Université Joseph Fourrier
Département Informatique et Mathématiques Appliquées

Mémoire de Master de Recherche en Informatique
Filière : Systèmes et Logiciels

Décidabilité des requêtes XPath avec contraintes de comptage

Soutenu par

Nebil BEN MABROUK
le 18 juin 2007

Laboratoire d'accueil :
INRIA Rhône-Alpes

Responsable du projet :
Nabil LAYAÏDA

Composition du jury :
Jean Marc VINCENT
Rachid ECHAHED
Laurent BESACIER
Nabil LAYAÏDA
Jean-Yves VION-DURY

Grenoble, 2007

À ma famille..

À mes amis..

Et enfin à elle..

Remerciements

Je tiens à remercier Vincent Quint directeur de l'équipe WAM pour m'avoir accueilli dans son équipe.

Je témoigne ma profonde reconnaissance à Nabil Layaïda pour son encadrement et son soutient tout au long du projet.

Mes remerciements s'adressent également aux membres du jury, notamment à Jean-Marc Vincent, Rachid Echahed, Laurent Besacier et Jean-Yves Vion-Dury pour l'honneur qu'ils me portent en jugeant ce travail.

Enfin, j'espère ne jamais manquer de gratitude envers tous ceux qui ont contribué à la réalisation de ce travail et en particulier Pierre Genevès pour son aide précieuse et pour l'intérêt qu'il a accordé à mon travail.

Résumé

L'émergence du langage XML comme standard pour la structuration et l'échange des données, a rendu nécessaire l'optimisation du langage XPath utilisé pour rechercher et extraire les informations dans les arbres XML.
C'est dans ce contexte qu'intervient le problème d'analyse statique d'une logique capable de supporter XPath, auquel se réduisent les problèmes d'inclusion, d'équivalence, de jointure et de satisfaisabilité des requêtes XPath. En fait, pouvoir décider ce type de problèmes en un temps raisonnable permet d'aider à manipuler avec sûreté et efficacité les données XML et peut contribuer à réduire considérablement l'évaluation des requêtes XPath.

Jusqu'à présent, les travaux menés par l'équipe WAM de l'INRIA ont permis de résoudre ces problèmes pour un fragment bien précis du langage XPath. Le but à long terme, est d'élargir progressivement le fragment étudié pour couvrir la totalité du langage XPath.

L'étape primordiale qui est traitée dans ce mémoire, est de considérer le problème de décidabilité pour les requêtes XPath avec contraintes de comptage.

Mots clés : XPath, contraintes de comptage, problème de décidabilité, μ-calcul, PDL.

Table des matières

Table des matières **3**

Table des figures **5**

Liste des tableaux **7**

1 Introduction **9**
1.1 Contexte, problématique et objectifs . . . 9
1.2 Introduction à XML et XPath . . . 10
1.2.1 XML . . . 10
1.2.2 XPath . . . 10
1.2.3 Requêtes XPath avec contraintes de comptage . . . 13
1.3 Organisation du rapport . . . 13

2 État de l'art **15**
2.1 Logiques pour XML . . . 16
2.1.1 Modal Fixpoint Logic . . . 16
2.1.2 Logique de Sheaves . . . 17
2.1.3 EXML . . . 19
2.1.4 μ- calcul . . . 20
2.1.5 Fully Enriched μ-calculus . . . 23
2.1.6 CXPath . . . 25
2.2 Logiques qui comptent . . . 28
2.2.1 Arithmétique de Presburger . . . 28
2.2.2 PDL (Propositional Dynamic Logic) . . . 29
2.3 Synthèse des travaux liés . . . 31

3 Logique pour XPath avec comptage **33**
3.1 Restriction de comptage . . . 34
3.2 Automates d'arbre et arbres binaires . . . 34
3.3 Syntaxe abstraite de XPath . . . 34
3.4 Dynamic μ-calculus . . . 35
3.4.1 Langage de base . . . 35
3.4.2 Formules de haut niveau . . . 36
3.5 Deux solutions de comptage . . . 37
3.5.1 Conditions de comptage . . . 37
3.5.2 Aperçu des approches proposées . . . 37
3.6 Deux sémantiques de traduction . . . 38

4 Décomposition de comptage et ordre relatif **39**
4.1 Exigences de comptage . . . 40
4.2 Sémantique de navigation ou sémantique de sélection ? . . . 40
4.3 Translation des concepts XPath en PDL . . . 40
4.3.1 Interprétation logique des axes . . . 40
4.3.2 Interprétation logique des expressions . . . 40
4.3.3 Interprétation logique des qualificateurs . . . 41
4.3.4 Illustration par l'exemple . . . 43
4.4 Traduction des formules PDL en μ-calcul . . . 44
4.4.1 Traduction des axes . . . 44
4.4.2 Traduction des formules génériques . . . 45
4.4.3 Traduction des contraintes de comptage . . . 45
4.4.4 Traduction des contraintes modulo . . . 49
4.4.5 Illustration par l'exemple . . . 50

5 Comptage global et ordre du document **53**
5.1 Exigences de comptage . . . 54
5.1.1 La monotonie du chemin . . . 54
5.1.2 Comptage ordonné . . . 54
5.2 Translation des concepts XPath en PDL . . . 56
5.3 Traduction des formules PDL en μ-calcul . . . 57
5.3.1 Détection des cycles de comptage . . . 57
5.3.2 Dualité de comptage . . . 57
5.3.3 Traduction des axes et des formules génériques . . . 58
5.3.4 Traduction des contraintes de comptage . . . 58
5.3.5 Traduction des contraintes modulo . . . 59
5.4 Synthèse des approches proposées . . . 60
5.4.1 Récapitulatif des deux approches . . . 60
5.4.2 Comparaison . . . 60

6 Algorithme de décision et implémentation **63**
6.1 Procédure de décision . . . 64
6.1.1 Translation des concepts XPath . . . 64
6.1.2 Algorithme de décision . . . 64
6.2 Implémentation et test . . . 65
6.2.1 Vue d'ensemble . . . 65
6.2.2 Détails de l'implémentation . . . 67
6.2.3 Test et illustration . . . 71

7 Conclusion **75**
7.1 Rappel des objectifs . . . 75
7.2 Travail réalisé . . . 75
7.3 Principales contributions . . . 76
7.4 Perspectives . . . 76

Bibliographie **77**

Table des figures

2.1 Syntaxe de la logique de Sheaves . . . 17
2.2 Formules de base . . . 21
2.3 Formules sans cycles . . . 21
2.4 Concepts XPath : Translation des expressions, des chemins et des qualificateurs . . . 22
2.5 Concepts XPath : Translation des axes . . . 23
2.6 Axiomes de la logique PDL . . . 29

3.1 Dynamic μ-calculus . . . 33
3.2 Arbre d'arité non bornée et arbre binaire équivalent . . . 34
3.3 Syntaxe des expressions XPath . . . 35
3.4 Langage de base . . . 36
3.5 Formules de haut niveau . . . 36

4.1 Translation des axes . . . 41
4.2 Translation des expressions et des chemins . . . 41
4.3 Translation des qualificateurs . . . 42
4.4 Dualité des contraintes de comptage . . . 43
4.5 Exemple de translation XPath . . . 44
4.6 Exemple de translation de PDL vers μ-calcul . . . 51

5.1 Relation de précédence selon l'ordre du document . . . 55
5.2 Translation des axes . . . 56
5.3 Translation des expressions et des chemins . . . 56
5.4 Translation des qualificateurs . . . 57
5.5 Position des deux approches par rapport à la complexité du comptage . . . 62

6.1 Vue d'ensemble du compilateur XPath . . . 68
6.2 Code Java de la classe XPathClosure . . . 68
6.3 Le paquetage "formulas" . . . 69
6.4 Le paquetage "xpath2pdl" . . . 69
6.5 'Parsing' des requêtes XPath . . . 69
6.6 Analyse statique des concepts XPath . . . 70
6.7 Translation des concepts XPath en PDL . . . 70
6.8 Test unitaire de la translation des concepts XPath en PDL . . . 72
6.9 Translation d'une requête XPath en PDL . . . 73
6.10 Translation de la formule PDL en μ-calcul . . . 73
6.11 Satisfaisabilité de la requête XPath . . . 74

Liste des tableaux

1.1 Description des axes XPath 12

2.1 Résultats établis pour la complexité des variantes du μ-calcul enrichi 24
2.2 Sémantique de CXPath 26
2.3 Synthèse des logiques pour XML 31

4.1 Traduction des primitives de navigation 45
4.2 Traduction des formules génériques 45
4.3 Traduction des formules de comptage selon l'opérateur $>$ 46
4.4 Traduction des formules de comptage selon l'opérateur $\geq$ 47
4.5 Traduction des formules de comptage selon l'opérateur $=$ 48
4.6 Traduction des formules de comptage selon l'opérateur modulo 49
4.7 Illustration (détails de la traduction) 50

5.1 Grille de détection des cycles 54
5.2 Traduction des primitives de navigation 58
5.3 Traduction des formules génériques 58
5.4 Récapitulatif des deux approches 60
5.5 Comparaison des deux approches 60

Chapitre 1

Introduction

D'abord, nous introduisons le contexte et la problématique de notre sujet et nous expliquons nos objectifs. Nous donnons par la suite une brève introduction à XML et XPath en mettant l'accent sur les requêtes XPath avec contraintes de comptage.

1.1 Contexte, problématique et objectifs

Le constat qui motive ce travail de recherche est l'explosion du nombre d'applications qui produisent, consomment et manipulent des données sous format XML. Dans de nombreux cas, il s'agit d'applications Web qui traitent des données assez volumineuses et qui nécessitent un temps de réponse assez réduit, d'où la nécessité d'optimiser le langage XPath[18], standard recommandé par le W3C pour l'exploitation des données XML[17] , et par conséquent, d'optimiser le traitement des requêtes.

Ces besoins expliquent l'importance de résoudre les problèmes de décision pour le langage XPath, y compris les problèmes d'inclusion, d'équivalence, de jointure et de satisfaisabilité des requêtes. En fait, décider de ces problèmes dans un temps raisonnable permet d'améliorer l'efficacité et la performance des langages de programmation utilisant XPath.

Jusqu'à présent, les travaux menés autour de ce sujet ont réussi à établir une logique décidable pour le fragment Core XPath[10]. Un des problèmes qui restent à résoudre, c'est d'élargir ce fragment et considérer les requêtes XPath avec contraintes de comptage.
Notre projet se propose d'étudier les théories et les logiques qui sont susceptibles de résoudre le problème de décision pour les requêtes XPath avec comptage. Ces formalismes doivent satisfaire deux exigences : avoir une expressivité assez riche qui permet d'exprimer les contraintes de comptage, et avoir une complexité assez réduite permettant de définir une procédure de décision efficace sur le plan pratique et par conséquent, répondre aux exigences de sureté et d'efficacité lors de la manipulation des données XML.

Notre ambition est de définir une logique pour le langage XPath, qui est décidable en un temps raisonnable, et qui est d'expressivité assez riche pour exprimer les requêtes avec contraintes de comptage.

1.2 Introduction à XML et XPath [2]

1.2.1 XML

Le langage XML, acronyme de eXtensible Markup Language, est un format textuel qui permet de créer des documents contenant des données structurées. Une de ses caractéristiques est d'être auto-descriptif : les données ont une structure d'arbre contenant des balises qui informent sur la structure et la sémantique des documents. Conçu à l'origine comme un dérivé simple et flexible de la norme SGML, le format XML joue un rôle croissant dans l'échange d'informations sur le Web et s'est imposé comme un pilier de l'initiative du Web Sémantique.

La force de XML réside dans sa capacité à pouvoir décrire n'importe quel domaine de données grâce à son extensibilité. C'est un méta-langage qui permet de structurer et de normaliser un ensemble de données contenues dans un document. Dans un document XML, la mise en forme des données est totalement séparée des données elles-mêmes. Les balises XML décrivent le contenu plutôt que la présentation des données. Ce mécanisme permet de gérer l'affichage d'un même document sur des applications ou des périphériques différents sans pour autant nécessiter de créer autant de versions du document que de représentations nécessaires. La mise en page des données est assurée par un langage de mise en page tiers comme, par exemple le langage XSLT (eXtensible StyleSheet Language Transformation). Les feuilles de style XSLT gèrent l'affichage des données, et permettent également d'effectuer des actions de sélection sur des fragments du document. Pour accéder aux différents éléments de contenu du document XML, ces feuilles de style utilisent le langage d'extraction de chemins XPath dont nous verrons les détails dans le paragraphe suivant.

1.2.2 XPath

XPath est une spécification permettant de décrire un modèle de parcours dans l'arbre d'un document XML à l'aide d'une expression de chemin. Cette technique vise à exploiter les relations logiques existant entre les noeuds qui composent le document afin de sélectionner l'ensemble des noeuds atteints en suivant tous les chemins conformes au modèle donné. Ce langage est utilisé dans de nombreux outils de la galaxie XML et notamment, dans XSL et XSLT pour localiser un noeud précis ou un ensemble de noeuds afin de leur associer une présentation ou une transformation. Il est également utilisé dans XPointer et XLink pour identifier des fragments de documents et pour pointer une cible précise dans un document. Enfin, dans les langages de requêtes XML, comme XQuery, nous l'utilisons pour localiser un noeud bien précis ou un ensemble de noeuds afin de les post-traiter.

1.2.2.1 Modèle XPath

Avec le langage XPath, la localisation des fragments documentaires peut se faire par adressage absolu aussi bien que par filtrage. Dans le premier cas, cela implique la connaissance parfaite de la structure manipulée pour pouvoir exprimer le chemin complet conduisant aux fragments à atteindre ; mais dans le second cas, une connaissance partielle de la forme globale du chemin et du contenu des noeuds explorés suffit pour atteindre des composants dont nous connaissons pas a priori la position. Avec XPath, tous les composants d'un document sont traités de manière homogène, qu'ils soient de niveau élémentaire, attributs ou bien feuilles (noeuds texte). Nous pourrons donc atteindre et contraindre :

- Un composant (élément ou attribut)
- Son modèle de contenu
- Le chemin pour atteindre un composant

XPath établit un arbre de noeuds correspondant au document XML. Les noeuds de cet arbre peuvent être de plusieurs types prédéfinis et sont ordonnés selon l'ordre de lecture des constituants du document XML. Le noeud ***Document*** est un noeud racine unique et obligatoire distinct de l'élément racine du document.

Un noeud ***Element*** est associé à chaque élément du document XML. Il est étiqueté avec le nom de l'élément et peut contenir d'autres noeuds éléments et/ou des noeuds textes. Le contenu de l'élément est ordonné. Un noeud ***Attribut*** est également créé pour chaque attribut rencontré dans le document. Au niveau de l'arbre XPath, il est associé au noeud élément qu'il caractérise. Il est étiqueté avec le nom de l'attribut et contient sa valeur. Il n'existe pas d'ordre entre les noeuds attributs d'un même élément. Les noeuds ***Text*** correspondent aux feuilles de l'arbre XML, c'est-à-dire au contenu entre les balises, et sont étiquetés par la valeur de ce contenu. Enfin les noeuds ***ProcessingInstruction*** servent à stocker les instructions de traitement et les noeuds ***Comment*** permettent de classer les commentaires.

1.2.2.2 Expression de chemins

La syntaxe de base du langage XPath est fondée sur l'utilisation d'expressions. Une expression XPath s'évalue en fonction d'un noeud contexte et désigne un ou plusieurs chemins dans l'arbre à partir de ce noeud. Le résultat de l'évaluation de cette expression renverra alors soit un ensemble de noeuds, soit une valeur numérique, booléenne ou alphanumérique. Chaque évaluation d'expression dépend du contexte courant. Une des expressions les plus importantes dans le standard XPath est le chemin de localisation qui sélectionne un ensemble de noeuds à partir d'un noeud contextuel.
Un chemin XPath peut être de type absolu ou relatif. S'il est absolu, il commence toujours par le signe / qui permet d'indiquer la racine du document XML. S'il est relatif, le noeud de départ est le noeud contextuel courant. Le chemin de parcours du document est constitué de différentes étapes séparées par un séparateur /, il est de la forme suivante : `Noeud contexte (= Étape 0)/Étape1/Étape2/ ...`

Chacune de ces étapes est elle-même décomposable en trois parties :

- Un axe qui permet de définir le sens de la relation entre le noeud courant et le jeu de noeuds à localiser.
- Un filtre de noeud qui spécifie le type de noeud à localiser.
- Un ensemble de qualificateurs (facultatif) qui permettent d'affiner la recherche sur le jeu de noeuds à récupérer.

La syntaxe d'écriture est la suivante : `Axe :: filtre [qualificateurs]`.

1.2.2.3 Axes

Les axes XPath définissent les primitives de navigation pour le parcours des structures XML. Le tableau 1.1 donne une description détaillée des différents axes.
Les axes les plus utilisés ont une convention d'écriture abrégée :

- `.` est la forme abrégée de l'axe `self : :node()`
- `..` est la forme abrégée de l'axe `parent : :node()`
- `nom` est la forme abrégée de l'axe `child : :nom`
- `@nom` est la forme abrégée de l'axe `attribute : :nom`
- `//` est la forme abrégée de l'axe `/descendant-or-self : :node()`

Dans la suite du document, nous utilisons la notation complète pour écrire les requêtes XPath.

Axe	Signification
`self`	contient seulement le noeud contextuel
`child`	contient les enfants directs du noeud contextuel
`attribute`	contient les attributs du noeud contextuel
`parent`	contient le parent du noeud contextuel
`descendant`	contient les descendants du noeud contextuel. Un descendant peut être un fils, un petit-fils,...
`descendant-or-self`	contient le noeud contextuel et ses descendants
`ancestor`	contient les ancêtres du noeud contextuel. Cela comprend le noeud père, le noeud grand-père,...
`ancestor-or-self`	contient le noeud contextuel et ses ancêtres
`following`	contient tous les noeuds qui seront parcourus après le noeud contextuel
`following-sibling`	contient le noeud frère suivant du noeud contextuel
`preceding`	contient tous les noeuds qui ont été parcourus avant le noeud contextuel
`preceding-sibling`	contient le noeud frère précédent du noeud contextuel

TAB. 1.1: Description des axes XPath

1.2.2.4 Filtres de noeuds

Il existe deux manières pour filtrer les noeuds : par leur nom ou par leur type. Le filtrage sur le nom n'est possible que sur les types de noeuds possédant un nom (Element, ProcessingInstruction et Attribute) et s'effectue simplement en indiquant ce nom. Le filtrage sur le type de noeud permet de sélectionner tous les noeuds (indépendamment de leur nom) qui correspondent à un certain type. Il existe quatre types de filtre :

- text() : filtre seulement les noeuds de type Text
- comment() : filtre les noeuds de type Comment
- processing-instruction() : noeuds de type ProcessingInstruction
- node() : tous les types de noeuds

Exemples :

- child : :a/descendant : :b

  renvoie les noeuds b descendants d'un noeud a lui-même fils du noeud contexte.
- self : :a/descendant : :text()

  renvoie tous les noeuds de type Text descendants du noeud contexte a.

1.2.2.5 Les qualificateurs

Les qualificateurs sont des expressions booléennes construites à partir d'expressions de chemin et de fonctions prédéfinies. Le résultat de l'évaluation d'un qualificateur renvoie forcément un booléen. Pour cela, l'expression calculée est automatiquement convertie en un résultat booléen. Ainsi, si le résultat

de l'expression est une valeur numérique qui vaut 0 ou Null, le qualificateur prendra la valeur false. Toute autre valeur numérique renverra true. Si le résultat de l'expression est une chaîne de caractères, le qualificateur vaudra false si elle est vide et true sinon. Enfin, si le résultat de l'expression doit renvoyer un ensemble de noeuds, le qualificateur prendra la valeur false seulement si cet ensemble est vide. Ainsi, par exemple, le qualificateur `[attribute : :att1]` renverra vrai seulement si l'élément auquel nous l'appliquons possède un attribut `att1`.

Exemples : voici quelques exemples de qualificateurs basés sur des expressions de chemins

- child : :a / child : :b [attribute : :c]

  renvoie les éléments b, fils d'éléments a et qui répondent vrai au qualificateur, c'est-à-dire qui possèdent un attribut c.
- child : :a / child : :b [child : :c]

  renvoie alors les éléments b, fils d'éléments a et qui possèdent des fils c.

Comme indiqué précédemment, ces qualificateurs peuvent également utiliser des fonctions prédéfinies de XPath pour vérifier des conditions sur les noeuds. Nous reviendrons sur ces fonctions plus tard.

Exemple des qualificateurs qui utilisent des fonctions

- child : :a / descendant : :text() [position()=1]

  renvoie le premier noeud de type Text descendant d'un élément a fils du noeud contexte.

1.2.3 Requêtes XPath avec contraintes de comptage

Dans les requêtes XPath, les contraintes de comptage sont généralement exprimées en utilisant les fonctions[22]**count()**, **last()** et **position()** figurant dans l'API core de ce langage. La fonction 'count', ayant comme argument une expression XPath, permet de compter le nombre de noeuds référencés. Les autres fonctions permettent de positionner un noeud par rapport à ses frères. La fonction 'position' retourne la position du noeud contextuel. La fonction 'last' retourne quant à elle la position du dernier noeud dans un ensemble de noeuds frères.

Les fonctions 'last()' et 'position()' peuvent être substituées par des expressions contenant la fonction 'count()' et qui ont la même sémantique. Dans la suite du document, nous nous intéressons seulement à la fonction 'count()'.

Exemples :

- /child : :a/child : :b[count(*)=2]

  Dans cet exemple, la fonction count retourne, à partir du noeud contexte, le noeud qui a exactement deux fils, d'où l'expression entière aura comme résultat, les noeuds b possédant deux fils, et dont le noeud parent est la racine a.
- count(/child : :a/child : :b)

  Dans le second exemple, la fonction count retourne le nombre des noeuds b ayant comme noeud parent la racine a.

Les deux exemples illustrent l'interférence de la fonction count avec d'autres expressions XPath pour apparaître dans différentes positions des requêtes, ce qui rend leurs traductions, à première vue, plus compliquées. Dans ce document, nous nous intéressons seulement au contraintes de comptage qualificatives, c'est-à-dire qui interviennent au niveau qualificateur, le premier exemple illustre ce cas d'utilisation.

1.3 Organisation du rapport

La suite du rapport se décline en quatre parties : d'abord, nous examinons l'état de l'art du sujet et nous étudions les travaux déjà effectués autour de ce problème, en particulier les travaux de l'équipe WAM de l'INRIA afin de les positionner par rapport à notre objectif, et envisager la possibilité de les considérer comme une base pour ce travail de recherche. Par la suite, nous présentons le fragment XPath à considérer dans ce travail et la logique proposée pour étudier sa décidabilité. En partant de cette logique, nous présentons dans la troisième partie (chapitres 5 et 6), les différentes approches proposées pour résoudre le problème de comptage. Dans la dernière partie du rapport, nous détaillons l'algorithme de décision de la logique considérée, et nous donnons également les détails de l'implémentation ainsi que quelques résultats expérimentaux. Enfin, le chapitre 7 conclut cette mémoire et donne de nouvelles perspectives.

Chapitre 2

État de l'art

Les recherches axées sur les problèmes d'analyse statique pour XML sont loin de la maturité, ce qui est accrédité par le nombre croissant des travaux dans ce domaine et par la grande distance qui existe encore entre le travail académique et les prototypes industriels.
La situation est encore plus sommaire lorsqu'il s'agit du langage XPath, qui a été créé dans l'esprit de s'accommoder d'une multitude d'utilisations distinctes, relatives à l'exploitation des données XML par différents langages et outils de l'univers XML.
Les travaux de recherche autour de ce sujet sont encore plus limités si nous considérons le problème du fragment XPath avec contraintes de comptage. Cet état des faits est en partie lié à l'ancienneté des logiques qui expriment les contraintes de comptage et à leur complexité accrue. Ces logiques sont entrain de regagner intérêt grâce à l'émergence de XML.
Le développement autour des logiques pour XPath avec comptage peut être grossièrement regroupé en deux catégories :

- Les logiques pour XML.
- Les logiques de comptage.

2.1 Logiques pour XML

Plusieurs logiques ont été proposées comme base théorique pour le langage XML. Dans ce qui suit nous présentons un aperçu des logiques les plus pertinentes à ce propos dont nous citons la logique modale des points fixes, la logique de sheaves, EXML, le μ-calcul, le μ-calcul étendu et CXPath. Pour chaque logique nous donnons sa syntaxe, son modèle de satisfaisabilité et sa décidabilité.

