN11] Olivier Gauwin and Joachim Niehren, *Streamable fragments of forward XPath*, CIAA (Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, eds.), Lecture Notes in Computer Science, vol. 6807, Springer, 2011, pp. 3–15.

[Gra04] Fabio Grandi, *Introducing an annotated bibliography on temporal and evolution aspects in the world wide web*, SIGMOD Record **33** (2004), no. 2, 84–86.

[GRS06] Giorgio Ghelli, Christopher Re, and Jérôme Siméon, *Xquery ! : An xml query language with side effects*, EDBT Workshops (Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen, eds.), Lecture Notes in Computer Science, vol. 4254, Springer, 2006, pp. 178–191.

[GRS07] Giorgio Ghelli, Kristoffer Høgsbro Rose, and Jérôme Siméon, *Commutativity analysis in XML update languages*, ICDT, 2007.

[GRS08] ———, *Commutativity analysis for XML updates*, ACM Trans. Database Syst. **33** (2008), no. 4.

[GS03a] D. Gao and R. T. Snodgrass, *Syntax, semantics, and query evaluation of the txquery temporal xml query language*, Tech. report, TimeCenter, 2003.

[GS03b] Dengfeng Gao and Richard T. Snodgrass, *Temporal slicing in the evaluation of xml queries*, VLDB, 2003, pp. 632–643.

[KHH+09] Hiroyuki Kato, Soichiro Hidaka, Zhenjiang Hu, Yasunori Ishihara, and Keisuke Nakano, *Rewriting xquery to avoid redundant expressions based on static emulation of xml store*, 2009.

[MACM01] Amélie Marian, Serge Abiteboul, Gregory Cobena, and Laurent Mignet, *Change-centric management of versions in an xml warehouse*, VLDB (Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, eds.), Morgan Kaufmann, 2001, pp. 581–590.

[Mar05] Maarten Marx, *Conditional xpath*, ACM Trans. Database Syst. **30** (2005), no. 4, 929–959.

[MdR05] Maarten Marx and Maarten de Rijke, *Semantic characterizations of navigational xpath*, SIGMOD Record **34** (2005), no. 2, 41–46.

[MRV04] Alberto O. Mendelzon, Flavio Rizzolo, and Alejandro A. Vaisman, *Indexing temporal xml documents*, VLDB (Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, eds.), Morgan Kaufmann, 2004, pp. 216–227.

[MS03] A. Marian and J. Siméon, *Projecting XML documents.*, VLDB '03, 2003, pp. 213–224.

[mxq] *MXquery*, http ://mxquery.org/.

[Olt07] Dan Olteanu, *Forward node-selecting queries over trees*, ACM Trans. Database Syst **32** (2007), no. 1, 3.

[Pet05] Luuk Peters, *Change Detection in XML Trees : a Survey*, 3rd Twente Student Conference on IT, Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, June 2005.

[PHM+05] Wim Le Page, Jan Hidders, Philippe Michiels, Jan Paredaens, and Roel Vercammen, *On the expressive power of node construction in xquery*, WebDB (AnHai Doan, Frank Neven, Robert McCann, and Geert Jan Bex, eds.), 2005, pp. 85–90.

[qiza] *QizX Free-Engine-3.0*, `http://www.xmlmind.com/qizx/free_engine.html`.

[qizb] *QizX/open*, `http://www.xmlmind.com/qizx/qizxopen.shtml`.

[RV08] Flavio Rizzolo and Alejandro A. Vaisman, *Temporal XML : modeling, indexing, and query processing*, VLDB Journal : Very Large Data Bases **17** (2008), no. 5, 1179–1212.

[Sah11] Marina Sahakyan, *Main memory xml update optimization : algorithms and experiments.*, Ph.D. thesis, LRI - Laboratoire de Recherche en Informatique - INRIA Saclay - Ile de France - LEO, 2011.

[saxa] *SAX*, http ://www.saxproject.org/.

[saxb] *Saxon-ee*, `http://www.saxonica.com/`.

[SJ08] Sangchul Song and Joseph JaJa, *Archiving temporal web information : Organization of web contents for fast access and compact storage*, Tech. report, Institute for Advanced Computer Studies, Department of Electrical and Computer Engineering Institute for Advanced Computer Studies University of Maryland, College Park, 2008.

[SK95] Richard T. Snodgrass and Henry Kucera, *Rationale for a temporal extension to sql*, The TSQL2 Temporal Query Language, 1995, pp. 3–18.

[SWK+02] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse, *XMark : A benchmark for XML data management*, VLDB, 2002.

[Tic85] Walter F. Tichy, *Rcsa system for version control*, Softw. Pract. Exper. **15** (1985), no. 7, 637–654.

[TIHW01a] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld, *Updating xml*, SIGMOD Conference, 2001, pp. 413–424.

[TIHW01b] ———, *Updating xml*, SIGMOD Conference, 2001, pp. 413–424.

[Tom98] David Toman, *Point-based temporal extensions of SQL and their efficient implementation*, Temporal Databases : Research and Practice, Lecture Notes in Computer Science, vol. 1399, Springer-Verlag, 1998.

[TVB+02] Igor Tatarinov, Stratis Viglas, Kevin S. Beyer, Jayavel Shanmugasundaram, Eugene J. Shekita, and Chun Zhang, *Storing and querying ordered xml using a relational database system*, SIGMOD Conference (Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, eds.), ACM, 2002, pp. 204–215.

[WZ08] Fusheng Wang and Carlo Zaniolo, *Temporal queries and version management in XML-based document archives*, Data Knowl. Eng (2008).

[WZZ05] Fusheng Wang, Carlo Zaniolo, and Xin Zhou, *Temporal XML ? SQL strikes back !*, IEEE Computer Society, 2005.

[XDM] XDM, *Xquery 1.0 and xpath 2.0 data model (xdm) (second edition)*.

[XML] XML, *Extensible markup language (xml)1.0 (fifth edition)*.

[XPa] XPath, *Xquery 1.0 and xpath 2.0 data model (xdm) (second edition)*.

[XQu] XQuery, *Xquery 1.0 : An xml query language (second edition)*.

[XS] XPath-Semantics, *Xquery 1.0 and xpath 2.0 formal semantics (second edition)*.

[XSD] XSD, *Xml schema*.

[XSL] XSLT, *Xsl transformations (xslt) version 1.0*.

[XUP] XUP, *Xquery update facility 1.0*.