2.1.1 Modal Fixpoint Logic [15]

Ce formalisme est une logique de point fixe étendue par les contraintes de quantification numérique de l'arithmétique de Presburger.

2.1.1.1 Syntaxe

Les formules de la logique modale de point fixe sont de la forme suivante :
$\varphi ::= \top \mid \mu\, x.\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid a\langle F\rangle \mid *\langle F\rangle$
$F ::= r \mid \neg r \mid f$

avec :
$\varphi, \varphi_1, \varphi_2$: formules propositionnelles.
a : proposition atomique représentant le nom d'un noeud pour lequel a est vrai.
$*$: référence n'importe noeud étiqueté.
F : pré-condition sur les fils d'un noeud, cette condition peut être une expression régulière (r) ou une contrainte de Presburger (f).
➪ $|\varphi|$ indique le nombre de noeuds fils satisfaisant φ.

2.1.1.2 Modèle

La logique modale de point fixe admet comme modèle les automates PTA (Presburger Tree Automata) qui sont des automates d'arbre ordonnés, déterministes et d'arité non bornée.
Un automate PTA est un tuplet $(Q, \sum, \delta, T)$ ou Q est l'ensemble des états, $\sum$ un alphabet fini, δ une relation de transition et $T \subseteq Q$ l'ensemble des états accepteurs.

L'ensemble des arbres t sur l'alphabet $\sum$ par est défini par : $t ::= a\langle t_1, ..., t_K\rangle$, $k \geq 0$
Les relations de satisfaction $t \models_A q$ (pour les arbres t et les états q) et $u \models p$ (pour les séquences d'états $u \in Q^*$ et les pré-conditions p) sont définies par :
$t ::= a\langle t_1, ..., t_K\rangle \models_A q$ ssi $t_i \models_A q_i$ pour tout i et $q_1...q_k \models \delta(q, a)$
$q_1...q_k \models p_1 \vee p_2$ ssi $q_1...q_k \models p_1$ ou $q_1...q_k \models p_2$
$q_1...q_k \models p_1 \wedge p_2$ ssi $q_1...q_k \models p_1$ et $q_1...q_k \models p_2$
$q_1...q_k \models \neg p$ ssi $q_1...q_k \not\models p$
$q_1...q_k \models r$ ssi $q_1...q_k \in L(r)$
$q_1...q_k \models f$ ssi $\sigma \models f$, étant donné une fonction f et une attribution σ affectant des valeurs numériques aux variables de f, σ est dite satisfaisant f, *ssi* f est vrai pour σ, dénoté $\sigma \models f$.

2.1.1.3 Décidabilité

La satisfaisabilité des formules de la logique modale de point fixe sans quantification numérique peut être décidée en temps exponentiel (Exptime-complete). Par ailleurs, décider si un automate PTA A satisfait une formule φ est de complexité polynomiale selon la taille de φ $o(|t||\phi|^2)$ et linéaire selon la taille de A $o(|t||A|)$.

2.1.2 Logique de Sheaves [23]

La logique de Sheaves est une logique modale qui est proposée comme formalisme de base pour les langages de requête des données semi-structurées. Cette logique permet d'exprimer les propriétés des arbres ordonnés et non ordonnés, mais sa puissance expressive est limitée dans le sens ou elle ne possède pas d'opérateurs qui permettent la quantification numérique récursive sur les éléments d'arbres.

2.1.2.1 Formules

Les formules SL sont organisées selon l'ordre alphabétique A,B,..., elles sont données par une grammaire qui comprend trois catégories syntaxiques : Formules d'élément (*element formulas* ; E) qui expriment les propriétés d'un élément du document, les formules régulières (*regular formulas* ; S) qui correspondent aux expressions régulières sur les séquences d'éléments, les formules de comptage (*counting formulas* ; T) qui traduisent les contraintes de comptage sur un ensemble d'éléments. L'ensemble de ces formules est détaillé par la figure 2.1.

E	::=		élément
		$a[S]$	élément a avec une formule régulière S
	|	$a[T]$	élément a avec une formule de comptage T
	|	$AnyE$	correspond à n'importe quel élément
	|	`Datatype`	type de données constant
	|	cst	constante
S	::=		formule régulière
		ϵ	vide
	|	E	élément
	|	S, S'	composition séquentielle
	|	$S*$	répétition indéfinie
	|	$S \vee S$	disjonction
	|	$\neg S$	négation
T	::=		formule de comptage
		$\exists\, \mathbf{N}$: $\phi(\mathbf{N}) = N_1E_1 \,\&\, ... \,\&\, N_pE_p$	entrelacement généralisé($\mathbf{N} : (N_1, ..., N_p)$)
	|	$T \vee T$	disjonction
	|	$\neg T$	négation
$A, B, ...$	::=		formule
		S	formule régulière
	|	T	formule de comptage
	|	$A \vee A$	disjonction
	|	$\neg A$	négation

FIG. 2.1: Syntaxe de la logique de Sheaves

2.1.2.2 La relation de satisfaction

La relation $d \models A$ indique que le document d satisfait la formule A. Cette relation est définie inductivement sur la structure de A. Étant donné une formule A de la logique de Sheaves, le modèle de A est l'ensemble $Mod(A) = \{d | d \models A\}$ des documents qui satisfont A. Dans ce qui suit, ψ représente une formule de type S,T ou A.

$d \models a[\psi]$	ssi	$(d = a[d']) \wedge (d' \models \psi)$
$d \models AnyE$	ssi	$(d = a[d'])$
$d \models$ `Datatype`	ssi	$(d = cst) \wedge (cst \in$ `Datatype` $)$
$d \models cst$	ssi	$(d = cst)$
$d \models \epsilon$	ssi	$(d = \epsilon)$
$d \models S, S'$	ssi	$(d = d_1 \cdot d_2) \wedge (d_1 \models S) \wedge (d_2 \models S')$
$d \models S*$	ssi	$(d = \epsilon) \vee (d = d_1 \cdot d_2 ... d_p \wedge \forall i \in 1..p, d_i \models S)$
$d \models \exists \mathbf{N}$: $\phi(\mathbf{N}) = N_1E_1 \ \& \ N_2E_2 \ \& \ ... \ \& \ N_pE_p$	ssi	$\exists\, n_1..n_p, \exists (e_1^j)_{j\in 1..n_1}, ..., (e_p^j)_{j\in 1..n_p}$ $e_i^j \models E_i \ \wedge e_i^j \models \ \phi(n_1, .., n_p) \ \wedge \ d \in inter(e_1^1 \cdot ... \cdot e_p^{n_p})$
$d \models \psi \vee \psi'$	ssi	$(d \models \psi) \wedge (d \models \psi')$
$d \models \neg\psi$	ssi	not $(d \models \psi)$

2.1.2.3 Expressivité

Les principales adjonctions de la logique de Sheaves sont la modalité *AnyE* qui décrit un document avec un seul élément, et la quantification existentielle $\exists \mathbf{N}$: $\phi(\mathbf{N}) = N_1E_1 \ \& \ N_2E_2 \ \& \ ... \ \& \ N_pE_p$ qui est satisfaite par un document de $n_1 + ... + n_p$ éléments, avec n_i éléments qui correspondent à E_i indépendamment de leur ordre, et ainsi $(n_1, ..., n_p)$ satisfont la formule ϕ.
Cet opérateur d'entrelacement est utilisé pour exprimer plus de propriétés sur les documents. Par exemple, en utilisant la formule $\exists N_1, N_2 : (N_1 \geq 0) \wedge (N_2 = 1) : N_1E_1 \ \& \ N_2E_2$, nous pouvons définir l'expression $(E_1 * \& E_2)$ des documents qui contiennent plusieurs éléments satisfaisant E_1 et un élément unique qui satisfait E_2.

Exemple
La référence d'un livre est donnée par les champs "auteur", "titre" et "année de publication". Une collection est une séquence de références qui sont collectionnées à partir de plusieurs entrepôts de données hétérogènes ayant des structures différentes. Cela fait que l'ordre des champs n'est pas le même pour toutes les références. La formule suivante correspond aux collections des livres écrits par Engels et Marx et qui contiennent le même nombre de livres écrits par les deux auteurs.

collection [*livre* [*auteur* [`string`] & *titre* [`string`] & *année* [`string`]]*]
$\wedge$
collection [$\exists N, M : (N = M)$: N *livre* [*auteur* [Engels] & ...] & M *livre* [*auteur* [Marx] & ...]]

2.1.2.4 Décidabilité

Pour un entier $k \geq 1$ donné, SL_k dénote la restriction de la logique de Sheaves aux formules ayant au plus k occurrences des contraintes de régularité. SL_k est décidable en espace polynomial (PSPACE-complete), tandis que l'intégralité de la logique Sheaves SL est décidable avec un algorithme de complexité non-élémentaire.

2.1.3 EXML (EXtended Modal Logic)[7]

EXML est la logique conçue pour XML et construite à partir de l'extension de la logique modale par les contraintes de régularité et les contraintes de Presburger.

2.1.3.1 Syntaxe

Étant donné un ensemble de propositions atomiques AP= $\{p_1, p_2, ..\}$ et un ensemble de relations $\sum = \{R_1, R_2, ..\}$, les formules et les termes de la logique EXML sont définis inductivement comme suit :

$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid t \sim b \mid t \equiv_k c \mid \mathcal{A}(R, \phi_1, \phi_2, .., \phi_n)$
$t ::= a \times \sharp^R\phi \mid t + a \times \sharp^R\phi$

avec
- $p \in$ AP, $R \in \sum$,
- $b, k, c \in \mathbb{N}, a \in \mathbb{Z}$,
- $\sim \in \{=, <, >\}$,
- $\mathcal{A}$ est un automate fini non déterministe, défini sur un alphabet fini $\sum_{\mathcal{A}}$ dans lequel les lettres sont linéairement ordonnées $\sum_{\mathcal{A}} = a_1, a_2, .., a_n$. Le langage reconnu par $\mathcal{A}$ est noté $L_{\mathcal{A}}$.

notation
- $|\phi|$ désigne la taille de la formule EXML, tandis que $md(\phi)$ indique le degré modal de la formule défini comme le nombre maximal des occurrences de $\sharp$ dans ϕ.
- $\sum_i a_i\sharp^{R_i}\phi_i$ est l'abréviation des termes de la forme : $a_1\sharp^{R_1}\phi_1 + ... + a_m\sharp^{R_m}\phi_m$.
- $R_R(nd) = nd_1 < ... < nd_\alpha$ désigne $R_R(nd) =_{def} \{nd\prime \in T : \langle nd, nd\prime\rangle \in R_R\} = \{nd_1, ..., nd_\alpha\}$ et $nd_1 < ... < nd_\alpha$.
- Étant donné une relation binaire $R \subseteq T \times T$, $R^\sharp(q)$ indique la cardinalité de l'ensemble $\{q\prime \in T : \langle q, q\prime\rangle \in R\}$.

2.1.3.2 Modèle

Un modèle EXML est une structure $\mathcal{M} = \langle T, (R_R)_{R\in\sum}, (<^R_{nd})_{nd\in T}, l\rangle$ avec :
- T l'ensemble des noeuds,
- $(R_R)_{R\in\sum}$ l'ensemble des relations binaires dans $T \times T$, tel que pour tout $R \in \sum$ et $nd \in T$ l'ensemble $\{\ nd\prime \in T : \langle nd, nd\prime\rangle \in T\ \}$ est fini,
- chaque relation $<^R_{nd}$ est un ensemble totalement ordonné sur les R_R-successeurs de nd,
- $l : T \to 2^{AP}$ est une fonction d'évaluation.

Pour un modèle $\mathcal{M}$ et un noeud $nd \in T$, la relation de satisfaction est définie comme suit :
- $\mathcal{M}, nd \models \phi$ ssi $p \in l(nd)$,
- $\mathcal{M}, nd \models \neg\phi$ ssi not $\mathcal{M}, nd \models \phi$,
- $\mathcal{M}, nd \models \phi_1 \wedge \phi_2$ ssi $\mathcal{M}, nd \models \phi_1$ et $\mathcal{M}, nd \models \phi_2$,
- $\mathcal{M}, nd \models \sum_i a_i\sharp^{R_i}\phi_i \ \sim\ b$ ssi $\sum_i a_i R^\sharp_{R_i,\phi_i}(nd) \sim b$ avec $R_{R_i,\phi_i} = \{\langle nd', nd''\rangle \in T \times T : \langle nd', nd''\rangle \in R_{R_i}$, et $\mathcal{M}, nd'' \models \phi_i\}$,
- $\mathcal{M}, nd \models \sum_i a_i\sharp^{R_i}\phi_i \ \equiv_k\ b$ ssi $n \in \mathbb{N}$ tel que $\sum_i a_i R^\sharp_{R_i,\phi_i}(nd) = nk + c$,
- $\mathcal{M}, nd \models \mathcal{A}(R, \phi_1, \phi_2, .., \phi_n)$ ssi il existe $a_{i_1}...a_{i_\alpha} \in L_{\mathcal{A}}$ tel que $R_R(nd) = nd_1 < ... < nd_\alpha$ et pour tout $j \in \{1, .., \alpha\}$ $\mathcal{M}, nd_j \models \phi_{i_j}$.

2.1.3.3 Décidabilité

– Pour tout $k \leq 0$, EXML$_k$ désigne le fragment EXML limité aux formules avec au plus k contraintes de régularité et sans contraintes de Presburger. Ce fragment est décidable en espace polynomial

(PSPACE-complete).
- PML, le fragment EXML des formules sans contraintes de régularité et avec l'intégralité des contraintes de Presburger, est décidable en espace polynomial (PSPACE-complete).
- RML, le fragment EXML des formules sans contraintes de Presburger, est décidable en espace polynomial (PSPACE-complete).
- L'intégralité de la logique EXML est décidable en espace exponentiel (EXPSPACE).

2.1.4 μ- calcul : une logique temporelle de point fixe pour XML [10]

L'équipe WAM de l'INRIA a proposé une logique temporelle de point fixe comme logique unificatrice pour XML. Cette logique qui dérive du μ-calcul* et hérite de ses avantages, a permis de réduire encore plus la meilleure complexité déjà établie pour les problèmes de typage XML.

2.1.4.1 Arbres ciblés

Les arbres ciblés est une approche moins conventionnelle qui a été choisie pour modéliser les arbres XML. Ces structures décrivent les propriétés des arbres mais aussi leur contexte, ainsi que les noeuds précédents, suivants et les noeuds parents.
Exploiter ce type de structure présente l'avantage de préserver la totalité des informations sur l'arbre, ce qui est utile dans le contexte du langage XPath.

t	$::=$	$\sigma[tl]$	arbre
tl	$::=$		liste d'arbres
	$::=$	ϵ	liste vide
	$\mid$	$t :: tl$	construction d'un élément
c	$::=$		contexte
		(tl, Top, tl)	racine de l'arbre
	$\mid$	$(tl, c[\sigma], tl)$	noeud contexte
f	$::=$	(t, c)	arbre ciblé

Un ***signe contextuel*** a été introduit pour résoudre le problème d'inclusion des requêtes XPath. Ce signe noté $\sigma^{\bullet}[tl]$, indique le noeud où l'évaluation à commencé. Lorsque la présence du signe est inconnu, l'arbre est noté $\sigma^{\circ}[tl]$.
Étant donné un ensemble $\mathcal{F}$ d'arbres ciblés portant le signe contexte, et une fonction $\mathtt{nm}$ attribuant à un arbre son nom (exemple $\mathtt{nm}(\sigma^{\circ}[tl], c) = \sigma$), la navigation dans un arbre ciblé est donnée par les formules ci-dessous.

$$
\begin{array}{lll}
(\sigma^{\circ}[t :: tl], c)\ \langle 1\rangle & =_{def} & (t, (\epsilon, c[\sigma^{\circ}], tl)) \\
(t, (tl_l, c[\sigma^{\circ}], t' :: tl_r))\ \langle 2\rangle & =_{def} & (t', (t :: tl_l, c[\sigma^{\circ}], tl_r)) \\
(t, (\epsilon, c[\sigma^{\circ}], tl))\ \langle \overline{1}\rangle & =_{def} & (\sigma^{\circ}[t :: tl], c) \\
(t', (t :: tl_l, c[\sigma^{\circ}], tl_r))\ \langle \overline{2}\rangle & =_{def} & (t, (tl_l, c[\sigma^{\circ}], t' :: tl_r))
\end{array}
$$

2.1.4.2 Formules

Nous introduisons la logique dans laquelle les expressions XPath sont traduites. Cette logique consiste en un sous-ensemble du μ-calcul sans alternance et avec programmes inverses. Nous donnons par la suite une restriction de cette logique comprenant uniquement les formules sans cycles.
La traduction des expressions XPath comprend quatre étapes : d'abord elles sont traduites en formules de bases (données par la figure 2.2), elles sont transformées par la suite en formules sans cycles (définies dans la figure 2.3), puis elles sont écrites sous la forme négationelle. La dernière étape consiste à tester

*μ-calcul est une logique modale propositionnelle étendue par le plus petit et le plus grand point fixe, elle est utilisée comme formalisme cible pour l'incorporation des logiques temporelles et modales ayant pour but la description des propriétés des systèmes.

leur satisfaisabilité via une procédure de décision.

Les formules sans cycles doivent satisfaire deux contraintes : les variables des points fixes doivent ap-

$\mathcal{L}_\mu \ni \varphi, \psi$	::=			formule
	$\top$			vrai
|	σ	| $\neg\sigma$		proposition atomique (négation)
|	$\gamma^\bullet$	| $\neg\gamma^\bullet$		contexte (négation)
|	X			variable
|	$\varphi \vee \psi$			disjonction
|	$\varphi \wedge \psi$			conjonction
|	$\langle a \rangle \varphi$	| $\langle a \rangle \neg\varphi$		existentiel (négation)
|	$\mu\overline{X_i.\varphi_i}$ *in* ψ			plus petit point fixe
|	$\nu\overline{X_i.\varphi_i}$ *in* ψ			plus grand point fixe

FIG. 2.2: Formules de base

paraître sous au moins un opérateur existentiel. En plus, entre une variable et son point fixe, les opérateurs existentiels ne doivent pas contenir des programmes inverses tels que $\langle a \rangle$ et $\langle \overline{a} \rangle$.

Dans l'évaluation $\Delta \| \Gamma \vdash_I^R \varphi$, Δ désigne l'environnement d'affectation des variables de récursion à leurs formules, et l'environnement Γ lie les variables aux modalités. Une modalité peut avoir trois valeurs possible : $\_$ (aucune information sur la variable), $\langle a \rangle$ (la dernière modalité consistante) ou $\perp$ (aucune modalité : un cycle a été détecté). R représente l'ensemble des variables déjà parcourues et I les variables qui restent à vérifier.

Une formule est dite sans cycle si aucune variable de point fixe n'est sous la modalité ($\Gamma(X) = \_$), ou sous un cycle ($\Gamma(X) = \perp$).

$$\frac{\varphi = \top, \sigma, \neg\sigma, \gamma^\bullet, ou\ \neg\gamma^\bullet}{\Delta \| \Gamma \vdash_I^R \varphi} \qquad \frac{\Delta \| \Gamma \vdash_I^R \varphi \quad \Delta \| \Gamma \vdash_I^R \psi}{\Delta \| \Gamma \vdash_I^R \varphi \vee \psi}$$

$$\frac{\Delta \| \Gamma \vdash_I^R \varphi \quad \Delta \| \Gamma \vdash_I^R \psi}{\Delta \| \Gamma \vdash_I^R \varphi \wedge \psi} \qquad \frac{}{\Delta \| \Gamma \vdash_I^R \neg\langle a \rangle \top} \qquad \frac{\Delta \| (\Gamma \triangleleft \langle a \rangle) \vdash_I^R \varphi}{\Delta \| \Gamma \vdash_I^R \langle a \rangle \varphi}$$

$$\frac{\forall X_j \in (X_i \cap fv(\psi)).\ ((\Delta + \overline{X_i : \varphi_i}) \| (\Gamma + \overline{X_i : \_}) \vdash_{I \setminus \overline{X_i}}^{(R \setminus \overline{X_i}) \cup \{X_j\}} \varphi_j) \qquad \Delta \| \Gamma \vdash_{I \cup \overline{X_i}}^{R \setminus \overline{X_i}} \psi}{\Delta \| \Gamma \vdash_I^R \mu\overline{X_i.\varphi_i}\ in\ \psi}$$

$$\frac{\forall X_j \in (X_i \cap fv(\psi)).\ ((\Delta + \overline{X_i : \varphi_i}) \| (\Gamma + \overline{X_i : \_}) \vdash_{I \setminus \overline{X_i}}^{(R \setminus \overline{X_i}) \cup \{X_j\}} \varphi_j) \qquad \Delta \| \Gamma \vdash_{I \cup \overline{X_i}}^{R \setminus \overline{X_i}} \psi}{\Delta \| \Gamma \vdash_I^R \nu\overline{X_i.\varphi_i}\ in\ \psi}$$

$$\frac{X \in R \qquad \Gamma(X) = \langle a \rangle}{\Delta \| \Gamma \vdash_I^R X} \qquad \frac{X \notin R \qquad \Delta \| \Gamma \vdash_I^{R \cup \{X\}} \Delta(X)}{\Delta \| \Gamma \vdash_I^R X} \qquad \frac{X \in I}{\Delta \| \Gamma \vdash_I^R X}$$

FIG. 2.3: Formules sans cycles

2.1.4.3 Interprétation des formules

L'interprétation des formules du μ-calcul considéré comme des ensembles d'arbres ciblés, est donnée par les règles suivantes :

$$
\begin{array}{lll}
[\![\top]\!]_V & =_{def} & \mathcal{F} \\
[\![X]\!]_V & =_{def} & V(X) \\
[\![\varphi \vee \psi]\!]_V & =_{def} & [\![\varphi]\!]_V \cup [\![\psi]\!]_V \\
[\![\varphi \wedge \psi]\!]_V & =_{def} & [\![\varphi]\!]_V \cap [\![\psi]\!]_V \\
[\![\sigma]\!]_V & =_{def} & \{f \mid \mathtt{nm}(f) = \sigma\} \\
[\![\neg\sigma]\!]_V & =_{def} & \{f \mid \mathtt{nm}(f) \neq \sigma\} \\
[\![\gamma^\bullet]\!]_V & =_{def} & \{f \mid \mathtt{nm}(f) = (\sigma^\bullet[tl], c)\} \\
[\![\neg\gamma^\bullet]\!]_V & =_{def} & \{f \mid \mathtt{nm}(f) = (\sigma[tl], c)\} \\
[\![\langle a\rangle\varphi]\!]_V & =_{def} & \{f\langle\overline{a}\rangle \mid f \in [\![\varphi]\!] \wedge f\langle\overline{a}\rangle \text{ définie}\} \\
[\![\neg\langle a\rangle\top]\!]_V & =_{def} & \{f \mid f\langle a\rangle \text{ définie}\} \\
[\![\mu\overline{X_i.\varphi_i} \text{ in } \psi]\!]_V & =_{def} & \text{soit } T_i = \left(\bigcap\{\ \overline{T}_i \subseteq \overline{\mathcal{F}} \mid [\![\overline{\varphi}_i]\!]_{V[\overline{T_i/X_i}]} \subseteq \overline{T}_i\ \}\right)_i \text{ dans } [\![\psi]\!]_{V[\overline{T_i/X_i}]} \\
[\![\nu\overline{X_i.\varphi_i} \text{ in } \psi]\!]_V & =_{def} & \text{soit } T_i = \left(\bigcup\{\ \overline{T}_i \subseteq \overline{\mathcal{F}} \mid \overline{T}_i \subseteq [\![\overline{\varphi}_i]\!]_{V[\overline{T_i/X_i}]}\ \}\right)_i \text{ dans } [\![\psi]\!]_{V[\overline{T_i/X_i}]}
\end{array}
$$

2.1.4.4 Translation des concepts XPath

La logique proposée est suffisamment expressive pour exprimer la syntaxe XPath et en particulier la navigation bidirectionnelle dans les arbres ciblés. Les figures 2.4 et 2.5 donnent l'interprétation des concepts XPath dans cette logique.

$$
\begin{array}{lll}
S_e[\![\cdot]\!]. & : & \mathcal{L}_{\text{XPath}} \to 2^{\mathcal{F}} \to 2^{\mathcal{F}} \\
S_e[\![/p]\!]_F & =_{def} & S_p[\![p]\!]_{\text{root}(F)} \\
S_e[\![p]\!]_F & =_{def} & S_p[\![p]\!]_{\{(\sigma^\bullet[tl],c)\in F\}} \\
S_e[\![e_1|e_2]\!]_F & =_{def} & S_p[\![e_1]\!]_F \cup S_p[\![e_2]\!]_F \\
S_e[\![e_1 \cap e_2]\!]_F & =_{def} & S_p[\![e_1]\!]_F \cap S_p[\![e_2]\!]_F \\
 & & \\
S_p[\![\cdot]\!]. & : & Path \to 2^{\mathcal{F}} \to 2^{\mathcal{F}} \\
S_p[\![p_1/p_2]\!]_F & =_{def} & \{f' \mid f' \in S_p[\![p_2]\!]_{(S_p[\![p_1]\!]_F)}\} \\
S_p[\![p[q]]\!]_F & =_{def} & \{f \mid f \in S_p[\![p]\!]_F \wedge S_q[\![q]\!]_f\} \\
S_p[\![a :: \sigma]\!]_F & =_{def} & \{f \mid f \in S_a[\![a]\!]_F \wedge \mathtt{nm}(f) = \sigma\} \\
S_p[\![a :: *]\!]_F & =_{def} & \{f \mid f \in S_a[\![a]\!]_F\} \\
 & & \\
S_q[\![\cdot]\!]. & : & Qualif \to \mathcal{F} \to \{true, false\} \\
S_q[\![q_1 \textit{ and } q_2]\!]_f & =_{def} & S_q[\![q_1]\!]_f \wedge S_q[\![q_2]\!]_f \\
S_q[\![q_1 \textit{ or } q_2]\!]_f & =_{def} & S_q[\![q_1]\!]_f \vee S_q[\![q_2]\!]_f \\
S_q[\![not\ q]\!]_f & =_{def} & \neg\ S_q[\![q]\!]_f \\
S_q[\![p]\!]_f & =_{def} & S_p[\![p]\!]_{\{f\}} \neq \emptyset
\end{array}
$$

Fig. 2.4: Concepts XPath : Translation des expressions, des chemins et des qualificateurs

$$
\begin{array}{lll}
S_a[\![\cdot]\!]. & : & Axis \to 2^{\mathcal{F}} \to 2^{\mathcal{F}} \\
S_a[\![\text{self}]\!]_F & =_{def} & F \\
S_a[\![\text{child}]\!]_F & =_{def} & \texttt{fchild}(F) \cup S_a[\![\text{following} - \text{sibling}]\!]_{\text{fchild}(F)} \\
S_a[\![\text{following} - \text{sibling}]\!]_F & =_{def} & \texttt{nsibling}(F) \cup S_a[\![\text{following} - \text{sibling}]\!]_{\text{nsibling}(F)} \\
S_a[\![\text{preceding} - \text{sibling}]\!]_F & =_{def} & \texttt{psibling}(F) \cup S_a[\![\text{preceding} - \text{sibling}]\!]_{\text{psibling}(F)} \\
S_a[\![\text{parent}]\!]_F & =_{def} & \texttt{parent}(F) \\
S_a[\![\text{descendant}]\!]_F & =_{def} & S_a[\![\text{child}]\!]_F \cup S_a[\![\text{descendant}]\!]_{(S_a[\![\text{child}]\!]_F)} \\
S_a[\![\text{descendant}-\text{or}-\text{self}]\!]_F & =_{def} & F \cup S_a[\![\text{descendant}]\!]_F \\
S_a[\![\text{ancestor}]\!]_F & =_{def} & S_a[\![\text{parent}]\!]_F \cup S_a[\![\text{ancestor}]\!]_{(S_a[\![\text{parent}]\!]_F)} \\
S_a[\![\text{ancestor} - \text{or} - \text{self}]\!]_F & =_{def} & F \cup S_a[\![\text{ancestor}]\!]_F \\
S_a[\![\text{following}]\!]_F & =_{def} & S_a[\![\text{descendant} - \text{or} - \text{self}]\!]_{(S_a[\![\text{following}-\text{sibling}]\!]_{(S_a[\![\text{ancestor}-\text{or}-\text{self}]\!]_F)})} \\
S_a[\![\text{preceding}]\!]_F & =_{def} & S_a[\![\text{descendant} - \text{or} - \text{self}]\!]_{(S_a[\![\text{preceding}-\text{sibling}]\!]_{(S_a[\![\text{ancestor}-\text{or}-\text{self}]\!]_F)})} \\
 & & \\
\texttt{fchild}(F) & =_{def} & \{f\langle 1\rangle \mid f \in F \ et \ f\langle 1\rangle \ existe\} \\
\texttt{nsibling}(F) & =_{def} & \{f\langle 2\rangle \mid f \in F \ et \ f\langle 2\rangle \ existe\} \\
\texttt{psibling}(F) & =_{def} & \{f\langle \overline{2}\rangle \mid f \in F \ et \ f\langle \overline{2}\rangle \ existe\} \\
\texttt{parent}(F) & =_{def} & \{(\sigma^{\circ}[\texttt{rev\_a}(tl_l, t :: tl_r)], c) \mid (t, (tl_l, c[\sigma^{\circ}], tl_r)) \in F\} \\
 & & \\
\texttt{rev\_a}(\epsilon, tl_r) & =_{def} & tl_r \\
\texttt{rev\_a}(t :: tl_l, tl_r) & =_{def} & \texttt{rev\_a}(tl_l, t :: tl_r) \\
\texttt{root}(F) & =_{def} & \{(\sigma^{\bullet}[tl], (tl, Top, tl)) \in F\} \cup \texttt{root}(\texttt{parent}(F))
\end{array}
$$

FIG. 2.5: Concepts XPath : Translation des axes

2.1.4.5 Décidabilité

L'équipe WAM de l'INRIA a proposé un algorithme pour la décidabilité du μ-calcul considéré. La satisfaisabilité des formules de cette logique est résoluble en temps exponentiel simple en fonction de la taille de la formule ($2^{0(n)}$) où n est la taille de la formule[10].

2.1.5 Fully Enriched μ-calculus [4]

Le Fully Enriched μ-calculus est une logique résultante de l'extension du μ-calcul propositionnel par les programmes inverses, les symboles nominaux et des modalités étendues. Les programmes inverses permettent la navigation en arrière à travers les relations entre les états, les symboles nominaux sont des variables propositionnelles interprétées comme des sous-ensembles de singletons, tandis que les modalités étendues permettent les déclarations sur le nombre des successeurs et des prédécesseurs d'un état.

2.1.5.1 Syntaxe

Soit $AP, Var, Prog$ et Nom des ensembles finis deux à deux disjoints de propositions atomiques, de variables propositionnelles, de programmes atomiques et de symboles nominaux. Un programme atomique est un programme a ou son inverse $\overline{a}$. Les formules du Fully Enriched μ-calculus sont définies comme suit :

$$\varphi ::= true \mid false \mid p \mid \neg p \mid x \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \langle n, \alpha\rangle\varphi_1 \mid [n, \alpha]\varphi_1 \mid \mu y.\varphi_1(y) \mid \nu y.\varphi_1(y)$$

2.1.5.2 Sémantique

La sémantique du Fully Enriched μ-calculus est donnée par une structure de Kripke $K = \langle W, R, L\rangle$ où W représente un ensemble non vide d'états, et $R : Prog \to 2^{W\times W}$ une fonction qui affecte à chaque programme atomique une relation de transition sur W. La fonction $L : AP \cup Nom \to 2^W$ affecte à chaque

proposition atomique un ensemble d'états et à chaque symbole nominal un singleton.
Étant donné une structure de Kripke $K = \langle W, R, L\rangle$ et un ensemble $\{y_1, y_2, .., y_n\}$ de variables dans Var, une évaluation $\nu : \{y_1, y_2, .., y_n\} \to 2^W$ est une affectation des sous-ensemble de W aux variables $\{y_1, y_2, .., y_n\}$. Pour une évaluation ν, une variable y et un sous-ensemble $W' \subseteq W$, $\nu[y \to W']$ dénote l'évaluation obtenue par l'affectation de W' à y.
Une formule φ avec variables $y_1, y_2, .., y_n$ est interprétée à travers la structure K comme une affectation φ^K entre les évaluations et les sous-ensembles 2^W, $\varphi^K(\nu)$ désigne l'ensemble des états qui satisfont φ sous une évaluation ν. Cette affectation est définie inductivement par les règles suivantes :

- $true^K(\nu) = W$ et $false^K(\nu) = \emptyset$;
- Pour $p \in Ap \cup Nom$, $p^K(\nu) = L(p)$ et $(\neg p)^K(\nu) = W \backslash L(p)$;
- Pour $y \in Var$, $y^K(\nu) = \nu(y)$;
- $(\varphi_1 \wedge \varphi_2)^K(\nu) = (\varphi_1)^K(\nu) \cap (\varphi_2)^K(\nu)$ et $(\varphi_1 \vee \varphi_2)^K(\nu) = (\varphi_1)^K(\nu) \cup (\varphi_2)^K(\nu)$;
- $(\langle n, \alpha\rangle\varphi_1)^K(\nu) = \{w : |\{w' \in W | (w, w)' \in R(\alpha) \text{ et } w' \in \varphi^K(\nu)\}| \geq n + 1\}$;
- $([n, \alpha]\varphi_1)^K(\nu) = \{w : |\{w' \in W | (w, w)' \notin R(\alpha) \text{ et } w' \in \varphi^K(\nu)\}| \leq n\}$;
- $(\mu y.\varphi_1(y))^K(\nu) = \bigcap\{W' \subseteq W : \varphi([y \leftarrow W']) \subseteq W'\}$;
- $(\nu y.\varphi_1(y))^K(\nu) = \bigcup\{W' \subseteq W : W' \subseteq \varphi([y \leftarrow W'])\}$;

Soit $K = \langle W, R, L\rangle$ une structure de Kripke et une formule φ. Pour un état $w \in W$, φ est valide à w sur K (noté $K, w \models \varphi$), si $w \in \varphi^K$. K est considéré comme un modèle pour φ s'il existe $w \in W$ telle que $K, w \models \varphi$. Finalement, φ est dite satisfaisable si elle admet un modèle.

2.1.5.3 Décidabilité

Dans le tableau 2.1, nous donnons les principaux résultats établis pour la complexité des différentes variantes du μ-calcul enrichi.

	Programmes inverses	Modalités étendues	Symboles nominaux	complexité
fully enriched μ-calculus	X	X	X	indécidable
full graded μ-calculus	X	X		EXPTIME
full hybrid μ-calculus	X		X	EXPTIME
hybrid graded enriched μ-calculus		X	X	EXPTIME
graded enriched μ-calculus		X		EXPTIME

Tab. 2.1: Résultats établis pour la complexité des variantes du μ-calcul enrichi

2.1.6 Conditional XPath (CXPath)[13]

L'expressivité de XPath 1.0 est assez limitée, surtout en ce qui concerne l'utilisation des chemins itératifs et des chemins conditionnés. Maarten Marx a proposé Regular XPath et CXPath[13] comme deux variantes plus expressives que la définition XPath 1.0. Dans ce qui suit, nous nous intéressons plutôt à CXPath et à sa capacité d'exprimer des requêtes XPath avec itération sur les chemins conditionnés.

2.1.6.1 Grammaire de $\mathcal{X}_{Core}$ et CXPath

CXPath est une extension du fragment Core XPath selon l'expressivité de la logique du premier ordre. L'avantage majeur de CXPath par rapport à $\mathcal{X}_{Core}$ (syntaxe de base pour les langages XPath) c'est qu'il permet d'exprimer les requêtes avec condition sur les chemins, d'ou le nom Conditional.

locpath ::= axis‘ : :’ntst | axis ‘ : :’ntst ‘[’fexpr‘]’ | ‘/’locpath | locpath ‘/’locpath | locpath ‘|’locpath
fexpr ::= locpath | not fexpr | fexpr and fexpr | fexpr or fexpr
axis ::= self | primitive_axis | primitive_axis$^+$ | primitive_axis*

avec
- les axes primitifs de $\mathcal{X}_{Core}$ sont $\Downarrow$, $\Uparrow$, $\Rightarrow$ et $\Leftarrow$
- les axes primitifs de CXPath sont $\Downarrow$, $\Uparrow$, $\Rightarrow$, $\Leftarrow$, $\Downarrow_{\text{fexpr}}$, $\Uparrow_{\text{fexpr}}$, $\Rightarrow_{\text{fexpr}}$ et $\Leftarrow_{\text{fexpr}}$
- “locpath” est l'acronyme de location path, est le chemin de départ
- “ntst” désigne le nom d'un noeud et “*” correspond à tout les noeuds
- “fexpr” est l'acronyme de filter expression, qui désigne le qualificateur d'un chemin donné.

2.1.6.2 Regular XPath, Core XPath et CXPath

Pour mettre l'accent sur l'expressivité de CXPath par rapport aux autres variantes XPath, nous donnons ci-dessous la syntaxe du Regular XPath, Core XPath et CXPath dans une forme similaire à la logique PDL.

Regular XPath
step ::= child | parent | right | left
p_wff ::= step | p_wff / p_wff | p_wff $\cup$ p_wff | primitive_axis* | ?n_wff
n_wff ::= p_i | $\top$ | $\langle$p_wff$\rangle$ | $\neg$ n_wff | n_wff $\vee$ n_wff | n_wff $\wedge$ n_wff

Core XPath
step ::= child | parent | right | left
p_wff ::= step | step* | p_wff / p_wff | p_wff $\cup$ p_wff | ?n_wff
n_wff ::= p_i | $\top$ | $\langle$p_wff$\rangle$ | $\neg$ n_wff | n_wff $\vee$ n_wff | n_wff $\wedge$ n_wff

CXPath
step ::= child | parent | right | left
p_wff ::= step | (step/?n_wff)* | p_wff / p_wff | p_wff $\cup$ p_wff | ?n_wff
n_wff ::= p_i | $\top$ | $\langle$p_wff$\rangle$ | $\neg$ n_wff | n_wff $\vee$ n_wff | n_wff $\wedge$ n_wff

p_wff est l'abréviation de “path_wff”, où wff est l'acronyme de “well-formed formula”, tandis que “n_wff” désigne les formules propositionnelles valides sur les noeuds.

2.1.6.3 Sémantique de CXPath

La sémantique des expressions CXPath est donnée en considérant la modélisation des documents XML comme des arbres dont les noeuds sont étiquetés par les propositions atomiques d'un alphabet

donné. Nous considérons un type particulier d'arbres qui sont les arbres ordonnés d'arité finie. Ces arbres sont basés sur deux relations binaires : la relation `child` notée par $R_{\Downarrow}$, et la relation `immediate-right-sibling` notée par $R_{\Rightarrow}$. Ces deux relations et leurs inverses respectifs $R_{\Uparrow}$ et $R_{\Leftarrow}$ sont utilisées pour interpréter les axes de CXPath.
locpath indique une relation binaire sur un ensemble d'axes. La signification des qualificateurs est donnée par le prédicat $\varepsilon_{\mathfrak{M}}(n, \text{fexpr})$ qui reçoit une valeur booléenne, ainsi une expression de qualificateur fexpr désigne un ensemble de noeuds tel que $\varepsilon_{\mathfrak{M}}(n, \text{fexpr})$ est vrai.
Étant donné un arbre $\mathfrak{M}$ et une expression A, la sémantique de A dans $\mathfrak{M}$ est écrite $[\![A]\!]_{\mathfrak{M}}$. La définition de $[\![.]\!]_{\mathfrak{M}}$ est donnée par le tableau 2.2.

$[\![\mathcal{X} :: t]\!]_{\mathfrak{M}}$	=	$\{(n,n') \mid n[\![\mathcal{X}]\!]_{\mathfrak{M}} n'$ et $t(n')\}$
$[\![\mathcal{X} :: t[e]]\!]_{\mathfrak{M}}$	=	$\{(n,n') \mid n[\![\mathcal{X}]\!]_{\mathfrak{M}} n'$ et $t(n')$ et $\varepsilon_{\mathfrak{M}}(n',e)\}$
$[\![/\text{locpath}]\!]_{\mathfrak{M}}$	=	$\{(n,n') \mid (\text{root},n') \in [\![\text{locpath}]\!]_{\mathfrak{M}}\}$
$[\![\text{locpath/locpath}]\!]_{\mathfrak{M}}$	=	$[\![\text{locpath}]\!]_{\mathfrak{M}} \circ [\![\text{locpath}]\!]_{\mathfrak{M}}$
$[\![\text{locpath—locpath}]\!]_{\mathfrak{M}}$	=	$[\![\text{locpath}]\!]_{\mathfrak{M}} \cup [\![\text{locpath}]\!]_{\mathfrak{M}}$
$[\![\Downarrow]\!]_{\mathfrak{M}}$	:=	$R_{\Downarrow}$
$[\![\Rightarrow]\!]_{\mathfrak{M}}$	:=	$R_{\Rightarrow}$
$[\![\Uparrow]\!]_{\mathfrak{M}}$	:=	$R_{\Uparrow}$
$[\![\Leftarrow]\!]_{\mathfrak{M}}$	:=	$R_{\Leftarrow}$
$[\![\text{self}]\!]_{\mathfrak{M}}$	=	$\{(x,y) \mid x = y\}$
$[\![p_{\text{fexpr}}]\!]_{\mathfrak{M}}$	:=	$\{(x,y) \mid (x,y) \in [\![p]\!]_{\mathfrak{M}}$ et $\varepsilon_{\mathfrak{M}}(y,\text{fexpr}) = true\}$
$[\![p^+]\!]_{\mathfrak{M}}$	:=	$[\![p]\!]_{\mathfrak{M}} \cup [\![p]\!]_{\mathfrak{M}} \circ [\![p]\!]_{\mathfrak{M}} \cup [\![p]\!]_{\mathfrak{M}} \circ \ldots$
$[\![p^*]\!]_{\mathfrak{M}}$	:=	$[\![\text{self}]\!]_{\mathfrak{M}} \cup [\![p^+]\!]_{\mathfrak{M}}$
$\varepsilon_{\mathfrak{M}}(n,\text{locpath}) = true$	$\Longleftrightarrow$	$\exists n' : (n,n') \in [\![\text{locpath}]\!]_{\mathfrak{M}}$
$\varepsilon_{\mathfrak{M}}(n,\text{fexpr}_1 and \text{fexpr}_2) = true$	$\Longleftrightarrow$	$\varepsilon_{\mathfrak{M}}(n,\text{fexpr}_1) = true$ *and* $\varepsilon_{\mathfrak{M}}(n,\text{fexpr}_2) = true$
$\varepsilon_{\mathfrak{M}}(n,\text{fexpr}_1 or \text{fexpr}_2) = true$	$\Longleftrightarrow$	$\varepsilon_{\mathfrak{M}}(n,\text{fexpr}_1) = true$ *or* $\varepsilon_{\mathfrak{M}}(n,\text{fexpr}_2) = true$
$\varepsilon_{\mathfrak{M}}(n, not \text{fexpr} = true$	$\Longleftrightarrow$	$\varepsilon_{\mathfrak{M}}(n,\text{fexpr}) = false$

TAB. 2.2: Sémantique de CXPath

2.1.6.4 Expressivité de CXPath

L'expressivité de CXPath est similaire à l'expressivité de la logique du premier ordre. En fait, à chaque formule de la logique du premier ordre correspond une requête CXPath. Grâce à cette richesse expressive nous pouvons écrire des requêtes avec des chemins conditionnés. Un chemin conditionné est décrit par la phrase suivante : "on fait un pas selon un chemin bien défini si le test est vrai dans le noeud résultant".
Avec CXPath il est possible aussi de formuler des requêtes avec clôture réflexive transitive sur les chemins, ce qui peut résoudre un nombre important de problèmes. Pour illustrer cette richesse expressive, nous donnons ci-dessous deux exemples qui mettent l'accent sur des expressions ne pouvant pas être exprimées avec Core XPath.

Exemple 1
Considérons les requêtes avec l'expression "until-like" qui exprime la satisfaction permanente d'une proposition atomique à partir d'un noeud contexte jusqu'à un ensemble de descendants. La réponse à cette expression consiste en un ensemble de noeuds descendants du noeud contexte, et dont les ancêtres jusqu'au noeud contexte satisfont la proposition atomique considérée, le résultat de cette expression est un ensemble de noeuds n' qui répondent à la condition suivante : n' est un descendant de n, le libellé de n est A, et pour tout x, si x est un descendant de n, et n' est un descendant de x, alors le libellé de x est A.
Ce type d'expressions est décrit grâce à l'itération sur le chemin du noeud fils : $(\text{child} :: \text{A})^+$

Exemple 2
A partir d'un noeud contexte, nous cherchons les éléments d'un document XML dont les noeuds suivants ont la balise A. Ce problème est formulé selon la logique du premier ordre comme suit :
$\exists y(x \ll y \wedge \mathrm{A}(y) \wedge \neg\exists z(x \ll z \ll y))$
avec $\ll$ l'abréviation de :
`descendant or ancestor-or-self/following-sibling/descendant-or-self.`
Pour exprimer cette information dans XPath, il faut définir la relation "suivant dans l'ordre du document XML", pour cela nous utilisons les macros : `first, last et leaf` définies respectivement comme suit :
`self::*[not` $\Rightarrow$`::*], self::*[not` $\Leftarrow$`::*] et self::*[not` $\Downarrow$`::*]`
Nous utilisons aussi les axes inverses conditionnés tel que $_{\text{last}}\Uparrow^{+}$ qui peut être écrit sous la forme `self::*[last]/`$\Uparrow$`::*`, maintenant avec CXPath, nous pouvons écrire la relation "suivant dans l'ordre du document XML" comme suit :
$\Downarrow$`::*[first] | self::*[leaf]/` $\Rightarrow$`::*` | $_{\text{last}}\Uparrow^{+}$`::*/`$\Rightarrow$`::*.`
Grâce à cette expression, le reste du problème peut être résolu facilement.

2.1.6.5 Décidabilité

Les requêtes CXPath peuvent être évaluées en un temps polynomial complet en fonction de la taille de la requête et des données ; o (|D|.|Q|) avec |D| la taille des données et |Q| la taille de la requête. Core XPath admet la même complexité malgré qu'il est moins expressif que CXPath.

2.2 Logiques qui comptent

Le comptage est l'un des problèmes majeurs auquel le sujet est confronté. Dans ce qui suit, nous étudions les théories qui peuvent exprimer des contraintes numériques en essayant de trouver des variantes efficaces de ces formalismes avec lesquelles nous pouvons formuler les contraintes de comptage XPath.

2.2.1 Arithmétique de Presburger [14]

L'arithmétique de Presburger est la théorie de premier ordre des nombres normaux avec l'addition. L'arithmétique prouvée par Presburger est :

- ✓ Conforme : Tous les rapports prouvables dans l'arithmétique de Presburger sont vrais.
- ✓ Complète : Tous les rapports vrais dans l'arithmétique de Presburger sont prouvables par des axiomes.
- ✓ Décidable : Il existe un algorithme qui décide si n'importe quel rapport donné dans l'arithmétique de Presburger est vrai ou faux.

2.2.1.1 Syntaxe

Soit $\mathbb{Z}$ l'ensemble des entiers, et V un ensemble de variables entières. Dans l'arithmétique de Presburger, un terme est une combinaison finie de variables tandis qu'une inégalité est une comparaison $(=, <, >, \leq, \geq)$ entre termes. Une formule basique de Presburger est construite à partir d'une combinaison booléenne finie d'inégalités, alors qu'une formule de Presburger est une combinaison booléenne finie d'inégalités dans laquelle les variables peuvent être quantifiée. Ainsi, si x, y, z, u sont des variables, alors $x + 2y + 1$ est un terme, $x \leq y \wedge x + 2y + 1 > 0$ est une formule basique de Presburger, et $\forall z.x \leq y + z \wedge \exists u.x + 2y + u > y$ est une formule de Presburger.
L'arithmétique de Presburger admet les axiomes suivants :

$$\neg(0 = x + 1) \tag{2.1}$$

$$x + 1 = y + 1 \longrightarrow x = y \tag{2.2}$$

$$x + 0 = x \tag{2.3}$$

$$(x + y) + 1 = x + (y + 1) \tag{2.4}$$

$$(P(0) \wedge (P(x) \longrightarrow P(x + 1))) \longrightarrow P(x) \tag{2.5}$$

Avec $P(x)$ une formule de la logique du premier ordre incluant les constantes $(0, 1, +, =)$, et x une variable sans quantification.
Les formules de Presburger peuvent être écrites comme des prédicats binaires sous la forme suivante : $t \equiv c \ (mod\ n)$, avec n constante.

2.2.1.2 Décidabilité

Soit n la taille d'entrée d'une formule dans l'arithmétique de Presburger ; Fischer et Rabin (1974) ont montré que n'importe quel algorithme de décision pour l'arithmétique de Presburger a un temps d'exécution minimal de $2^{2^{cn}}$, pour une certaine constante c>0.
Par ailleurs, d'autres résultats montrent que le problème de décision pour l'arithmétique de Presburger nécessite un temps d'exécution exponentiel triple en fonction de la taille de la formule. Le fragment de l'arithmétique de Presburger restreint aux formules basiques est décidable en un temps exponentiel simple.

2.2.2 PDL (Propositional Dynamic Logic) [3]

Introduite par Fisher et Ladner, PDL est une logique conçue pour la simulation des programmes informatiques. Dans ce travail de recherche, nous nous intéressons uniquement à l'aptitude de cette logique à exprimer les contraintes de comptage. Nous donnons par la suite la définition relative à la version standard de PDL (regular PDL).

2.2.2.1 Syntaxe

Étant donné un ensemble fini de propositions atomiques $P = \{p, q, ...\}$ et un ensemble de programmes atomiques (instructions ou actions) $\pi = \{\rho, ...\}$. Les formules et les programmes de PDL sont construites à partir des règles suivantes :

$\varphi ::= p \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid [\pi]\varphi$
$\pi ::= \rho \mid \pi_1 \cup \pi_2 \mid \pi_1;\pi_2 \mid \pi^*$

La lecture intuitive des formules et des programmes PDL est comme suit :

- La relation ; est l'exécution séquentielle, $\pi_1;\pi_2$ est le programme formé de π_1 suivi de π_2.
- La relation $\cup$ est le choix entre deux programmes atomiques (exécuter l'un des deux).
- Le symbole $*$ indique l'itération, π^* est le programme qui répète π un nombre fini de fois (y compris 0).
- $[\pi]\varphi$ est une formule qui est vraie si φ est nécessairement vraie après toutes les exécutions de π.
- $\langle\pi\rangle\varphi$ est une formule qui est vraie si φ est vraie après au moins une exécution du programme π.

La logique PDL est axiomatisable, elle comprend les axiomes et les règles d'inférence données par la figure 2.6.

$\langle\pi\rangle\varphi \leftrightarrow \neg[\pi]\neg\varphi$
$\langle\pi_1;\pi_2\rangle\varphi \leftrightarrow \langle\pi_1\rangle\langle\pi_2\rangle\varphi$
$\langle\pi_1 \cup \pi_2\rangle\varphi \leftrightarrow \langle\pi_1\rangle\varphi \vee \langle\pi_2\rangle\varphi$
$\langle\pi^*\rangle\varphi \leftrightarrow (\varphi \vee \langle\pi\rangle\langle\pi^*\rangle\varphi)$
$[\pi](\varphi_1 \rightarrow \varphi_2) \rightarrow [\pi]\varphi_1 \rightarrow [\pi]\varphi_2$
$[\pi]\varphi_1 \wedge [\pi](\varphi_1 \rightarrow \varphi_2) \rightarrow [\pi]\varphi_2$
$[\pi_1;\pi_2]\varphi \leftrightarrow [\pi_1][\pi_2]\varphi$
$[\pi_1 \cup \pi_2]\varphi \leftrightarrow [\pi_1]\varphi \vee [\pi_2]\varphi$
$[\varphi_1?]\varphi_2 \leftrightarrow \varphi_1 \rightarrow \varphi_2$
$[\pi^*]\varphi \rightarrow [\pi^*][\pi^*]\varphi$
$[\pi^*]\varphi \rightarrow \varphi \wedge [\pi]\varphi$
$[\pi^*](\varphi \rightarrow [\pi]\varphi) \rightarrow (\varphi \rightarrow [\pi^*]\varphi)$

FIG. 2.6: Axiomes de la logique PDL

2.2.2.2 Modèle

Un modèle PDL est une structure $\mathcal{M} = \langle W, R, V\rangle$, W est un ensemble fini d'états, R est une fonction qui affecte à chaque programme π une relation binaire R_π sur W, et V est une fonction d'évaluation qui donne pour chaque proposition atomique p un sous-ensemble $V(p)$ de W.
Dans ce modèle, la relation de satisfaction $\mathcal{M}, w \models \varphi$ est lue comme "φ est vrai à l'état w pour le modèle $\mathcal{M}$", et elle est définie comme suit :

1. $\mathcal{M}, w \models p$ ssi $w \in V(p)$ pour toute proposition atomique p
2. $\mathcal{M}, w \models \neg\, p$ ssi $\mathcal{M}, w \not\models p$
3. $\mathcal{M}, w \models p \rightarrow q$ ssi $\mathcal{M}, w \not\models p$ ou $\mathcal{M}, w \models q$

4. $\mathcal{M}, w \models [\pi]p$ ssi $\forall w\prime \in W$, si $(w, w\prime) \in R_\pi$ alors $\mathcal{M}, w\prime \models p$

Les programmes PDL doivent satisfaire les contraintes suivantes :

- Si $R_\pi(w, w\prime)$ et $\mathcal{M}, w\prime \models \varphi$, alors $\mathcal{M}, w \models \langle\pi\rangle\varphi$.
- $R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}$
- $R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$ (composition)
- $R_{\pi^*} = (R_\pi)^*$ (clôture réflexive et transitive)

2.2.2.3 Décidabilité

La logique PDL est décidable en temps exponentiel (EXPTIME-complete).

2.2.2.4 PDL enrichi

Nous considérons une variante de PDL qui comprend, en plus des formules et des programmes PDL standards, les formules et les programmes suivants :

Formule de comptage

Pour exprimer les contraintes de comptage, les formules PDL sont enrichies par des sous-formules qui traduisent deux types de contraintes numériques : la comparaison (égalité, inégalité) et l'équivalence (modulo), ces sous-formules sont de la forme suivante :

- $\#^{<\rho>}\varphi \sim c$ (comptage)
- $\#^{<\rho>}\varphi \equiv_{(m)} c$ (modulo)

avec $\sim \in \{<, >, =, \geq, \leq\}$ et c une constante.

Programme inverse

Ce programme est employé pour formaliser le parcours des arbres XML et le comptage des noeuds dans des sens opposés. $\overline{\rho}$ dénote l'action inverse du programme atomique ρ, sa relation est définie comme suit : $R_{\overline{\rho}} =_{def} \{(w\prime, w) \in W \times W | R_\rho(w, w\prime)\}$.

Même en limitant ce constructeur à des programmes atomiques, il pose des problèmes à la fois de calcul et d'interprétation qui en limitent l'usage. Dans ce travail, nous utilisons seulement des expressions monotones ne contenant que des programmes atomiques de même sens.

Négation de programme

$\neg\pi$ dénote le programme dont la relation est le complémentaire de celle de π : $R_{\neg\pi} =_{def} \{(w\prime, w) \in W \times W - R_\pi\}$.

Ce constructeur est employé pour formaliser des questions de planification où l'on veut disposer d'une commande ``ne pas faire π''. L'opérateur de négation est limité à l'ensemble des programmes atomiques.

Intersection de programme

Ce programme est noté $\pi_1 \cap \pi_2$, avec la définition sémantique naturelle

$R_{\pi_1 \cap \pi_2} =_{def} R_{\pi_1} \cap R_{\pi_2}$. Autrement dit, deux états sont reliés par le programme intersection ssi ils sont reliés par chacun des programmes. Nous expliquons cette relation en disant qu'il s'agit d'une exécution parallèle, ou encore d'un choix (disjonction) de programme : il est possible de passer de tel à tel état par l'un ou l'autre des programmes.

Programme de test

La version standard de PDL n'a pas la possibilité d'écrire une conditionnelle. Pour cela, il faut introduire un type d'actions particulier : des actions qui ne changent rien au statut des états, mais qui produisent de l'information. Nous définissons pour chaque formule φ une action $\varphi?$ qui réussit si φ est vrai dans l'état où s'applique le test, et qui échoue sinon.

Nous définissons donc $R_{\varphi?} =_{def} \{(w,w)|\mathcal{M} \models_w \varphi\}$. Dans ce travail, nous considérons le test comme un constructeur qui s'applique seulement à des propositions atomiques.

Synthèse de la syntaxe PDL
$\varphi ::= p \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \langle\pi\rangle\varphi \mid [\pi]\varphi \mid \#^{<\rho>}\varphi \sim c \mid \#^{<\rho>}\varphi \equiv_{(m)} c$
$\pi ::= \rho \mid \overline{\rho} \mid \neg\rho \mid \pi_1 \cup \pi_2 \mid \pi_1 \cap \pi_2 \mid \pi_1;\pi_2 \mid \pi^* \mid \varphi?$ avec $\sim \in \{<, >, =, \geq, \leq\}$ et c une constante.

2.3 Synthèse des travaux liés

La synthèse des travaux menés autour de notre sujet peut être faite en subdivisant les logiques étudiées en deux catégories selon leur objectif et contexte d'usage : nous distinguons les logiques pour XML et les logiques de comptage.
Le tableau 2.3 donne les principaux résultats établis pour logiques XML selon deux critères, à savoir : l'expressivité et la complexité pour le problème de la satisfaisabilité. Ce Tableau montre deux résultats

Logique	Expressivité				complexité
	programmes inverses	modalités étendues	chemin conditionné	clôture réflexive et itération	
Modal Fixpoint Logic					EXPTIME complete
Sheaves Logic				X	EXPTIME complete
EXML					EXPTIME
μ-calcul	X	X			simple EXPTIME
fully enriched μ-calcul	X	X			undecidable
CXPath	X		X	X	PTIME complete

Tab. 2.3: Synthèse des logiques pour XML

importants : La puissance expressive de CXPath qui permet, outre la traduction des concepts XPath, d'enrichir ce langage en lui autorisant d'autres opérations sur les arbres XML. Le deuxième résultat important concerne la complexité du μ-calcul qui est très efficace sur le plan pratique pour le problème de décision du langage XPath.
En ce qui concerne les logiques de comptage, les études déjà établies montrent que la logique PDL est plus efficace que l'arithmétique de Presburger en terme d'expressivité et de complexité.

En résumé, l'approche que nous allons présenter dans le chapitre suivant sera le brassage des trois idées suivantes :

a. Considérer un fragment XPath de puissance expressive qui peut atteindre celle de Core XPath et CXPath à la fois.
b. Considérer une logique construite à partir du brassage de la logique PDL et du μ-calcul.
c. Assurer la décidabilité du formalisme résultant par une traduction en μ-calcul.

Chapitre 3

Logique pour XPath avec comptage

Nous présentons dans ce chapitre le fragment XPath à considérer pour les requêtes XPath avec comptage, ainsi que la logique proposée pour la décidabilité de ce fragment.
La variante XPath considérée est une synthèse des fragments Core Xpath et CXPath, enrichie par les contraintes de comptage.
Baptisée Dynamic μ-calculus, la logique proposée pour la décidabilité du fragment en question, est inspirée de la synthèse de la littérature des logiques pour XML et des logiques de comptage présentée dans le chapitre précédent.
Dynamic μ-calculus consiste à considérer le μ-calcul proposé pour XML comme logique de base, et à lui ajouter des "sucres syntaxiques commodes" à partir de la logique PDL. Cette adjonction apporte de nouvelles primitives de programmation qui sont essentiellement des primitives de navigation et de comptage.
Dynamic μ-calculus tire profit de la décidabilité du μ-calcul pour prouver la satisfaisabilité des requêtes XPath avec contraintes de comptage.
Notre approche permet d'assurer la continuité avec les travaux déjà établis par l'équipe WAM de l'INRIA, et d'exploiter les propriétés d'analyse statique et de décidabilité du μ-calcul considéré dans ces travaux.

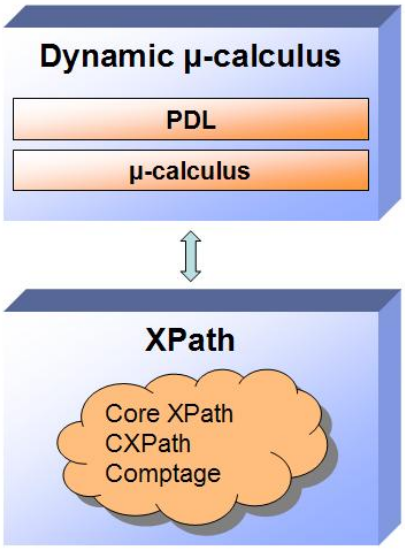


FIG. 3.1: Dynamic μ-calculus

3.1 Restriction de comptage

Dans ce travail, nous nous intéressons uniquement aux contraintes de comptage exprimées par la fonction count. Cette fonction peut intervenir à plusieurs niveaux dans les requêtes XPath, y compris les expressions de chemin et les qualificateurs. Notre objectif est de résoudre le problème de comptage pour les qualificateurs (qualifiers). La syntaxe XPath que nous considérons sera restreinte au comptage au niveau qualificateur : seuls les qualificateurs peuvent exprimer des contraintes numériques.

3.2 Automates d'arbre et arbres binaires

Pour modéliser les documents XML, nous avons opté pour une approche conventionnelle qui consiste à adopter les automates d'arbre[1]. Ce choix présente deux avantages : d'abord tirer profit des propriétés des automates qui permettent de définir les langages réguliers d'arbres avec une sémantique opérationnelle claire, et qui représentent un outil algorithmique efficace pour les procédures de décision logique, ainsi que pour la validation et la preuve des formules. La deuxième motivation pour ce choix consiste à avoir une vision arborescente des structures XML. En fait, les arbres reflètent la structure hiérarchique de XML et modélisent ses données sous-jacentes. Les automates choisis pour modéliser

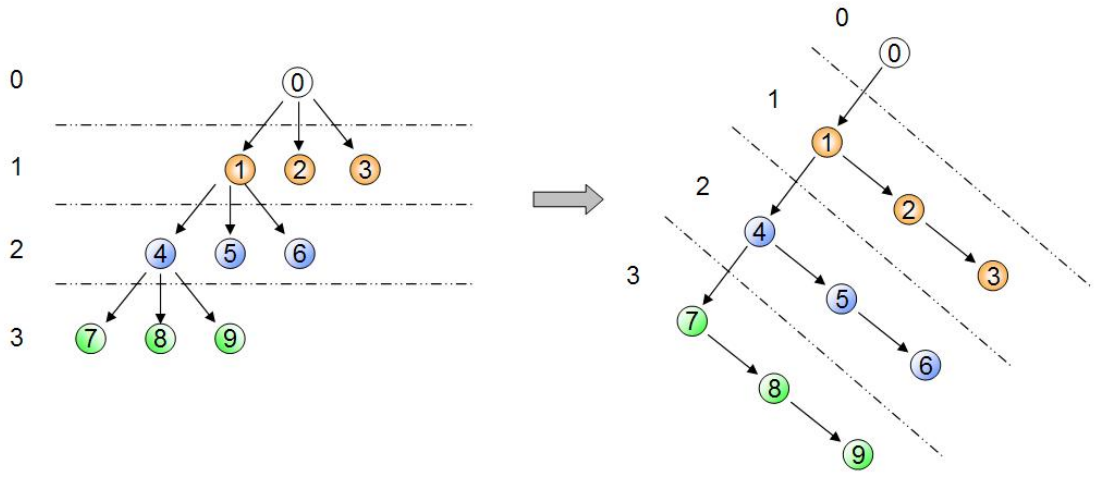


FIG. 3.2: Arbre d'arité non bornée et arbre binaire équivalent

les documents XML sont des arbres finis, étiquetés, ordonnés, de profondeur et d'arité non bornée. La propriété de la non bornitude peut engendrer des problèmes pour la procédure de décision de la logique considérée. Pour remédier à cela nous utilisons un encodage bijectif des arbres d'arité non bornée en arbres binaires. La figure 3.2 illustre cette conversion qui permet de considérer les arbres binaires sans perte de généralités.

3.3 Syntaxe abstraite de XPath

Nous avons centralisé nos efforts sur un fragment XPath aussi large et aussi efficace que possible, couvrant les caractéristiques principales de la norme XPath 2.0. Le fragment considéré de XPath est ,en fait, la composition du fragment Core XPath, avec les principaux opérateurs de comptage, et les expressions des chemins de CXPath. La syntaxe des expressions XPath considérées est donnée par la figure 3.3.
La syntaxe abstraite de XPath donnée ci-dessus comprend trois types de comptage : à savoir, la comparaison selon les opérateurs $\{!=, =, <, >, \geq, \leq\}$, le comptage modulo (*mod*) et la division (*div*).
Ce fragment inclut également les expressions des chemins conditionnés, les chemins itératifs et la clôture

$\mathcal{L}_{\text{XPath}} \ni e$	::=		expression XPath
		$/p$	chemin absolu
	|	p	chemin relatif
	|	$e_1 \mid e_2$	union
	|	$e_1 \cap e_2$	intersection
Path p	::=		chemin
		p^*	clôture réflexive transitive
	|	p^+	itération
	|	$p_1 \,/\, p_2$	chemin composé
	|	$p \,/\, q?$	chemin conditionné
	|	$p\,[q]$	chemin avec qualificateur
	|	$a :: \sigma$	étape avec test de noeuds
	|	$a :: *$	étape
Qualif q	::=		qualificateur
		$count(e) \sim c$	comptage, $\sim \in \{!=, =, <, >, \geq, \leq\}$ et c constante
	|	$count(e)\ mod\ c$	comptage modulo
	|	$count(e)\ div\ c$	division
	|	$q_1\ and\ q_2$	conjonction
	|	$q_1\ or\ q_2$	disjonction
	|	$\neg\, q$	négation
	|	p	chemin
Axis a	::=		axe de navigation
		child | self | parent	
	|	descendant | descendant-or-self	
	|	ancestor | ancestor-or-self	
	|	following-sibling | preceding-sibling	
	|	following | preceding	

FIG. 3.3: Syntaxe des expressions XPath

réflexive transitive sur les chemins, ce qui augmente considérablement la puissance expressive du fragment XPath considéré.

3.4 Dynamic μ-calculus

Bâti selon une approche en couches, Dynamic μ-calculus comprend deux logiques : le μ-calcul comme langage de base $\mathcal{L}_\mu$ et la logique PDL constituant les formules de haut niveau.
Déterminer la satisfaisabilité d'une requête XPath en Dynamic μ-calculus est un processus englobant trois phases : d'abord, la traduction de la requête en question en logique PDL (puisque l'expressivité de cette logique est assez riche pour supporter l'ensemble des requêtes du fragment XPath considéré), ensuite la traduction de la formule PDL résultante en μ-calcul, et enfin l'application de l'algorithme de décision à la formule μ-calcul obtenue pour déterminer la satisfaisabilité de la requête considérée.
Dans ce qui suit, nous donnons la syntaxe du Dynamic μ-calculus.

3.4.1 Langage de base

Nous introduisons la logique de base du Dynamic μ-calculus, qui représente le langage cible de la traduction des concepts XPath. C'est un sous-ensemble du μ-calcul propositionnel avec programme inverse.
Dans cette logique, $a \in \{1, 2, \overline{1}, \overline{2}\}$ indique les programmes de navigation et σ représente une proposition atomique qui correspond à l'étiquette d'un noeud de la structure XML.
La formule de la figure 3.4 inclut les prédicats vrai ($\top$) et faux ($\bot$), les propositions atomiques, la disjonction et la conjonction des formules, la modalité existentielle, le plus petit et le plus grand point fixe.

$\mathcal{L}_\mu \ni \varphi, \psi$	::=				formule
		$\top$	|	$\bot$	vrai
	|	σ	|	$\neg\sigma$	proposition atomique (négation)
	|	Ⓢ	|	$\neg$Ⓢ	proposition de départ (négation)
	|	X			variable
	|	$\varphi \vee \psi$			disjonction
	|	$\varphi \wedge \psi$			conjonction
	|	$\langle a \rangle \varphi$	|	$\neg\langle a \rangle \varphi$	existentiel (négation)
	|	$\mu \overline{X_i.\varphi_i}$			plus petit point fixe
	|	$\nu \overline{X_i.\varphi_i}$			plus grand point fixe

FIG. 3.4: Langage de base

3.4.2 Formules de haut niveau

Les formules de haut niveau se déclinent en deux catégories, les formules propositionnelles (relatives aux propositions sur les noeuds XML) et les constructeurs de programmes [5](relatifs aux chemins de navigation dans la structure XML).
Les constructeurs de programmes exploitent la richesse expressive de PDL et de CXPath pour définir des programmes avancés qui permettent une meilleure navigation dans la structure XML. Parmi ces programmes nous citons les programmes atomiques inverses autorisant le parcours des arbres XML dans des sens opposés, le test d'existence qui permet d'exprimer des conditionnelles, le chemin conditionné, le chemin itératif et la clôture réflexive transitive sur les chemins.
Les formules propositionnelles incluent les propositions atomiques, la négation et la conjonction des formules, la modalité existentielle, mais aussi les opérateurs de comptage et l'opérateur modulo qui sont les modalités essentielles pour satisfaire les contraintes de comptage. La syntaxe des formules de haut niveau est donnée par la figure 3.5.

π	::=		programme de chemin
		ρ	programme atomique, avec $\rho \in \{\epsilon, 1, 2\}$
	|	$\overline{\rho}$	programme atomique inverse
	|	$\rho/\varphi?$	programme conditionnel
	|	$\neg\pi$	négation de programme
	|	$\pi;\pi$	composition
	|	$\pi \cup \pi$	union
	|	$\pi \cap \pi$	intersection
	|	π^+	itération
	|	π^*	clôture réflexive transitive
	|	$\varphi?$	test

φ	::=		formule
		p	proposition atomique
	|	$\varphi \wedge \varphi$	conjonction
	|	$\neg\varphi$	négation
	|	$\langle\pi\rangle\varphi$	existentiel
	|	$\#^{<\pi>}\varphi \sim c$	comparaison de comptage
	|	$\#^{<\pi>}\varphi \equiv_{(m)} c$	égalité modulo

FIG. 3.5: Formules de haut niveau

3.5 Deux solutions de comptage

Une fois la logique Dynamic μ-calculus définie, l'interprétation des concepts XPath dans cette logique s'impose. Les études faites à cet égard ont abouti à définir deux approches qui dépendent essentiellement de la solution proposée pour la translation des contraintes de comptage. Dans ce qui suit, nous introduisons les exigences que doivent satisfaire ces approches afin de conserver la sémantique de XPath et assurer la décidabilité de la logique présentée. Nous donnons par la suite un aperçu des deux solutions proposées.

3.5.1 Conditions de comptage

L'interprétation des formules de comptage doit satisfaire deux contraintes principales, à savoir : la monotonie du chemin et la définition d'un ordre de comptage. Ces exigences ont pour objectif de garantir la sûreté et l'efficacité lors du dénombrement des noeuds concernés par une requête XPath avec comptage.

3.5.1.1 La monotonie

La monotonie des programmes est l'une des conditions nécessaires que doivent assurer les formules de comptage afin d'éviter les cycles et les boucles infinies pour une récursion de comptage finie. Cette condition indique que l'occurrence des programmes sous un chemin de comptage ne doit pas inclure une composition de programmes inverses. Soit φ la formule à compter, la contrainte de monotonie exige que φ ne contienne pas de programmes de la forme $\langle \pi; \overline{\pi} \rangle$ avec $\pi \in \{\rho, \rho^+, \rho^*\}$ et $\overline{\pi} \in \{\overline{\rho}, \overline{\rho}^+, \overline{\rho}^*\}$.

3.5.1.2 L'ordre de comptage

Un comptage ordonné a pour objectif de référencer les noeuds déjà comptés et éviter le dénombrement multiple des noeuds satisfaisant une formule de comptage donnée. Pour définir un ordre de comptage, nous avons proposé un ordre qui se limite aux noeuds visés par la requête et qui se base sur la monotonie du comptage. En fait, si le programme de comptage est monotone, le problème du comptage multiple d'un noeud ne se pose pas puisqu'il nécessite un retour en arrière par le chemin de comptage, ce qui est contradictoire avec la condition de monotonie.

3.5.2 Aperçu des approches proposées

La première approche consiste à décomposer la formule de comptage en formules simples et élémentaires comprenant des chemins de comptage primitifs et des propositions atomiques. La monotonie du comptage selon cette approche est évidente puisque le chemin est primitif, tandis que le comptage ordonné est assuré par la condition de monotonie, l'ordre de comptage considéré dans ce cas est dit relatif (puisqu'il dépend du chemin du comptage).
La deuxième solution proposée effectue un comptage global selon l'intégralité de la formule de comptage qui contient des chemins et des formules propositionnelles composites. Pour assurer la monotonie des programmes de comptage selon cette approche, nous avons défini un environnement de détection des cycles qui vérifie la présence des cycles au sein de chaque programme figurant dans la formule considérée. Tandis que l'ordre de comptage est assuré par la définition d'un ordre total intrinsèque à la structure XML qui est l'ordre du document (c'est-à-dire l'ordre d'apparition des noeuds dans le document XML). Dans le quatrième et le cinquième chapitre, nous reviendrons sur les détails de ces deux approches.

3.6 Deux sémantiques de traduction

Dans chaque approche, la translation des concepts XPath se fait selon deux types d'interprétation sémantique[21], à savoir la sémantique de navigation et la sémantique de sélection. La première interprétation a pour principe de référencer le noeud contexte, ensuite de suivre le chemin de navigation jusqu'à trouver le noeud satisfaisant la formule considérée en substituant à chaque fois le contexte par le noeud courant. Tandis que la sémantique de sélection consiste à tester la satisfaisabilité d'une condition à partir d'un noeud contexte sans parcourir le chemin de navigation relatif à la condition et sans substituer le contexte. Ces sémantiques sont utilisées dans les deux approches pour la translation des différents types de formules relatives aux concepts XPath.

Chapitre 4

Décomposition de comptage et ordre relatif

Dans ce chapitre nous présentons la première approche proposée pour résoudre le problème de décision des requêtes XPath avec contraintes de comptage. La particularité de cette solution réside dans le fait qu'elle décompose le problème de comptage en plusieurs problèmes élémentaires, elle se résume en deux actions, d'abord subdiviser la formule de comptage en formules propositionnelles simples lors de la translation des concepts XPath en PDL, ensuite subdiviser le chemin de comptage en programmes atomiques lors de la traduction des formules $\mathcal{L}_{\text{PDL}}$ en μ-calcul. Dans ce qui suit nous donnons les détails de cette décomposition.

4.1 Exigences de comptage

L'approche par décomposition de comptage répond à la fois aux exigences de monotonie et du comptage ordonné, la monotonie est évidente puisque le chemin est atomique donc il est forcément monotone, tandis que l'ordre de comptage est assuré par la condition de monotonie. En fait, si le chemin est monotone, le dénombrement multiple d'un noeud devient absurde car cela nécessite un retour par le même chemin ce qui est contradictoire avec la condition de monotonie. Ces résultats font que l'ordre est relatif au chemin de comptage d'où le nom ordre relatif.

4.2 Sémantique de navigation ou sémantique de sélection ?

Les sémantiques de navigation et de sélection interviennent d'une manière directe dans l'interprétation des concepts XPath en Dynamic μ-calculus. La question qui se pose à cet égard c'est quel est la sémantique à adopter pour l'interprétation des différents types de formules ?
Notre réponse à cette question consiste à considérer la sémantique de navigation pour les expressions des chemins et des axes puisque ce sont des primitives de parcours de la structure XML. Pour les expressions des qualificateurs nous avons adopté la sémantique de sélection puisque ces derniers sont des expressions de sélection.

4.3 Translation des concepts XPath en PDL

La procédure de décision du fragment XPath considéré comprend plusieurs étapes, une des étapes à franchir en premier lieu c'est de traduire des concepts XPath en PDL. Dans cette section nous expliquons comment transformer les requêtes XPath en formules $\mathcal{L}_{\text{PDL}}$. Cette traduction adhère à la sémantique XPath dans le sens où la formule PDL obtenue par la traduction d'une requête, est valide par rapport à l'ensemble des noeuds sélectionnés par la requête en question. Dans le même contexte, la navigation sur un arbre XML non borné est assimilée à une navigation dans l'arbre binaire équivalent (obtenu par le mécanisme de conversion introduit dans le chapitre précédent).

4.3.1 Interprétation logique des axes

Nous donnons la traduction des primitives de navigation (les axes XPath). La fonction de traduction $A^{\leftarrow}[\![a]\!]_{\chi}$ prend un axe a comme paramètre d'entrée et retourne la traduction équivalente en terme de formules $\mathcal{L}_{\text{PDL}}$. Le paramètre χ indique le contexte de la requête et permet la décomposition de la formule. $A^{\leftarrow}[\![a]\!]_{\chi}$ est valide pour tous les noeuds accessibles suivant l'axe a à partir du contexte χ. La figure 4.1 détaille la traduction les primitives de navigation XPath.

4.3.2 Interprétation logique des expressions

La figure 4.2 présente la traduction des expressions XPath en $\mathcal{L}_{\text{PDL}}$. La fonction $E^{\leftarrow}[\![e]\!]_{\chi}$ ayant comme paramètres d'entrée une expression XPath e et un noeud de contexte χ, retourne la traduction de l'expression en question.
La traduction des expressions relatives utilise le noeud de contexte courant comme point de référence, tandis que la traduction des expressions absolues considère la racine de la structure XML comme noeud contexte.
Il est important de souligner le rôle primordial que jouent les programmes inverses dans la composition de la traduction des concepts XPath. Le constructeur le plus important des chemins XPath p_1/p_2 est traduit en une composition de formules $\mathcal{L}_{\text{PDL}}$, la formule résultante est valide pour les noeuds accessibles par p_2 à partir de l'ensemble des noeuds accessibles par p_1 qui sont à leur tour accessibles à partir du noeud contexte.
La traduction des expressions de type $p[q]$ est différente de l'interprétation précédente dans le sens où la

$A^{\leftarrow}[\![\cdot]\!] : Axis \to \mathcal{L}_{\text{PDL}} \to \mathcal{L}_{\text{PDL}}$
$A^{\leftarrow}[\![\text{self}]\!]_\chi =_{def} \chi$
$A^{\leftarrow}[\![\text{child}]\!]_\chi =_{def} \langle \overline{2}^* ; \overline{1} \rangle \chi$
$A^{\leftarrow}[\![\text{following-sibling}]\!]_\chi =_{def} \langle \overline{2}^* ; \overline{2} \rangle \chi$
$A^{\leftarrow}[\![\text{preceding-sibling}]\!]_\chi =_{def} \langle 2^* ; 2 \rangle \chi$
$A^{\leftarrow}[\![\text{parent}]\!]_\chi =_{def} \langle 1 ; 2^* \rangle \chi$
$A^{\leftarrow}[\![\text{descendant}]\!]_\chi =_{def} \langle (\overline{1} \mid \overline{2})^* ; \overline{1} \rangle \chi$
$A^{\leftarrow}[\![\text{descendant-or-self}]\!]_\chi =_{def} \langle \epsilon \mid (\overline{1} \mid \overline{2})^* ; \overline{1} \rangle \chi$
$A^{\leftarrow}[\![\text{ancestor}]\!]_\chi =_{def} \langle 1 ; (1 \mid 2)^* \rangle \chi$
$A^{\leftarrow}[\![\text{ancestor-or-self}]\!]_\chi =_{def} \langle \epsilon \mid 1 ; (1 \mid 2)^* \rangle \chi$
$A^{\leftarrow}[\![\text{following}]\!]_\chi =_{def} A^{\leftarrow}[\![\text{descendant-or-self}]\!]_{\eta_1}$
$A^{\leftarrow}[\![\text{preceding}]\!]_\chi =_{def} A^{\leftarrow}[\![\text{descendant-or-self}]\!]_{\eta_2}$
$\eta_1 =_{def} A^{\leftarrow}[\![\text{following-sibling}]\!]_{A^{\leftarrow}[\![\text{ancestor-or-self}]\!]_\chi}$
$\eta_1 =_{def} A^{\leftarrow}[\![\text{preceding-sibling}]\!]_{A^{\leftarrow}[\![\text{ancestor-or-self}]\!]_\chi}$

FIG. 4.1: Translation des axes

$E^{\leftarrow}[\![\cdot]\!] : \mathcal{L}_{\text{XPath}} \to \mathcal{L}_{\text{PDL}} \to \mathcal{L}_{\text{PDL}}$
$E^{\leftarrow}[\![/p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\neg\langle \overline{2}^* ; \overline{1} \rangle \top)}$
$E^{\leftarrow}[\![p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_\chi$
$E^{\leftarrow}[\![e_1 \mid e_2]\!]_\chi =_{def} E^{\leftarrow}[\![e_1]\!]_\chi \vee E^{\leftarrow}[\![e_2]\!]_\chi$
$E^{\leftarrow}[\![e_1 \cap e_2]\!]_\chi =_{def} E^{\leftarrow}[\![e_1]\!]_\chi \wedge E^{\leftarrow}[\![e_2]\!]_\chi$

$P^{\leftarrow}[\![\cdot]\!] : Path \to \mathcal{L}_{\text{PDL}} \to \mathcal{L}_{\text{PDL}}$
$P^{\leftarrow}[\![p^+]\!]_\chi =_{def} (P^{\leftarrow}[\![p]\!]_\top)^+ \chi$
$P^{\leftarrow}[\![p^*]\!]_\chi =_{def} (P^{\leftarrow}[\![p]\!]_\top)^* \chi$
$P^{\leftarrow}[\![p_1 / p_2]\!]_\chi =_{def} P^{\leftarrow}[\![p_2]\!]_{P^{\leftarrow}[\![p_1]\!]_\chi}$
$P^{\leftarrow}[\![p/q?]\!]_\chi =_{def} P^{\leftarrow}[\![p/ \texttt{self} :: *[\texttt{q}]]\!]_\chi$
$P^{\leftarrow}[\![p[q]]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\chi)} \wedge Q^{\leftarrow}[\![q]\!]_\top$
$P^{\leftarrow}[\![a :: \sigma]\!]_\chi =_{def} \sigma \wedge A^{\leftarrow}[\![a]\!]_{(\chi)}$
$P^{\leftarrow}[\![a :: *]\!]_\chi =_{def} A^{\leftarrow}[\![a]\!]_\chi$

FIG. 4.2: Translation des expressions et des chemins

formule obtenue par cette traduction est valide par rapport aux noeuds accessibles par p à partir du noeud contexte, et pour lesquels q est valide. La translation de $p[q]$ limite la navigation aux noeuds accessibles par le chemin p, par la suite, ces noeuds sont filtrés selon la validité de q par rapport à eux : c'est la sémantique des qualificateurs XPath. La traduction de cette sémantique est assurée par une fonction de filtrage $Q^{\leftarrow}[\![q]\!]_\chi$ dont les détails sont développés dans le paragraphe suivant.

4.3.3 Interprétation logique des qualificateurs

$Q^{\leftarrow}[\![\cdot]\!]_\chi$ est une fonction de filtrage et de qualification, elle teste la validité d'une expression par rapport à un noeud en gardant la position courante et sans se déplacer selon les chemins de l'expression en question. La sémantique XPath implique que la translation de $Q^{\leftarrow}[\![q]\!]_\chi$ doit être valide par rapport aux noeuds pour lesquels le qualificateur q est valide.
Nous développons dans ce qui suit la traduction du comptage qui est l'une des fonctions de filtrage les plus importantes et dont la traduction est la plus ambiguë. La traduction d'une contrainte de comptage se décline en quatre phases itératives et non-indépendantes : d'abord, il faut traduire l'expression à compter ,paramètre de la fonction count, en une formule PDL équivalente suivant la traduction des expressions et des chemins à l'intérieur des qualificateurs qui est donnée par la figure 4.3. Cette traduction redéfinit l'interprétation des expressions et des chemins, et utilise une traduction spécifique pour les axes basée sur la symétrie XPath : *symmetric(a)* indique l'axe symétrique à a, par exemple, *symmetric(child)=parent*.
La deuxième phase consiste à éliminer l'ambiguïté liée à l'opérateur de négation $\neg$. Si la formule φ obtenue par la translation précédente contient des négations appliquées à ses sous-formules, il faut réécrire φ sous la forme négationelle normale en ramenant la négation sur les propositions atomiques. Considérons

$Q^{\leftarrow}[\![\cdot]\!] : Qualif \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$Q^{\leftarrow}[\![q_1 \text{ and } q_2]\!]_\chi =_{def} Q^{\leftarrow}[\![q_1]\!] \wedge Q^{\leftarrow}[\![q_2]\!]_\chi$
$Q^{\leftarrow}[\![q_1 \text{ or } q_2]\!]_\chi =_{def} Q^{\leftarrow}[\![q_1]\!] \vee Q^{\leftarrow}[\![q_2]\!]_\chi$
$Q^{\leftarrow}[\![not\ q]\!]_\chi =_{def} \neg Q^{\leftarrow}[\![q]\!]_\chi$
$Q^{\leftarrow}[\![p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_\chi$

$Q^{\leftarrow}[\![count(e) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![count\ (E^{\leftarrow}[\![e]\!]_\chi) \sim c]\!]_\chi$ — $\sim \in \{=, <, >, \geq, \leq\}$ et c une constante
$Q^{\leftarrow}[\![count\ (E^{\leftarrow}[\![e]\!]_\chi) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![count(\varphi) \sim c]\!]_\chi$ — avec φ translation de $E^{\leftarrow}[\![e]\!]_\chi$ en PDL

$Q^{\leftarrow}[\![count\ (\neg(\varphi_1 \wedge \varphi_2)) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![count\ (\neg\varphi_1 \vee \neg\varphi_2) \sim c]\!]_\chi$ — négation sous la forme normale
$Q^{\leftarrow}[\![count\ (\neg(\langle\pi\rangle\varphi)) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![count\ (\langle\pi\rangle\neg\varphi) \sim c]\!]_\chi$
$Q^{\leftarrow}[\![count\ (\neg\rho) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![\#^{<\epsilon>}\neg\rho \sim c]\!]_\chi$

$Q^{\leftarrow}[\![count\ (\varphi_1 \wedge \varphi_2) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![count\ (\varphi_1) \sim c]\!]_\chi \wedge Q^{\leftarrow}[\![count\ (\varphi_2) \sim c]\!]_\chi$ — conjonction du comptage
$Q^{\leftarrow}[\![count\ (\varphi_1 \vee \varphi_2) \sim c]\!]_\chi =_{def} [\![\#^{<\pi_1 \cup \pi_2>}(\varphi'_1 \vee \varphi'_2) \sim c]\!]_\chi$ — disjonction du comptage, avec $\varphi_1 = <\pi_1>\varphi'_1$ et $\varphi_2 = <\pi_2>\varphi'_2$

$Q^{\leftarrow}[\![count\ (\langle\pi\rangle\varphi) \sim c]\!]_\chi =_{def} [\![\#^{<\pi>}\varphi \sim c]\!]_\chi$
$Q^{\leftarrow}[\![count\ (\rho) \sim c]\!]_\chi =_{def} [\![\#^{<\epsilon>}\rho \sim c]\!]_\chi$

$Q^{\leftarrow}[\![count(e)\ mod\ c]\!]_\chi \quad =_{def} \quad Q^{\leftarrow}[\![count\ (E^{\leftarrow}[\![e]\!]_\chi)\ mod\ c]\!]_\chi$
$Q^{\leftarrow}[\![count\ (E^{\leftarrow}[\![e]\!]_\chi)\ mod\ c]\!]_\chi \quad =_{def} \quad Q^{\leftarrow}[\![count(\varphi) \equiv_{(m)} c]\!]_\chi$ — avec φ translation de $E^{\leftarrow}[\![e]\!]_\chi$ en PDL

$Q^{\leftarrow}[\![count\ (\neg(\varphi_1 \wedge \varphi_2)) \equiv_{(m)} c]\!]_\chi =_{def} Q^{\leftarrow}[\![count\ (\neg\varphi_1 \vee \neg\varphi_2) \equiv_{(m)} c]\!]_\chi$ — négation sous la forme normale
$Q^{\leftarrow}[\![count\ (\neg(\langle\pi\rangle\varphi)) \equiv_{(m)} c]\!]_\chi =_{def} Q^{\leftarrow}[\![count\ (\langle\pi\rangle\neg\varphi) \equiv_{(m)} c]\!]_\chi$
$Q^{\leftarrow}[\![count\ (\neg\rho) \equiv_{(m)} c]\!]_\chi =_{def} [\![\#^{<\epsilon>}\neg\rho \equiv_{(m)} c]\!]_\chi$

$Q^{\leftarrow}[\![count\ (\varphi_1 \wedge \varphi_2) \equiv_{(m)} c]\!]_\chi =_{def} Q^{\leftarrow}[\![count\ (\varphi_1) \equiv_{(m)} c]\!]_\chi \wedge Q^{\leftarrow}[\![count\ (\varphi_2) \equiv_{(m)} c]\!]_\chi$ — conjonction modulo
$Q^{\leftarrow}[\![count\ (\varphi_1 \vee \varphi_2) \equiv_{(m)} c]\!]_\chi =_{def} [\![\#^{<\pi_1 \cup \pi_2>}(\varphi'_1 \vee \varphi'_2) \equiv_{(m)} c]\!]_\chi$ — disjonction modulo, avec $\varphi_1 = <\pi_1>\varphi'_1$ et $\varphi_2 = <\pi_2>\varphi'_2$

$Q^{\leftarrow}[\![count\ (\langle\pi\rangle\varphi) \equiv_{(m)} c]\!]_\chi =_{def} [\![\#^{<\pi>}\varphi \equiv_{(m)} c]\!]_\chi$
$Q^{\leftarrow}[\![count\ (\rho) \equiv_{(m)} c]\!]_\chi =_{def} [\![\#^{<\epsilon>}\rho \equiv_{(m)} c]\!]_\chi$

$E^{\leftarrow}[\![\cdot]\!] : \mathcal{L}_{\text{XPath}} \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$E^{\leftarrow}[\![/p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\chi \wedge \langle \bar{2}^* ; \neg\bar{1} \rangle \top)}$
$E^{\leftarrow}[\![p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_\chi$

$P^{\leftarrow}[\![\cdot]\!] : Path \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$P^{\leftarrow}[\![p^+]\!]_\chi =_{def} (P^{\leftarrow}[\![p]\!]_\top)^+\chi$
$P^{\leftarrow}[\![p^*]\!]_\chi =_{def} (P^{\leftarrow}[\![p]\!]_\top)^*\chi$
$P^{\leftarrow}[\![p_1\ /\ p_2]\!]_\chi =_{def} P^{\leftarrow}[\![p_1]\!]_{(P^{\leftarrow}[\![p_2]\!]_\chi)}$
$P^{\leftarrow}[\![p/q?]\!]_\chi =_{def} P^{\leftarrow}[\![p/\texttt{ self : :*[q]}]\!]_\chi$
$P^{\leftarrow}[\![p[q]]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\chi \wedge Q^{\leftarrow}[\![q]\!]_\top)}$
$P^{\leftarrow}[\![a :: \sigma]\!]_\chi =_{def} A^{\leftarrow}[\![a]\!]_{(\chi \wedge \sigma)}$
$P^{\leftarrow}[\![a :: *]\!]_\chi =_{def} A^{\leftarrow}[\![a]\!]_\chi$

$A^{\leftarrow}[\![\cdot]\!] : Axis \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$A^{\leftarrow}[\![a]\!]_\chi =_{def} A^{\leftarrow}[\![symmetric(a)]\!]_\chi$

FIG. 4.3: Translation des qualificateurs

le cas le plus important qui est relatif au comptage de l'expression $\neg(\langle\pi\rangle\varphi)$. Afin de pouvoir compter les noeuds satisfaisant cette formule, il faut résoudre d'abord le problème de la négation, ce qui se fait de deux manières : soit en ramenant la négation sur le programme de navigation et écrire $\langle\neg\pi\rangle\varphi$, soit en niant la formule elle même $\langle\pi\rangle\neg\varphi$. Puisque le comptage selon un chemin nié est impossible alors le comptage de l'expression $\neg(\langle\pi\rangle\varphi)$ se ramène inévitablement au comptage de l'expression $\langle\pi\rangle\neg\varphi$.
Dans une troisième étape, la complexité du comptage d'une formule composée constitue un problème qui doit être résolu. Pour cela, il faut procéder à la décomposition du comptage en le factorisant sur des sous-formules simples et élémentaires constituant φ. Le comptage d'une conjonction de formules est relativement simple ; il consiste à la conjonction du comptage des formules considérées. Par contre, la factorisation de la disjonction de formules représente le point le plus délicat de la décomposition de comptage, car il faut franchir la dernière étape que nous détaillerons par la suite, et qui consiste à écrire les formules disjointes φ_1 et φ_2 sous les formes $< \pi_1 > \varphi_1\prime$ et $< \pi_2 > \varphi_2\prime$, ensuite procéder au comptage

en effectuant l'union des programmes et en gardant l'opérateur de disjonction pour les formules propositionnelles obtenues. Au cas où les formules φ_1 et φ_2 ne possèdent pas de programmes de chemin, $< \pi_1 >$ et $< \pi_2 >$ seront remplacés par le chemin vide ϵ.
Un fois la décomposition du comptage faite, il reste à écrire la formule φ à compter sous la forme $< \pi > \varphi'$ pour pouvoir la traduire en μ-calcul ; cette étape sera détaillée dans la section 4.4 de ce chapitre.
Nous abordons maintenant les propriétés intrinsèques du comptage, il est important de mettre l'accent sur une propriété fondamentale des contraintes de comptage qui est la dualité : l'ensemble des opérateurs de comptage sont opposés deux à deux. Cette opposition se manifeste via l'opérateur de négation, en fait, il suffit d'appliquer la négation au comptage d'une formule effectué selon un opérateur (par exemple $\leq$) pour trouver le résultat du comptage de la même formule effectué selon l'opérateur opposé ($>$). La figure 4.4 donne les détails de la dualité du comptage.

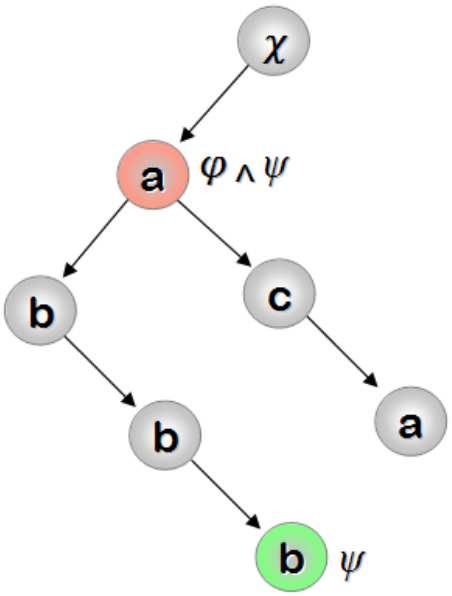
$$
\begin{aligned}
&Q^{\leftarrow}[\![count(e)\ !=\ c]\!]_{\chi} =_{def} Q^{\leftarrow}[\![not\ (count(e) = c)]\!]_{\chi}\\
&Q^{\leftarrow}[\![count(e)\ div\ c]\!]_{\chi} =_{def} Q^{\leftarrow}[\![not\ (count(e)\ mod\ c)]\!]_{\chi}\\
&Q^{\leftarrow}[\![count(e)\ >\ c]\!]_{\chi} =_{def} Q^{\leftarrow}[\![not\ (count(e) \leq c)]\!]_{\chi}\\
&Q^{\leftarrow}[\![count(e)\ \geq\ c]\!]_{\chi} =_{def} Q^{\leftarrow}[\![not\ (count(e) < c)]\!]_{\chi}\\
&Q^{\leftarrow}[\![count(e)\ <\ c]\!]_{\chi} =_{def} Q^{\leftarrow}[\![not\ (count(e) \geq c)]\!]_{\chi}\\
&Q^{\leftarrow}[\![count(e)\ \leq\ c]\!]_{\chi} =_{def} Q^{\leftarrow}[\![not\ (count(e) > c)]\!]_{\chi}
\end{aligned}
$$

FIG. 4.4: Dualité des contraintes de comptage

4.3.4 Illustration par l'exemple

La figure 4.5 illustre la traduction de l'expression child : : a [count (child : : b) > 2]. Cette expression sélectionne les noeuds *a* fils du noeud contexte χ qui ont au moins trois fils *b*. La formule $\mathcal{L}_{\text{PDL}}$ obtenue est valide pour les noeuds *a* sélectionnés par cette expression.
La première partie de la traduction désignée par φ, correspond à l'étape child : : a de la requête. Cette étape sélectionne les noeuds *a* candidats pour satisfaire le qualificateur [count (child : : b) > 2]. La deuxième partie de la traduction navigue de haut en bas les sous-arbres des noeuds *a* obtenus et vérifie s'ils ont plus de deux fils. Notons que la traduction de l'expression à compter count (child : : b) se fait selon les règles relatives aux expressions et aux chemins à l'intérieur des qualificateurs. Nous donnons ci-dessous les différentes étapes de cette traduction.

(1) $E^{\leftarrow}[\![child :: a\ [count(child :: b) > 2]]\!]_{\chi}$
(2) $P^{\leftarrow}[\![child :: a\ [count(child :: b) > 2]]\!]_{\chi}$
(3) $P^{\leftarrow}[\![child :: a]\!]_{\chi} \wedge Q^{\leftarrow}[\![count(child :: b) > 2]\!]_{\top}$
(4) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge Q^{\leftarrow}[\![count(child :: b) > 2]\!]_{\top}$
(5) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge Q^{\leftarrow}[\![\ count(E^{\leftarrow}[\![child :: b]\!]_{\top}) > 2\ ]\!]_{\top}$
(6) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge Q^{\leftarrow}[\![\ count(P^{\leftarrow}[\![child :: b]\!]_{\top}) > 2\ ]\!]_{\top}$
(7) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge Q^{\leftarrow}[\![\ count(A^{\leftarrow}[\![child]\!]_{b \wedge \top}) > 2\ ]\!]_{\top}$ or $b \wedge \top$ est équivalent à b
(8) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge Q^{\leftarrow}[\![\ count(A^{\leftarrow}[\![symmetric(child)]\!]_{b}) > 2\ ]\!]_{\top}$
(9) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge Q^{\leftarrow}[\![\ count(A^{\leftarrow}[\![parent]\!]_{b}) > 2\ ]\!]_{\top}$
(10) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge Q^{\leftarrow}[\![\ count(\langle 1; 2^{*} \rangle b) > 2\ ]\!]_{\top}$
(11) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge [\![\ count(\langle 1; 2^{*} \rangle b) > 2\ ]\!]_{\top}$
(12) $a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge [\![\ \#^{< 1; 2^{*} >} b > 2\ ]\!]_{\top}$



child : : a [count (child : : b) > 2]
$\underbrace{a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi}_{\varphi} \wedge \underbrace{[\![\#^{< 1; 2^{*} >} b > 2]\!]_{\top}}_{\psi}$

FIG. 4.5: Exemple de translation XPath

4.4 Traduction des formules PDL en μ-calcul

Une fois la translation des concepts XPath en PDL effectuée, la traduction des formules $\mathcal{L}_{\text{PDL}}$ vers le μ-calcul s'impose, cette traduction a pour objectif de produire les formules μ-calcul utilisées par la procédure de décision pour déterminer la satisfaisabilité des requêtes XPath.
Dans ce qui suit, nous détaillons la traduction respective des primitives de navigation, des formules génériques (sans comptage), des formules de comptage et des formules selon l'opérateur modulo.

4.4.1 Traduction des axes

Le tableau 4.1 spécifie la traduction des primitives de navigation, cette traduction consiste à donner pour chaque axe XPath, la formule équivalente en PDL et en μ-calcul.

Axe	**PDL**	**μ-calcul**
χ/child : :*	$\langle \overline{2}^*; \overline{1} \rangle\ \chi$	$\mu Z.\langle \overline{1} \rangle\ \chi \vee \langle \overline{2} \rangle\ Z$
χ/parent : :*	$\langle 1; 2^* \rangle\ \chi$	$\langle 1 \rangle\ \mu Z.\ \chi \vee \langle 2 \rangle\ Z$
χ/descendant : :*	$\langle (\overline{1} \cap \overline{2})^*; \overline{1} \rangle\ \chi$	$\mu Z.\langle \overline{1} \rangle\ (\chi \vee Z) \vee \langle \overline{2} \rangle\ Z$
χ/descendant-or-self : :*	$\langle \epsilon \cap (\overline{1} \cap \overline{2})^*; \overline{1} \rangle\ \chi$	$\mu Z.\chi \vee \mu Y.\langle \overline{1} \rangle\ (Y \vee Z) \vee \langle \overline{2} \rangle\ Y$
χ/ancestor : :*	$\langle 1; (1 \cap 2)^* \rangle\ \chi$	$\langle 1 \rangle\ \mu Z.\ \chi \vee \langle 1 \rangle\ Z \vee \langle 2 \rangle\ Z$
χ/ancestor-or-self : :*	$\langle (\epsilon \cap 1); (1 \cap 2)^* \rangle\ \chi$	$\mu Z.\chi \vee \langle 1 \rangle\ \mu(Y \vee Z) \vee \langle 2 \rangle\ Y$
χ/following-sibling : :*	$\langle \overline{2}^*; \overline{2} \rangle\ \chi$	$\mu Z.\langle \overline{2} \rangle\ \chi \vee \langle \overline{2} \rangle\ Z$
χ/preceding-sibling : :*	$\langle 2^*; 2 \rangle\ \chi$	$\mu Z.\langle 2 \rangle\ \chi \vee \langle 2 \rangle\ Z$

TAB. 4.1: Traduction des primitives de navigation

Formule PDL	Interprétation en μ-calcul	μ-calcul selon la sémantique de navigation	μ-calcul selon la sémantique de sélection
$[\![\varphi]\!]_{[\![\pi]\!]_\chi}$	$[\![\langle\pi\rangle\varphi]\!]_\chi$		
$[\![\langle \pi_1 \ ; \ \pi_2 \rangle\ \varphi]\!]_\chi$	$[\![\langle\pi_1\rangle\langle\pi_2\rangle\varphi]\!]_\chi$		
$[\![\langle \pi_1 \ \cup \ \pi_2 \rangle\ \varphi]\!]_\chi$	$[\![\langle\pi_1\rangle\varphi]\!]_\chi \vee [\![\langle\pi_2\rangle\varphi]\!]_\chi$		
$[\![\langle \pi_1 \ \cap \ \pi_2 \rangle\ \varphi]\!]_\chi$	$[\![\langle\pi_1\rangle\varphi]\!]_\chi \wedge [\![\langle\pi_2\rangle\varphi]\!]_\chi$		
$[\![\langle \rho \rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{([\![\rho]\!]_\chi)}$	$\varphi \wedge \langle \rho \rangle\ \chi$	$\chi \wedge \langle \rho \rangle\ \varphi$
$[\![\langle \overline{\rho} \rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{([\![\overline{\rho}]\!]_\chi)}$	$\varphi \wedge \langle \rho \rangle\ \chi$	$\chi \wedge \langle \overline{\rho} \rangle\ \varphi$
$[\![\langle \neg\pi \rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{([\![\neg\pi]\!]_\chi)}$	$\varphi \wedge \langle \neg\overline{\pi} \rangle\ \chi$	$\chi \wedge \langle \neg\pi \rangle\ \varphi$
$[\![\langle \pi^+ \rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{(\mu Z.[\![\pi]\!]_{(Z\vee\chi)})}$	$\varphi \wedge \mu Z.\chi \vee \langle\overline{\pi}\rangle Z$	$\chi \wedge \mu Z.\varphi \vee \langle\pi\rangle Z$
$[\![\langle \pi^* \rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{(\chi\vee\mu Z.[\![\pi]\!]_{(Z\vee\chi)})}$	$\chi \vee (\varphi \wedge \mu Z.\chi \vee \langle\overline{\pi}\rangle Z)$	$\chi \wedge (\varphi \vee \mu Z.\varphi \vee \langle\pi\rangle Z)$
$[\![\langle \varphi? \rangle\ \varphi\prime]\!]_\chi$	$[\![\varphi\prime]\!]_{(\varphi\wedge\chi)}$	$\varphi \wedge \varphi' \wedge \chi$	$\chi \wedge \varphi \wedge \varphi'$
$[\![\langle \rho/\varphi? \rangle\ \varphi']\!]_\chi$	$[\![\varphi']\!]_{([\![\varphi]\!]_{([\![\rho]\!]_\chi)})}$	$\varphi \wedge \varphi' \wedge \langle\overline{\rho}\rangle\chi$	$\chi \wedge \langle\rho\rangle(\varphi \wedge \varphi')$

TAB. 4.2: Traduction des formules génériques

4.4.2 Traduction des formules génériques

Nous désignons par formules génériques toute formule $\mathcal{L}_{\text{PDL}}$ sans contraintes de comptage. La traduction de ces formules met l'accent sur l'interprétation des programmes de navigation PDL en μ-calcul ; elle comprend les programmes atomiques, la négation de programme, le programme itératif, la clôture réflexive sur un programme, la concaténation, la disjonction et la conjonction de programmes ainsi que le programme de test et le chemin conditionné. Dans le tableau 4.2 nous donnons l'interprétation des formules génériques de PDL vers le μ-calcul ainsi que nous donnons pour les formules simples, une éventuelle écriture en μ-calcul selon les deux sémantiques de navigation et de sélection. En fait, l'interprétation sémantique des formules génériques présente deux cas : si la formule intervient dans la traduction d'une expression de chemin, son écriture en μ-calcul se fait selon la sémantique de navigation, sinon elle est écrite selon la sémantique de sélection.
Les remarques les plus intéressantes à l'égard de la traduction des formules génériques concernent la clôture réflexive des programmes et les chemins conditionnés : la clôture réflexive sur un programme peut être exprimée comme une disjonction entre le programme vide ϵ et l'itération sur le programme en question π^+, tandis que le chemin conditionné n'est que l'évaluation successive du programme atomique ρ et de la formule φ exprimant la condition, ensuite l'évaluation de la formule principale $\varphi\prime$ au cas ou la condition est vraie.

4.4.3 Traduction des contraintes de comptage

Nous abordons maintenant une phase assez complexe de la décidabilité des requêtes XPath avec comptage qui est la traduction des contraintes de comptage de PDL vers μ-calcul.

4.4.3.1 Formules de comptage

La traduction d'une formule de comptage de PDL vers le μ-calcul dépend essentiellement de la nature du chemin de comptage et de l'opérateur figurant dans la formule. En tenant compte de ces faits, nous avons proposé plusieurs traductions relatives à différents opérateurs de comptage et dont chacune est construite par itération sur l'ensemble des programmes de comptage. La question qui se pose à cet égard

$$
\begin{aligned}
Q^{\leftarrow}[\![count(e) \ < c]\!]_\chi &=_{def} Q^{\leftarrow}[\![not\ (count(e) \geq c)]\!]_\chi \\
Q^{\leftarrow}[\![count(e) \ \leq c]\!]_\chi &=_{def} Q^{\leftarrow}[\![not\ (count(e) > c)]\!]_\chi \\
Q^{\leftarrow}[\![count(e) \ != c]\!]_\chi &=_{def} Q^{\leftarrow}[\![not\ (count(e) = c)]\!]_\chi
\end{aligned}
$$

est comment représenter les chemins PDL en μ-calcul ? Pour chaque formule nous avons proposé une interprétation étroitement liée au chemin et qui consiste à décomposer le comptage en formules simples effectuant du calcul élémentaire qui fait référence à des formules génériques.

Tab. 4.3: Traduction des formules de comptage selon l'opérateur >

comptage selon l'opérateur >	
Formule générique	$[\![\#^{<\pi>}\varphi > c]\!]_\chi = \chi \wedge \langle\pi\rangle(\varphi \wedge [\![\#^{<\pi>}\varphi > c-1]\!]_\chi)$
Cas particulier	$c = 0 \qquad [\![\#^{<\pi>}\varphi > c]\!]_\chi = [\![\langle\pi\rangle\varphi]\!]_\chi$
cas $\pi = \pi_1;\pi_2$	$[\![\#^{<\pi_1;\,\pi_2>}\varphi > c]\!]_\chi = [\![\#^{<\pi_1>}\varphi > c]\!]_\chi \vee [\![\#^{<\pi_2>}\varphi > c]\!]_{[\![\pi_1]\!]_\chi} \vee$ $([\![\#^{<\pi_1>}\varphi = 1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi > c-1]\!]_{[\![\pi_1]\!]_\chi}) \vee$ $([\![\#^{<\pi_1>}\varphi = 2]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi > c-2]\!]_{[\![\pi_1]\!]_\chi}) \vee \ldots \vee$ $([\![\#^{<\pi_1>}\varphi = c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi > 0]\!]_{[\![\pi_1]\!]_\chi})$
cas $\pi = \pi_1 \cap \pi_2$	$[\![\#^{<\pi_1\cap\,\pi_2>}\varphi > c]\!]_\chi = [\![\#^{<\pi_1>}\varphi > c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi > c]\!]_\chi$
cas $\pi = \pi_1 \cup \pi_2$	$[\![\#^{<\pi_1\cup\,\pi_2>}\varphi > c]\!]_\chi = [\![\#^{<\pi_1>}\varphi > c]\!]_\chi \vee [\![\#^{<\pi_2>}\varphi > c]\!]_\chi \vee$ $([\![\#^{<\pi_1>}\varphi = 1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi > c-1]\!]_\chi) \vee$ $([\![\#^{<\pi_1>}\varphi = 2]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi > c-2]\!]_\chi) \vee \ldots \vee$ $([\![\#^{<\pi_1>}\varphi = c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi > 0]\!]_\chi)$
cas $\pi = \pi^+$	$[\![\#^{<\pi^+>}\varphi > c]\!]_\chi = \chi \wedge < \pi^+ > (\varphi \wedge [\![\#^{<\pi^+>}\varphi > c-1]\!]_\top)$
cas $\pi = \pi^*$	$[\![\#^{<\pi^*>}\varphi > c]\!]_\chi = \chi \wedge ((\varphi \wedge [\![\#^{<\pi^+>}\varphi > c-1]\!]_\top) \vee [\![\#^{<\pi^+>}\varphi > c]\!]_\top)$
cas $\pi = \neg\pi$	$[\![\#^{<\neg\pi>}\varphi > c]\!]_\chi = \bot$
cas $\pi = \rho \mid \overline{\rho}$	$[\![\#^{<\rho>}\varphi > c]\!]_\chi = \bot \qquad [\![\#^{<\overline{\rho}>}\varphi > c]\!]_\chi = \bot$
cas $\pi = \varphi?$	$[\![\#^{<\varphi'?>}\varphi > c]\!]_\chi = \bot$
cas $\pi = \rho/\varphi?$	$[\![\#^{<\rho/\varphi?>}\varphi' > c]\!]_\chi = \bot$

TAB. 4.4: Traduction des formules de comptage selon l'opérateur ≥

comptage selon l'opérateur ≥	
Formule générique	$[\![\#^{<\pi>}\varphi \geq c]\!]_\chi = \chi \wedge \langle\pi\rangle([\![\#^{<\pi>}\varphi \geq c-1]\!]_\chi)$
Cas particuliers	$c = 0$ $\quad [\![\#^{<\pi>}\varphi \geq c]\!]_\chi = \top$ $c = 1$ $\quad [\![\#^{<\pi>}\varphi \geq c]\!]_\chi = [\![\langle\pi\rangle\varphi]\!]_\chi$
cas $\pi = \pi_1;\pi_2$	$[\![\#^{<\pi_1;\,\pi_2>}\varphi \geq c]\!]_\chi = [\![\#^{<\pi_1>}\varphi \geq c]\!]_\chi \vee [\![\#^{<\pi_2>}\varphi \geq c]\!]_{[\![\pi_1]\!]_\chi} \vee$ $([\![\#^{<\pi_1>}\varphi = 1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \geq c-1]\!]_{[\![\pi_1]\!]_\chi}) \vee$ $([\![\#^{<\pi_1>}\varphi = 2]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \geq c-2]\!]_{[\![\pi_1]\!]_\chi}) \vee ... \vee$ $([\![\#^{<\pi_1>}\varphi = c-1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \geq 1]\!]_{[\![\pi_1]\!]_\chi})$
cas $\pi = \pi_1 \cap \pi_2$	$[\![\#^{<\pi_1 \cap \pi_2>}\varphi \geq c]\!]_\chi = [\![\#^{<\pi_1>}\varphi \geq c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \geq c]\!]_\chi$
cas $\pi = \pi_1 \cup \pi_2$	$[\![\#^{<\pi_1 \cup \pi_2>}\varphi \geq c]\!]_\chi = [\![\#^{<\pi_1>}\varphi \geq c]\!]_\chi \vee [\![\#^{<\pi_2>}\varphi \geq c]\!]_\chi \vee$ $([\![\#^{<\pi_1>}\varphi = 1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \geq c-1]\!]_\chi) \vee$ $([\![\#^{<\pi_1>}\varphi = 2]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \geq c-2]\!]_\chi) \vee ... \vee$ $([\![\#^{<\pi_1>}\varphi = c-1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \geq 1]\!]_\chi)$
cas $\pi = \pi^+$	$[\![\#^{<\pi^+>}\varphi \geq c]\!]_\chi = \chi \wedge < \pi^+ > \left(\varphi \wedge [\![\#^{<\pi^+>}\varphi \geq c-1]\!]_\top\right)$
cas $\pi = \pi^*$	$[\![\#^{<\pi^*>}\varphi \geq c]\!]_\chi = \chi \wedge \left((\varphi \wedge [\![\#^{<\pi^+>}\varphi \geq c-1]\!]_\top) \vee [\![\#^{<\pi^+>}\varphi \geq c]\!]_\top\right)$
cas $\pi = \neg\pi$	$[\![\#^{<\neg\pi>}\varphi \geq c]\!]_\chi = \bot$
cas $\pi = \rho \mid \overline{\rho}$	$[\![\#^{<\rho>}\varphi \geq c]\!]_\chi = \bot$ $\qquad$ $[\![\#^{<\overline{\rho}>}\varphi \geq c]\!]_\chi = \bot$
cas $\pi = \varphi?$	$[\![\#^{<\varphi'?>}\varphi \geq c]\!]_\chi = \bot$
cas $\pi = \rho/\varphi?$	$[\![\#^{<\rho/\varphi?>}\varphi' \geq c]\!]_\chi = \bot$

Dans ce qui suit, nous donnons l'interprétation des formules de comptage selon les opérateurs (= , >, ≥). La traduction de ces formules selon le reste des opérateurs peut être déduite en se basant sur la propriété de dualité. En fait, le comptage selon les opérateurs (<, ≤, ! =) se fait en appliquant l'opérateur de négation aux résultats booléens du comptage selon les opérateurs (=, >, ≥).

Les tableaux 4.3, 4.4 et 4.5 détaillent l'interprétation proposée pour le comptage selon les opérateurs cités. Pour chaque opérateur, nous donnons en premier lieu les cas particuliers relatifs à la constante de comptage (généralement 0 ou 1), ensuite nous itérons sur la typologie du chemin à parcourir pour donner une interprétation logique pour chaque chemin.

La traduction du comptage selon la composition et la disjonction de programmes ($\pi_1;\pi_2$ et $\pi_1 \cup \pi_2$) consiste à énumérer, en termes de formules de comptage simples, les différents cas possibles pour le comptage des noeuds satisfaisant φ et accessibles par π_1 et par π_2 à partir du noeud contexte. Cette interprétation respecte les conditions de monotonie et de comptage ordonnée puisqu'elle est composée d'un ensemble de formules de comptage simples qui adhèrent au principe général de comptage présenté dans le début du chapitre.

En ce qui concerne le comptage selon π^+, nous avons proposé une interprétation qui effectue le parcours du chemin du comptage à partir du noeud contexte et décrémente le nombre de comptage chaque fois que φ est satisfaite. Cette interprétation n'est que l'application du principe général de comptage dans le contexte des programmes itératifs.

Le comptage selon π^* est traduit comme une disjonction de deux possibilités : si le noeud contexte satisfait la formule de comptage φ, nous décrémentons le nombre de noeuds à trouver par 1, ensuite nous continuons le comptage selon π^+. Sinon, nous effectuons directement la totalité du comptage selon π^+.

Il est à noter que le comptage selon un programme nié n'a pas de sens , donc toute formule de comptage selon $\neg\pi$ ne sera jamais satisfaite.

Il reste à souligner que le comptage selon un programme atomique consiste en une seule étape donc il ne

TAB. 4.5: Traduction des formules de comptage selon l'opérateur =

comptage selon l'opérateur =	
Formule générique	$[\![\#^{<\pi>}\varphi = c]\!]_\chi = \chi \wedge \langle\pi\rangle(\varphi \wedge [\![\#^{<\pi>}\varphi = c-1]\!]_\chi)$
Cas particuliers	$c = 0$ $\quad [\![\#^{<\pi>}\varphi \geq c]\!]_\chi = [\![\langle\pi\rangle\neg\varphi]\!]_\chi$ $c = 1$ $\quad [\![\#^{<\pi>}\varphi \geq c]\!]_\chi = [\![\langle\pi\rangle\varphi]\!]_\chi$
cas $\pi = \pi_1;\pi_2$	$[\![\#^{<\pi_1;\,\pi_2>}\varphi = c]\!]_\chi = ([\![\#^{<\pi_1>}\varphi = 0]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = c]\!]_{[\![\pi_1]\!]_\chi}) \vee$ $([\![\#^{<\pi_1>}\varphi = 1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = c-1]\!]_{[\![\pi_1]\!]_\chi}) \vee \ldots \vee$ $([\![\#^{<\pi_1>}\varphi = 2]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = c-2]\!]_{[\![\pi_1]\!]_\chi}) \vee \ldots \vee$ $([\![\#^{<\pi_1>}\varphi = c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = 0]\!]_{[\![\pi_1]\!]_\chi})$
cas $\pi = \pi_1 \cap \pi_2$	$[\![\#^{<\pi_1\cap\,\pi_2>}\varphi = c]\!]_\chi = [\![\#^{<\pi_1>}\varphi = c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = c]\!]_\chi$
cas $\pi = \pi_1 \cup \pi_2$	$[\![\#^{<\pi_1\cup\,\pi_2>}\varphi = c]\!]_\chi = ([\![\#^{<\pi_1>}\varphi = 0]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = c]\!]_\chi) \vee$ $([\![\#^{<\pi_1>}\varphi = 1]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = c-1]\!]_\chi) \vee \ldots \vee$ $([\![\#^{<\pi_1>}\varphi = 2]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = c-2]\!]_\chi) \vee \ldots \vee$ $([\![\#^{<\pi_1>}\varphi = c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi = 0]\!]_\chi)$
cas $\pi = \pi^+$	$[\![\#^{<\pi^+>}\varphi = c]\!]_\chi = \chi \wedge <\pi^+>(\varphi \wedge [\![\#^{<\pi^+>}\varphi = c-1]\!]_\top)$
cas $\pi = \pi^*$	$[\![\#^{<\pi^*>}\varphi = c]\!]_\chi = \chi \wedge ((\varphi \wedge [\![\#^{<\pi^+>}\varphi = c-1]\!]_\top) \vee [\![\#^{<\pi^+>}\varphi = c]\!]_\top)$
cas $\pi = \neg\pi$	$[\![\#^{<\neg\pi>}\varphi = c]\!]_\chi = \bot$
cas $\pi = \rho \mid \overline{\rho}$	$[\![\#^{<\rho>}\varphi = c]\!]_\chi = \bot$ $\qquad [\![\#^{<\overline{\rho}>}\varphi = c]\!]_\chi = \bot$
cas $\pi = \varphi?$	$[\![\#^{<\varphi'?>}\varphi = c]\!]_\chi = \bot$
cas $\pi = \rho/\varphi?$	$[\![\#^{<\rho/\varphi?>}\varphi' = c]\!]_\chi = \bot$

peut avoir de sens que lorsque la constante de comptage c équivaut à 0 ou 1 selon le chemin et l'opérateur de comptage considérés.

4.4.4 Traduction des contraintes modulo

L'interprétation des formules de comptage selon l'opérateur *modulo* présente les mêmes propriétés que la traduction précédente dans le sens où elle dépend essentiellement de la nature du chemin de comptage et qu'elle doit satisfaire les contraintes de monotonie et du comptage ordonné.
Dans cette traduction, l'interprétation du comptage modulo selon la composition et la disjonction de programmes ($\pi_1;\pi_2$ et $\pi_1 \cup \pi_2$) consiste à vérifier, que le nombre de noeuds comptés par le chemin considéré est différent de la constante de comptage c et de ses multiples. Cette vérification se fait par étapes, pour chaque ordre de comptage i, il faut s'assurer que le nombre de noeuds satisfaisant φ est différent de $i * c$ et inférieur à $(i+1) * c$. Si ces conditions sont vérifiées à une étape donnée, le comptage s'arrête et la formule est vérifiée.
Le comptage modulo selon le programme itératif π^+ est interprété comme une évaluation récursive du fait que le nombre des noeuds satisfaisant φ soit différent de c. Cette récursivité s'arrête lorsqu'aucun noeud n'est accessible par π à partir du noeud courant. Tandis que le comptage modulo selon le programme π^* n'est que le comptage selon la disjonction du chemin vide ϵ et le chemin itératif π^+. Le tableau 4.6 donne les formules de cette traduction construite par itération sur les programmes de comptage.

Tab. 4.6: Traduction des formules de comptage selon l'opérateur modulo

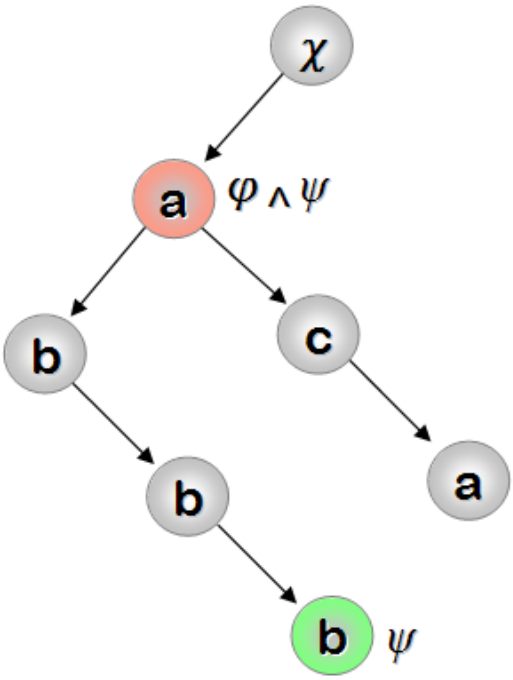
comptage selon l'opérateur *modulo*	
Cas particulier	$c = 0$ ou $c = 1$ $\quad [\![\#^{<\pi>}\varphi \equiv_{(m)} c]\!]_\chi = \bot$
cas $\pi = \pi_1;\pi_2$	$[\![\#^{<\pi_1;\,\pi_2>}\varphi \equiv_{(m)} c]\!]_\chi = [\![\#^{<\pi_1;\,\pi_2>}\varphi < c]\!]_\chi \vee$ $\big([\![\#^{<\pi_1;\,\pi_2>}\varphi != c]\!]_\chi \wedge \big([\![\#^{<\pi_1;\,\pi_2>}\varphi < 2c]\!]_\chi \vee$ $([\![\#^{<\pi_1;\,\pi_2>}\varphi != 2c]\!]_\chi \wedge ([\![\#^{<\pi_1;\,\pi_2>}\varphi < 3c]\!]_\chi \vee (...)))\big)\big)$
cas $\pi = \pi_1 \cap \pi_2$	$[\![\#^{<\pi_1 \cap\, \pi_2>}\varphi \equiv_{(m)} c]\!]_\chi = [\![\#^{<\pi_1>}\varphi \equiv_{(m)} c]\!]_\chi \wedge [\![\#^{<\pi_2>}\varphi \equiv_{(m)} c]\!]_\chi$
cas $\pi = \pi_1 \cup \pi_2$	$[\![\#^{<\pi_1 \cup\, \pi_2>}\varphi \equiv_{(m)} c]\!]_\chi = [\![\#^{<\pi_1 \cup\, \pi_2>}\varphi < c]\!]_\chi \vee$ $\big([\![\#^{<\pi_1 \cup\, \pi_2>}\varphi != c]\!]_\chi \wedge \big([\![\#^{<\pi_1 \cup\, \pi_2>}\varphi < 2c]\!]_\chi \vee$ $([\![\#^{<\pi_1 \cup\, \pi_2>}\varphi != 2c]\!]_\chi \wedge ([\![\#^{<\pi_1 \cup\, \pi_2>}\varphi < 3c]\!]_\chi \vee (...)))\big)\big)$
cas $\pi = \pi^+$	$[\![\#^{<\pi^+>}\varphi c]\!]_\chi = [\![\#^{<\pi^+>}\varphi != c]\!]_{[\![\#^{<\pi^+>}\varphi != c]\!]} \ddots_{[\![\#^{<\pi^+>}\varphi != c]\!]_\chi}$
cas $\pi = \pi^*$	$[\![\#^{<\pi^*>}\varphi \equiv_{(m)} c]\!]_\chi = [\![\#^{<\epsilon \cup \pi^+>}\varphi \equiv_{(m)} c]\!]_\chi$
cas $\pi = \neg\pi$	$[\![\#^{<\neg\pi>}\varphi \equiv_{(m)} c]\!]_\chi = \bot$
cas $\pi = \rho \mid \overline{\rho}$	$[\![\#^{<\rho>}\varphi \equiv_{(m)} c]\!]_\chi = \top \qquad [\![\#^{<\overline{\rho}>}\varphi \equiv_{(m)} c]\!]_\chi = \top$
cas $\pi = \varphi?$	$[\![\#^{<\varphi'?>}\varphi \equiv_{(m)} c]\!]_\chi = \top$
cas $\pi = \rho/\varphi?$	$[\![\#^{<\rho/\varphi?>}\varphi \equiv_{(m)} c]\!]_\chi = \top$

4.4.5 Illustration par l'exemple

Pour illustrer la traduction des formules PDL vers le μ-calcul, nous proposons de continuer l'exemple abordé dans le paragraphe 4.3.4. Dans cet exemple nous avons traduit l'expression XPath child : : a [count (child : : b) > 2] en PDL ce qui a produit la formule suivante :
$a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi \wedge [\![\#^{<1;2^{*}>} b > 2]\!]_{\top}$.
Cette formule comprend deux sous-formules (φ et ψ). La traduction de φ illustre l'application des règles de traduction relatives aux formules génériques, et particulièrement les formules correspondantes à la composition et à la clôture réflexive des programmes, tandis que la traduction de la deuxième sous-formule s'appuie sur les règles relatives aux formules de comptage selon l'opérateur >.
Pour assurer la lisibilité de la traduction et des formules obtenues, nous n'avons pas détaillé la traduction des formules génériques (qui sont des détails faciles à déduire à partir des règles données dans le paragraphe 4.4.2). Le tableau 4.7 montre comment traduire la formule considérée en μ-calcul et la figure 4.6 représente une structure XML satisfaisant cette formule.

Tab. 4.7: Illustration (détails de la traduction)

(1)	$[\![a \wedge \langle \overline{2}^{*} ; \overline{1} \rangle \chi]\!]_{\top} \wedge [\![\#^{<1;2^{*}>} b > 2]\!]_{\top}$
(2)	$a \wedge [\![\langle \overline{2}^{*} \rangle \langle \overline{1} \rangle \chi]\!]_{\top} \wedge [\![\#^{<1;2^{*}>} b > 2]\!]_{\top}$
(3)	$a \wedge \left(\langle \overline{1} \rangle \chi \vee \mu Z . \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z \right) \wedge [\![\#^{<1;2^{*}>} b > 2]\!]_{\top}$
(4)	$a \wedge \left(\langle \overline{1} \rangle \chi \vee \mu Z . \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z \right) \wedge \big([\![\#^{<1>} b > 2]\!]_{\top} \vee [\![\#^{<2^{*}>} b > 2]\!]_{[\![1]\!]_{\top}} \vee$ $([\![\#^{<1>} b = 1]\!]_{\top} \wedge [\![\#^{<2^{*}>} b > 1]\!]_{[\![1]\!]_{\top}}) \vee ([\![\#^{<1>} b = 2]\!]_{\top} \wedge [\![\#^{<2^{*}>} b > 0]\!]_{[\![1]\!]_{\top}}) \big)$
(5)	$a \wedge \left(\langle \overline{1} \rangle \chi \vee \mu Z . \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z \right) \wedge \big((\bot \vee [\![\#^{<2^{*}>} b > 2]\!]_{[\![1]\!]_{\top}}) \vee$ $(b \wedge [\![\#^{<2^{*}>} b > 1]\!]_{[\![1]\!]_{\top}}) \vee (\bot \wedge [\![\#^{<2^{*}>} b > 0]\!]_{[\![1]\!]_{\top}}) \big)$
(6)	$a \wedge \left(\langle \overline{1} \rangle \chi \vee \mu Z . \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z \right) \wedge$ $\big((\langle 1 \rangle ((b \wedge [\![\#^{<2^{+}>} b > 1]\!]_{\top}) \vee [\![\#^{<2^{+}>} b > 2]\!]_{\top})) \vee$ $(b \wedge \langle 1 \rangle ((b \wedge [\![\#^{<2^{+}>} b > 0]\!]_{\top}) \vee [\![\#^{<2^{+}>} b > 1]\!]_{\top})) \big)$
(7)	$a \wedge \left(\langle \overline{1} \rangle \chi \vee \mu Z . \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z \right) \wedge$ $\big((\langle 1 \rangle ((b \wedge \langle 2^{+} \rangle b \wedge [\![\#^{<2^{+}>} b > 0]\!]_{\top})) \vee \langle 2^{+} \rangle (b \wedge [\![\#^{<2^{+}>} b > 1]\!]_{\top}))) \vee$ $(b \wedge \langle 1 \rangle ((b \wedge \langle 2^{+} \rangle b) \vee \langle 2^{+} \rangle (b \wedge [\![\#^{<2^{+}>} b > 0]\!]_{\top}))) \big)$
(8)	$a \wedge \left(\langle \overline{1} \rangle \chi \vee \mu Z . \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z \right) \wedge$ $\big((\langle 1 \rangle ((b \wedge \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b)) \vee \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle (b \wedge [\![\#^{<2^{+}>} b > 0]\!]_{\top})))) \vee$ $(b \wedge \langle 1 \rangle ((b \wedge \langle 2^{+} \rangle b) \vee \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b)))) \big)$
(9)	$a \wedge \left(\langle \overline{1} \rangle \chi \vee \mu Z . \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z \right) \wedge$ $\big((\langle 1 \rangle ((b \wedge \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b)) \vee \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b)))) \vee$ $(b \wedge \langle 1 \rangle ((b \wedge \langle 2^{+} \rangle b) \vee \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b)))) \big)$



$$\underbrace{\text{child} :: \text{a}\ [\text{count}\ (\text{child} :: \text{b}) > 2]}_{\varphi}$$

$$\underbrace{a \wedge \langle \bar{2}^{*} ; \bar{1} \rangle \chi}_{\varphi} \wedge \underbrace{[\![\#^{< 1 ; 2^{*} >} b > 2]\!]_{\top}}_{\psi}$$

$$\underbrace{a \wedge \left(\langle \bar{1} \rangle \chi \vee \mu Z . \langle \bar{1} \rangle \chi \vee \langle \bar{2} \rangle Z \right)}_{\varphi} \wedge$$

$$\underbrace{\left((\langle 1 \rangle ((b \wedge \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b)) \vee \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b)))) \vee (b \wedge \langle 1 \rangle ((b \wedge \langle 2^{+} \rangle b) \vee \langle 2^{+} \rangle (b \wedge \langle 2^{+} \rangle b))) \right)}_{\psi}$$

FIG. 4.6: Exemple de translation de PDL vers μ-calcul

Chapitre 5

Comptage global et ordre du document

Dans ce chapitre, nous présentons une alternative à l'approche par décomposition de comptage. Notre deuxième approche consiste à effectuer un comptage global selon des formules composites traduisant les contraintes de comptage XPath.
Pour assurer la monotonie de ce comptage complexe nous définissons un environnement de détection de cycle qui vérifie la présence des cycles au sein de chaque programme figurant dans la formule de comptage. En ce qui concerne l'ordre de comptage, il est assuré par l'ordre du document XML qui est l'ordre d'apparition des noeuds dans le document.
Dans la suite du chapitre, nous détaillons les solutions proposées pour répondre aux exigences de monotonie et du comptage ordonné, nous montrons par la suite comment traduire les concepts XPath en PDL puis en μ-calcul selon cette approche. Pour conclure, nous procédons à une discussion portant sur la comparaison entre les deux approches proposées et la synthèse de leurs avantages et leurs inconvénients.

5.1 Exigences de comptage

La translation des concepts XPath doit satisfaire deux contraintes principales, à savoir : la monotonie du chemin et la définition d'un ordre de comptage. Ces exigences ont pour objectif de garantir la sûreté et l'efficacité lors du dénombrement des noeuds concernés par une requête XPath avec comptage.

5.1.1 La monotonie du chemin

La monotonie du chemin est une condition nécessaire pour remédier aux problèmes des boucles et des cycles infinis dans les chemins de comptage. L'analyse des constructeurs de programmes montre que seul l'opérateur de composition de programmes (;) peut engendrer des cycles dans les formules de comptage. Un cycle est de la forme $\langle \pi ; \overline{\pi} \rangle$ avec $\pi \in \{\rho, \rho^+, \rho^*\}$ et $\overline{\pi} \in \{\overline{\rho}, \overline{\rho}^+, \overline{\rho}^*\}$.
Pour résoudre ce problème nous avons introduit un environnement de détection de cycles qui vérifie la présence des cycles au sein de chaque programme figurant dans la formule de comptage considérée.
Dans l'évaluation $\Delta \| \Gamma \vdash^{\pi} \varphi$, Δ désigne l'environnement de détection des cycles, Γ est une évaluation indiquant la présence de cycles pour les différents chemins de la formule φ, tandis que π représente l'ensemble des chemins à vérifier.
L'évaluation Γ boucle sur les chemins de comptage et teste l'occurrence d'une composition de programmes inverses dans chaque chemin. Une évaluation peut avoir trois valeurs possibles : _ (aucun programme n'est encore détecté), π (le dernier programme consistant) ou $\perp$ (un cycle a été détecté).
Pour détecter les cycles nous utilisons un opérateur auxiliaire $\lhd$ qui est défini comme suit :
$\Gamma \lhd \pi =_{def} \{C : \Gamma(C) = \pi\}$ avec C le chemin de comptage à vérifier.
Une formule est dite sans cycle si aucun de ses chemins ne vérifie ($\Gamma(C) = \perp$). Le tableau 5.1 décrit la grille de détection de cycles selon l'opérateur $\lhd$.

Tab. 5.1: Grille de détection des cycles

$\cdot \lhd \cdot$	1	2	$\overline{1}$	$\overline{2}$	1^+	2^+	$\overline{1}^+$	$\overline{2}^+$	1^*	2^*	$\overline{1}^*$	$\overline{2}^*$
_	1	2	$\overline{1}$	$\overline{2}$	1^+	2^+	$\overline{1}^+$	$\overline{2}^+$	1^*	2^*	$\overline{1}^*$	$\overline{2}^*$
1	1	2	$\perp$	$\overline{2}$	1^+	2^+	$\perp$	$\overline{2}^+$	1^*	2^*	$\perp$	$\overline{2}^*$
2	1	2	$\overline{1}$	$\perp$	1^+	2^+	$\overline{1}^+$	$\perp$	1^*	2^*	$\overline{1}^*$	$\perp$
$\overline{1}$	$\perp$	2	$\overline{1}$	$\overline{2}$	$\perp$	2^+	$\overline{1}^+$	$\overline{2}^+$	$\perp$	2^*	$\overline{1}^*$	$\overline{2}^*$
$\overline{2}$	1	$\perp$	$\overline{1}$	$\overline{2}$	1^+	$\perp$	$\overline{1}^+$	$\overline{2}^+$	1^*	$\perp$	$\overline{1}^*$	$\overline{2}^*$
1^+	1	2	$\perp$	$\overline{2}$	1^+	2^+	$\perp$	$\overline{2}^+$	1^*	2^*	$\perp$	$\overline{2}^*$
2^+	1	2	$\overline{1}$	$\perp$	1^+	2^+	$\overline{1}^+$	$\perp$	1^*	2^*	$\overline{1}^*$	$\perp$
$\overline{1}^+$	$\perp$	2	$\overline{1}$	$\overline{2}$	$\perp$	2^+	$\overline{1}^+$	$\overline{2}^+$	$\perp$	2^*	$\overline{1}^*$	$\overline{2}^*$
$\overline{2}^+$	1	$\perp$	$\overline{1}$	$\overline{2}$	1^+	$\perp$	$\overline{1}^+$	$\overline{2}^+$	1^*	$\perp$	$\overline{1}^*$	$\overline{2}^*$
1^*	1	2	$\perp$	$\overline{2}$	1^+	2^+	$\perp$	$\overline{2}^+$	1^*	2^*	$\perp$	$\overline{2}^*$
2^*	1	2	$\overline{1}$	$\perp$	1^+	2^+	$\overline{1}^+$	$\perp$	1^*	2^*	$\overline{1}^*$	$\perp$
$\overline{1}^*$	$\perp$	2	$\overline{1}$	$\overline{2}$	$\perp$	2^+	$\overline{1}^+$	$\overline{2}^+$	$\perp$	2^*	$\overline{1}^*$	$\overline{2}^*$
$\overline{2}^*$	1	$\perp$	$\overline{1}$	$\overline{2}$	1^+	$\perp$	$\overline{1}^+$	$\overline{2}^+$	1^*	$\perp$	$\overline{1}^*$	$\overline{2}^*$

5.1.2 Comptage ordonné

Le dénombrement multiple des noeuds satisfaisant une formule de comptage donnée est l'un des problèmes majeurs auxquels le comptage fait face. Pour remédier à ce problème, il faut établir une relation d'ordre entre les noeuds concernés par le comptage.
Une première proposition consiste à utiliser l'ordre de comptage proposé pour l'approche par décomposition de comptage qui est assuré par la monotonie du chemin. Cette solution n'est pas valable dans le contexte du comptage global puisque le chemin est composite et par conséquent non monotone, ainsi il est possible dans ce cas de parcourir le même noeud plusieurs fois par ce chemin étendu.

Face à ces faits, nous avons opté pour une autre convention qui repose sur l'ordre du document (ordre d'apparition des noeuds dans la structure XML) pour définir une relation d'ordre entre les noeuds cibles du comptage. En fait, l'ordre du document permet de définir une relation de précédence (notée $<<$) sur l'ensemble des noeuds de l'arbre XML qui est décrite comme suit :

$$<< = (1;2^*)^+ \cup ((\overline{2}^*;\overline{1})^*;2^+;(1;2^*)^*)$$

Soit a et b deux noeuds d'une structure XML, nous affirmons que a précède b (noté $a << b$) si la relation $a \wedge \langle(1;2^*)^+ \cup ((\overline{2}^*;\overline{1})^*;2^+;(1;2^*)^*)\rangle\ b$ est vrai. Cette relation implique qu'à partir du noeud a on peut obtenir le noeud b selon le chemin $(1;2^*)^+ \cup ((\overline{2}^*;\overline{1})^*;2^+;(1;2^*)^*)$.
Nous pouvons encore détailler cette relation de précédence en énumérant les différents cas de précédence possibles entre les deux noeuds :

1. $a \wedge \langle(1;2^*)^+\rangle\ b$
   b est un fils de a.
2. $a \wedge \langle 2^+\rangle\ b$
   b est un frère postérieur à a.
3. $a \wedge \langle 2^+;(1;2^*)^*\rangle\ b$
   b est un fils de l'un des noeuds frères postérieur à a.
4. $a \wedge \langle(\overline{2}^*;\overline{1})^*;2^+\rangle\ b$
   b est un frère postérieur à un ancêtre du noeud a.
5. $a \wedge \langle(\overline{2}^*;\overline{1})^*;2^+;(1;2^*)^*\rangle\ b$
   b est un fils de l'un des noeuds frère postérieur à un noeud ancêtre de a

Pour bien expliquer l'ordre du document, nous donnons la figure 5.1 qui décrit une structure XML dont les noeuds sont numérotés de 0 à 9. La relation de précédence entre ces noeuds implique l'ordre suivant : 0-1-4-7-8-9-5-6-2-3.

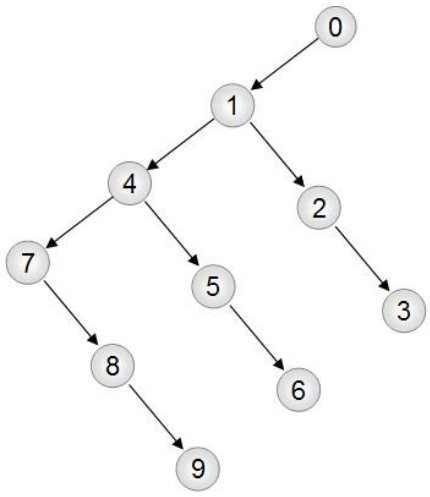


FIG. 5.1: Relation de précédence selon l'ordre du document

5.2 Translation des concepts XPath en PDL

La translation des concepts XPath en PDL présente les mêmes caractéristiques que celle proposée pour la première approche dans la mesure où elle adhère à la sémantique XPath et assimile l'exploration des structures XML à une navigation dans l'arbre binaire équivalent.
En ce qui concerne l'interprétation sémantique des concepts XPath, nous avons adopté la sémantique de navigation pour les expressions des axes et des chemins, et la sémantique de sélection pour les qualificateurs. Ces choix sont justifiés par les propriétés intrinsèques des chemins et des axes comme des primitives de navigation de la structure XML, et les propriétés des qualificateurs comme des expressions de sélection et de qualification.

$A^{\leftarrow}[\![\cdot]\!] : Axis \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$A^{\leftarrow}[\![self]\!]_\chi =_{def} \chi$
$A^{\leftarrow}[\![child]\!]_\chi =_{def} \langle \overline{2}^* ; \overline{1} \rangle \chi$
$A^{\leftarrow}[\![following - sibling]\!]_\chi =_{def} \langle \overline{2}^* ; \overline{2} \rangle \chi$
$A^{\leftarrow}[\![preceding - sibling]\!]_\chi =_{def} \langle 2^* ; 2 \rangle \chi$
$A^{\leftarrow}[\![parent]\!]_\chi =_{def} \langle 1 ; 2^* \rangle \chi$
$A^{\leftarrow}[\![descendant]\!]_\chi =_{def} \langle (\overline{1} \mid \overline{2})^* ; \overline{1} \rangle \chi$
$A^{\leftarrow}[\![descendant - or - self]\!]_\chi =_{def} \langle \epsilon \mid (\overline{1} \mid \overline{2})^* ; \overline{1} \rangle \chi$
$A^{\leftarrow}[\![ancestor]\!]_\chi =_{def} \langle 1 ; (1 \mid 2)^* \rangle \chi$
$A^{\leftarrow}[\![ancestor - or - self]\!]_\chi =_{def} \langle \epsilon \mid 1 ; (1 \mid 2)^* \rangle \chi$
$A^{\leftarrow}[\![following]\!]_\chi =_{def} A^{\leftarrow}[\![descendant - or - self]\!]_{\eta_1}$
$A^{\leftarrow}[\![preceding]\!]_\chi =_{def} A^{\leftarrow}[\![descendant - or - self]\!]_{\eta_2}$
$\eta_1 =_{def} A^{\leftarrow}[\![following - sibling]\!]_{A^{\leftarrow}[\![ancestor-or-self]\!]_\chi}$
$\eta_1 =_{def} A^{\leftarrow}[\![preceding - sibling]\!]_{A^{\leftarrow}[\![ancestor-or-self]\!]_\chi}$

FIG. 5.2: Translation des axes

$E^{\leftarrow}[\![\cdot]\!] : \mathcal{L}_{\text{XPath}} \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$E^{\leftarrow}[\![/p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\chi \wedge \langle \overline{2}^* ; \neg\overline{1} \rangle \top)}$
$E^{\leftarrow}[\![p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_\chi$
$E^{\leftarrow}[\![e_1 \mid e_2]\!]_\chi =_{def} E^{\leftarrow}[\![e_1]\!]_\chi \vee E^{\leftarrow}[\![e_2]\!]_\chi$
$E^{\leftarrow}[\![e_1 \cap e_2]\!]_\chi =_{def} E^{\leftarrow}[\![e_1]\!]_\chi \wedge E^{\leftarrow}[\![e_2]\!]_\chi$

$P^{\leftarrow}[\![\cdot]\!] : Path \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$P^{\leftarrow}[\![p^+]\!]_\chi =_{def} (P^{\leftarrow}[\![p]\!]_\top)^+ \chi$
$P^{\leftarrow}[\![p^*]\!]_\chi =_{def} (P^{\leftarrow}[\![p]\!]_\top)^* \chi$
$P^{\leftarrow}[\![p_1 / p_2]\!]_\chi =_{def} P^{\leftarrow}[\![p_2]\!]_{P^{\leftarrow}[\![p_1]\!]_\chi}$
$P^{\leftarrow}[\![p/q?]\!]_\chi =_{def} P^{\leftarrow}[\![p/\texttt{self :: *[q]}]\!]_\chi$
$P^{\leftarrow}[\![p[q]]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\chi)} \wedge Q^{\leftarrow}[\![q]\!]_\top$
$P^{\leftarrow}[\![a :: \sigma]\!]_\chi =_{def} \sigma \wedge A^{\leftarrow}[\![a]\!]_{(\chi)}$
$P^{\leftarrow}[\![a :: *]\!]_\chi =_{def} A^{\leftarrow}[\![a]\!]_\chi$

FIG. 5.3: Translation des expressions et des chemins

Les deux solutions de comptage proposées ont une traduction commune aux expressions des axes et des chemins XPath, mais leur différence intervient au niveau des qualificateurs puisqu'ils proposent deux approches différentes pour l'interprétation des contraintes de comptage.
Le comptage global traduit l'intégralité de la contrainte en un seul bloc sans décomposition ni simplification du comptage.
La fonction $Q^{\leftarrow}[\![\cdot]\!]_\chi$ relative à la translation des qualificateurs traduit l'expression e à compter en une formule $\mathcal{L}_{\text{PDL}}$ qui doit adhérer à la forme de comptage. Pour cela deux cas sont envisageables : si la formule obtenue commence par une expression de programme $\langle\pi\rangle$ alors nous écrivons la contrainte de comptage sous la forme $[\![\#^{<\pi>}\varphi \sim c]\!]_\chi$, sinon nous adoptons le programme vide ϵ comme chemin de comptage et la contrainte sera écrite $[\![\#^{<\epsilon>}\varphi \sim c]\!]_\chi$.
Les figures 5.1, 5.2 et 5.3 redéfinissent la traduction des concepts XPath en PDL.

$Q^{\leftarrow}[\![\cdot]\!] : Qualif \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$Q^{\leftarrow}[\![q_1 \ and \ q_2]\!]_\chi =_{def} Q^{\leftarrow}[\![q_1]\!] \wedge Q^{\leftarrow}[\![q_2]\!]_\chi$
$Q^{\leftarrow}[\![q_1 \ or \ q_2]\!]_\chi =_{def} Q^{\leftarrow}[\![q_1]\!] \vee Q^{\leftarrow}[\![q_2]\!]_\chi$
$Q^{\leftarrow}[\![not \ q]\!]_\chi =_{def} \neg Q^{\leftarrow}[\![q]\!]_\chi$
$Q^{\leftarrow}[\![p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_\chi$

$Q^{\leftarrow}[\![count(e) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![count \ (E^{\leftarrow}[\![e]\!]_\chi) \sim c]\!]_\chi$ $\sim \in \{=, <, >, \geq, \leq\}$ et c une constante
$Q^{\leftarrow}[\![count \ (E^{\leftarrow}[\![e]\!]_\chi) \sim c]\!]_\chi =_{def} Q^{\leftarrow}[\![count(\varphi) \sim c]\!]_\chi$ avec φ translation de $E^{\leftarrow}[\![e]\!]_\chi$ en PDL
$Q^{\leftarrow}[\![count(\langle\pi\rangle\varphi) \sim c]\!]_\chi =_{def} [\![\#^{<\pi>}\varphi \sim c]\!]_\chi$
$Q^{\leftarrow}[\![count(\varphi) \sim c]\!]_\chi =_{def} [\![\#^{<\epsilon>}\varphi \sim c]\!]_\chi$

$Q^{\leftarrow}[\![count(e) \ mod \ c]\!]_\chi \ =_{def} \ Q^{\leftarrow}[\![count \ (E^{\leftarrow}[\![e]\!]_\chi) \ mod \ c]\!]_\chi$
$Q^{\leftarrow}[\![count \ (E^{\leftarrow}[\![e]\!]_\chi) \ mod \ c]\!]_\chi \ =_{def} \ Q^{\leftarrow}[\![count(\varphi) \equiv_{(m)} c]\!]_\chi$ avec φ translation de $E^{\leftarrow}[\![e]\!]_\chi$ en PDL
$Q^{\leftarrow}[\![count(\langle\pi\rangle\varphi) \equiv_{(m)} c]\!]_\chi =_{def} [\![\#^{<\pi>}\varphi \equiv_{(m)} c]\!]_\chi$
$Q^{\leftarrow}[\![count(\varphi) \equiv_{(m)} c]\!]_\chi =_{def} [\![\#^{<\epsilon>}\varphi \equiv_{(m)} c]\!]_\chi$

$E^{\leftarrow}[\![\cdot]\!] : \mathcal{L}_{\text{XPath}} \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$E^{\leftarrow}[\![/p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\chi \wedge \langle \overline{2}^* ; \neg\overline{1}\rangle \top)}$
$E^{\leftarrow}[\![p]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_\chi$

$P^{\leftarrow}[\![\cdot]\!] : Path \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$P^{\leftarrow}[\![p^+]\!]_\chi =_{def} (P^{\leftarrow}[\![p]\!]_\top)^+\chi$
$P^{\leftarrow}[\![p^*]\!]_\chi =_{def} (P^{\leftarrow}[\![self]\!]_\top)^*\chi$
$P^{\leftarrow}[\![p_1 \ / \ p_2]\!]_\chi =_{def} P^{\leftarrow}[\![p_1]\!]_{(P^{\leftarrow}[\![p_2]\!]_\chi)}$
$P^{\leftarrow}[\![p/q?]\!]_\chi =_{def} P^{\leftarrow}[\![p/\texttt{ self : :*[q]}]\!]_\chi$
$P^{\leftarrow}[\![p[q]]\!]_\chi =_{def} P^{\leftarrow}[\![p]\!]_{(\chi \wedge Q^{\leftarrow}[\![q]\!]_\top)}$
$P^{\leftarrow}[\![a :: \sigma]\!]_\chi =_{def} A^{\leftarrow}[\![a]\!]_{(\chi \wedge \sigma)}$
$P^{\leftarrow}[\![a :: *]\!]_\chi =_{def} A^{\leftarrow}[\![a]\!]_\chi$

$A^{\leftarrow}[\![\cdot]\!] : Axis \rightarrow \mathcal{L}_{\text{PDL}} \rightarrow \mathcal{L}_{\text{PDL}}$
$A^{\leftarrow}[\![a]\!]_\chi =_{def} \ A^{\leftarrow}[\![symmetric(a)]\!]_\chi$

FIG. 5.4: Translation des qualificateurs

5.3 Traduction des formules PDL en μ-calcul

Dans cette section, nous abordons la traduction en μ-calcul, des formules $\mathcal{L}_{\text{PDL}}$ issues de l'étape précédente. Mais avant d'entamer les détails de cette traduction, nous évoquons d'abord l'étape de détection des cycles de comptage.

5.3.1 Détection des cycles de comptage

D'abord, il faut souligner que cette étape ne concerne que les formules de comptage et non pas les formules génériques puisqu'elle vise les cycles engendrant le dénombrement multiple des noeuds concernés par le comptage.
Donc, avant de procéder à la traduction des formules $\mathcal{L}_{\text{PDL}}$ en μ-calcul, il est nécessaire de tester la présence des cycles au sein des chemins figurant dans la formule de comptage. Cette étape intervient juste à ce niveau-là pour des raisons d'optimisation, en fait il faut détecter les cycles au sein des formules $\mathcal{L}_{\text{PDL}}$ et non pas au niveau des formules μ-calcul afin d'éviter une deuxième traduction inutile au cas où la formule est cyclique.

5.3.2 Dualité de comptage

La propriété de dualité de comptage détaillée dans le chapitre précédent est toujours valable pour l'approche du comptage global. Cette propriété permet de réduire le nombre des règles de traduction en autorisant la déduction des résultats de comptage selon un opérateur donné, à partir des résultats relatifs à l'opérateur de comptage opposé. En fait, les opérateurs de comptage sont opposés deux à deux $\{(<, \geq), (>, \leq), (=, !=), (mod, div)\}$ et il suffit d'appliquer l'opérateur de négation à un comptage selon

un opérateur bien précis pour trouver le résultat du comptage selon l'opérateur opposé, par exemple $[\![\#^{<\pi>}\varphi \leq c]\!]_\chi = \neg[\![\#^{<\pi>}\varphi > c]\!]_\chi$.

5.3.3 Traduction des axes et des formules génériques

La traduction des formules $\mathcal{L}_{\text{PDL}}$ en μ-calcul selon l'approche du comptage global préserve la même interprétation des axes et des formules génériques que l'approche par décomposition de comptage. La différence entre les deux solutions intervient au niveau de l'interprétation des contraintes numériques y compris les contraintes de comptage et les contraintes *modulo* dont les détails seront donnés plus tard. Les tableaux 5.2 et 5.3 redéfinissent la traduction en μ-calcul des axes et des formules génériques.

Axe	**PDL**	**μ-calcul**
χ/child : :*	$\langle \overline{2}^*; \overline{1}\rangle\ \chi$	$\mu Z.\langle\overline{1}\rangle\ \chi \vee \langle\overline{2}\rangle\ Z$
χ/parent : :*	$\langle 1; 2^*\rangle\ \chi$	$\langle 1\rangle\ \mu Z.\ \chi \vee \langle 2\rangle\ Z$
χ/descendant : :*	$\langle(\overline{1} \cap \overline{2})^*; \overline{1}\rangle\ \chi$	$\mu Z.\langle\overline{1}\rangle\ (\chi \vee Z) \vee \langle\overline{2}\rangle\ Z$
χ/descendant-or-self : :*	$\langle\epsilon \cap (\overline{1} \cap \overline{2})^*; \overline{1}\rangle\ \chi$	$\mu Z.\chi \vee \mu Y.\langle\overline{1}\rangle\ (Y \vee Z) \vee \langle\overline{2}\rangle\ Y$
χ/ancestor : :*	$\langle 1; (1 \cap 2)^*\rangle\ \chi$	$\langle 1\rangle\ \mu Z.\ \chi \vee \langle 1\rangle\ Z \vee \langle 2\rangle\ Z$
χ/ancestor-or-self : :*	$\langle(\epsilon \cap 1); (1 \cap 2)^*\rangle\ \chi$	$\mu Z.\chi \vee \langle 1\rangle\ \mu(Y \vee Z) \vee \langle 2\rangle\ Y$
χ/following-sibling : :*	$\langle \overline{2}^*; \overline{2}\rangle\ \chi$	$\mu Z.\langle\overline{2}\rangle\ \chi \vee \langle\overline{2}\rangle\ Z$
χ/preceding-sibling : :*	$\langle 2^*; 2\rangle\ \chi$	$\mu Z.\langle 2\rangle\ \chi \vee \langle 2\rangle\ Z$

Tab. 5.2: Traduction des primitives de navigation

Formule PDL	Interprétation en μ-calcul	μ-calcul selon la sémantique de navigation	μ-calcul selon la sémantique de sélection
$[\![\varphi]\!]_{[\![\pi]\!]_\chi}$	$[\![\langle\pi\rangle\varphi]\!]_\chi$		
$[\![\langle \pi_1\ ;\ \pi_2\rangle\ \varphi]\!]_\chi$	$[\![\langle\pi_1\rangle\langle\pi_2\rangle\varphi]\!]_\chi$		
$[\![\langle \pi_1\ \cup\ \pi_2\rangle\ \varphi]\!]_\chi$	$[\![\langle\pi_1\rangle\varphi]\!]_\chi \vee [\![\langle\pi_2\rangle\varphi]\!]_\chi$		
$[\![\langle \pi_1\ \cap\ \pi_2\rangle\ \varphi]\!]_\chi$	$[\![\langle\pi_1\rangle\varphi]\!]_\chi \wedge [\![\langle\pi_2\rangle\varphi]\!]_\chi$		
$[\![\langle \rho\rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{([\![\rho]\!]_\chi)}$	$\varphi \wedge \langle\rho\rangle\ \chi$	$\chi \wedge \langle\rho\rangle\ \varphi$
$[\![\langle \overline{\rho}\rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{([\![\overline{\rho}]\!]_\chi)}$	$\varphi \wedge \langle\rho\rangle\ \chi$	$\chi \wedge \langle\overline{\rho}\rangle\ \varphi$
$[\![\langle \neg\pi\rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{([\![\neg\pi]\!]_\chi)}$	$\varphi \wedge \langle\neg\overline{\pi}\rangle\ \chi$	$\chi \wedge \langle\neg\pi\rangle\ \varphi$
$[\![\langle \pi^+\rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{(\mu Z.[\![\pi]\!]_{(Z\vee\chi)})}$	$\varphi \wedge \mu Z.\chi \vee \langle\overline{\pi}\rangle Z$	$\chi \wedge \mu Z.\varphi \vee \langle\pi\rangle Z$
$[\![\langle \pi^*\rangle\ \varphi]\!]_\chi$	$[\![\varphi]\!]_{(\chi\vee\mu Z.[\![\pi]\!]_{(Z\vee\chi)})}$	$\chi \vee (\varphi \wedge \mu Z.\chi \vee \langle\overline{\pi}\rangle Z)$	$\chi \wedge (\varphi \vee \mu Z.\varphi \vee \langle\pi\rangle Z)$
$[\![\langle \varphi?\rangle\ \varphi'\rangle]\!]_\chi$	$[\![\varphi']\!]_{(\varphi\wedge\chi)}$	$\varphi \wedge \varphi' \wedge \chi$	$\chi \wedge \varphi \wedge \varphi'$
$[\![\langle \rho/\varphi?\rangle\ \varphi'\rangle]\!]_\chi$	$[\![\varphi']\!]_{([\![\varphi]\!]_{([\![\rho]\!]_\chi)})}$	$\varphi \wedge \varphi' \wedge \langle\overline{\rho}\rangle\chi$	$\chi \wedge \langle\rho\rangle(\varphi \wedge \varphi')$

Tab. 5.3: Traduction des formules génériques

5.3.4 Traduction des contraintes de comptage

Le comptage global se base sur un principe de comptage générique qui s'applique à l'ensemble des chemins et des opérateurs de comptage, néanmoins il présente quelques cas particuliers relatifs aux valeurs 0 et 1 de la constante de comptage. Dans ce qui suit, nous définissons le principe de comptage, nous énumérons par la suite les cas particuliers relatifs aux différents opérateurs de comptage.

5.3.4.1 Principe de comptage

Le principe de comptage global que nous proposons satisfait les exigences de monotonie et du comptage ordonné. L'ordre de comptage est assuré par la relation de précédence entre les différents noeuds cibles de comptage, tandis que la monotonie du chemin de comptage est déjà confirmée par l'environnement de détection des cycles appliqué à la formule concernée par le comptage.

Le principe proposé consiste à référencer le noeud contexte, ensuite à parcourir l'arbre XML selon le chemin de comptage et trouver les noeuds satisfaisant φ tout en assurant l'ordre de précédence entre ces noeuds et en décrémentant le comptage à chaque itération.

$$[\![\#^{<\pi>}\varphi \sim c]\!]_\chi =_{def} \chi \ \wedge \ \langle\pi\rangle\big(\varphi \wedge [\![\#^{<path_{\ll}>}\varphi \sim c-1]\!]_\top\big)$$

$$\text{avec } path_{\ll} = (1;2^*)^+ \cup ((\overline{2}^*;\overline{1})^*;2^+;(1;2^*)^*)$$

Cette formule est itérative et décrémentale dans le sens où elle se répète récursivement autant de fois que la constante de comptage, ainsi qu'elle décrémente le comptage à chaque itération. L'application de cette formule se déroule selon les itérations décrites ci-dessous.

itération 1 $[\![\#^{<\pi>}\varphi \sim c]\!]_\chi = \chi \ \wedge \ \langle\pi\rangle\ (\varphi \wedge [\![\#^{<path_{\ll}>}\varphi \sim c-1]\!]_\top)$
itération 2 $[\![\#^{<\pi>}\varphi \sim c]\!]_\chi = \chi \ \wedge \ \langle\pi\rangle\big(\varphi \ \wedge \ \langle path_{\ll}\rangle\ ([\![\#^{<path_{\ll}>}\varphi \sim c-2]\!]_\top)\big)$
itération i $[\![\#^{<\pi>}\varphi \sim c]\!]_\chi = \chi \ \wedge \ \langle\pi\rangle\big(\varphi \ \wedge \ \langle path_{\ll}\rangle\ (\varphi \ \wedge \ \langle path_{\ll}\rangle(...))\big)$

5.3.4.2 Cas particuliers

Nous donnons ci-dessous les cas particuliers relatifs au comptage selon les opérateurs $(>, \geq, =)$, les autres cas sont à déduire en se basant sur la propriété de dualité. L'énumération de ces particularités de comptage a pour objectif d'éviter l'application du principe de comptage pour des cas simples.

- Comptage selon l'opérateur $>$
  Si $c = 0$ $\quad [\![\#^{<\pi>}\varphi > c]\!]_\chi = [\![\langle\pi\rangle\varphi]\!]_\chi$
- Comptage selon l'opérateur $\geq$
  Si $c = 0$ $\quad [\![\#^{<\pi>}\varphi \geq c]\!]_\chi = \top$
  Si $c = 1$ $\quad [\![\#^{<\pi>}\varphi \geq c]\!]_\chi = [\![\langle\pi\rangle\varphi]\!]_\chi$
- Comptage selon l'opérateur $=$
  Si $c = 0$ $\quad [\![\#^{<\pi>}\varphi = c]\!]_\chi = [\![\langle\pi\rangle\neg\varphi]\!]_\chi$
  Si $c = 1$ $\quad [\![\#^{<\pi>}\varphi = c]\!]_\chi = [\![\langle\pi\rangle\varphi]\!]_\chi$

5.3.5 Traduction des contraintes modulo

La traduction des contraintes modulo vers le μ-calcul est similaire à celle des contraintes de comptage selon les opérateurs $(<, >, \geq, \leq, =, !=)$ dans le sens où elle se base sur un principe générique qui s'applique à la totalité des chemins. Ce principe adhère à la sémantique de l'opérateur *modulo* en proposant comme traduction, une évaluation récursive du fait que le nombre de noeuds accessibles à partir du chemin de comptage et satisfaisant φ est différent de la constante de comptage c. La formule suivante détaille le principe de traduction des contraintes modulo.

$$[\![\#^{<\pi>}\varphi \equiv_{(m)} c]\!]_\chi = [\![\#^{<\pi>}\varphi \ != \ c]\!]_{[\![\#^{<\pi>}\varphi \ != c]\!]} \ddots {}_{[\![\#^{<\pi>}\varphi \ != c]\!]_\chi}$$

Il est à souligner que la récursivité de ce principe admet comme condition d'arrêt la fin du chemin de comptage.
Le comptage selon l'opérateur *modulo* présente un seul cas particulier relatif à la valeur 1 de la constante de comptage, en fait, $[\![\#^{<\pi>}\varphi \equiv_{(m)} c]\!]_\chi = \bot$.
Il reste à noter que la traduction des contraintes de comptage selon l'opérateur *div* est à déduire en se basant sur la propriété de dualité appliquée à l'opérateur *modulo*.

5.4 Synthèse des approches proposées

Les approches proposées représentent deux résultats importants pour résoudre le problème de décidabilité des requêtes XPath avec contraintes de comptage. Dans ce qui suit, nous retraçons l'ensemble des idées liés à ces solutions, d'abord nous donnons un récapitulatif des deux approches, ensuite nous procédons à une comparaison entre ces solutions en se basant sur des illustrations par l'exemple. Pour conclure, nous positionnons les approches présentées par rapport à notre problématique.

5.4.1 Récapitulatif des deux approches

Le tableau 5.4 donne le résumé des méthodes proposées en présentant leurs approches pour satisfaire les exigences de comptage et leurs démarches de traduction des formules de comptage comprenant la translation des concepts XPath vers PDL et la traduction des formules $\mathcal{L}_{\text{PDL}}$ vers le μ-calcul.

TAB. 5.4: Récapitulatif des deux approches

Comptage par décomposition de chemin	**Comptage global**
Exigences de comptage	
Le comptage selon des programmes primitifs assure la monotonie du chemin de comptage. L'ordre de comptage est assuré par la monotonie du chemin.	Monotonie assurée par un environnement de détection de cycles. Le comptage est ordonné selon l'ordre du document (ordre d'apparition des noeuds dans la structure XML).
Translation des concept XPath en PDL	
Translation des expressions des chemins et des axes selon la sémantique de navigation. Translation des qualificateurs selon la sémantique de sélection. Écriture des formules sous la troisième forme normale (positionner l'opérateur de négation sur les propositions atomiques). Décomposition du comptage selon les propositions atomiques.	Translation des expressions des chemins et des axes selon la sémantique de navigation. Translation des qualificateurs selon la sémantique de sélection. Comptage selon une seule formule $\mathcal{L}_{\text{PDL}}$ complexe. Détection des cycles dans la formule $\mathcal{L}_{\text{PDL}}$ obtenue en l'appliquant à l'environnement de détection de cycles.
Traduction des formules $\mathcal{L}_{\text{PDL}}$ en μ-calcul	
Traduction relative à la typologie du chemin de comptage. Décomposition du comptage selon les différents programmes figurant dans la formule $\mathcal{L}_{\text{PDL}}$.	Principe de comptage générique. Comptage itératif selon une seule formule composite.

5.4.2 Comparaison

La comparaison entre les deux approches proposées se fait sur le plan théorique ainsi que pratique. Théoriquement, les deux solutions diffèrent dans la spécification d'une méthodologie de comptage. Leurs différences portent sur la définition du principe de comptage, la satisfaction des exigences de comptage (monotonie du chemin et comptage ordonné) et la nature des formules de comptage visées. Le tableau 5.5 compare les deux approches de comptage selon ces critères.

TAB. 5.5: Comparaison des deux approches

	Décomposition de comptage	Comptage global
Principe de comptage	Principe relatif à la typologie du chemin et à l'opérateur de comptage.	Principe générique couvrant l'ensemble des chemins et des opérateurs.
Monotonie du chemin	Monotonie assurée par le comptage selon des programmes primitifs.	Monotonie assurée par un environnement de détection des cycles de comptage.
Ordre de comptage	Ordre relatif au chemin de comptage.	Ordre du document (ordre d'apparition des noeuds dans la structure XML).
Type des contraintes de comptage	Approche plutôt adaptée aux contraintes de comptage simples.	Approche plutôt efficace pour les contraintes de comptage complexes.

Sur le plan pratique, les deux solutions sont approximativement similaires au niveau de l'implémentation puisqu'elles comprennent les mêmes étapes et les mêmes directives à quelques différences près. La constatation la plus importante à cet égard, est que l'approche par décomposition de comptage autorise l'exécution parallèle du comptage puisqu'elle divise la formule à compter en plusieurs formules élémentaires et indépendantes permettant ainsi d'évaluer parallèlement l'ensemble de ces formules et d'augmenter les performances du comptage, tandis que les formules de comptage issues de l'approche par comptage global sont caractérisées par leur récursivité, ce qui rend leur parallélisation un peu délicate.

5.4.2.1 Illustration par l'exemple

Pour mettre l'accent sur la différence entre les deux approches, nous considérons deux contraintes de comptage ayant deux chemins différents dont l'un est simple et l'autre est composite.
Commençons par la formule de comptage simple $[\![\#^{<2^+>}b > 1]\!]_\chi$ dont la traduction en Dynamic μ-calculus selon l'approche par décomposition du comptage donne le résultat suivant :

$$\begin{array}{ll} \chi \wedge \langle 2^+\rangle(b \wedge \langle 2^+\rangle b) & (1) \\ \Downarrow & \\ \chi \wedge \mu X.(b \wedge (\mu Y.b \vee \langle 2\rangle Y)) \vee \langle 2\rangle X & (2) \end{array}$$

Nous traduisons maintenant la même formule selon l'approche du comptage global. Pour permettre la lisibilité des formules de traduction, nous donnons directement le résultat sans fournir de détails. La

$$\begin{array}{ll} \chi \wedge \langle 2^+\rangle(b \wedge \langle (1;2^*)^+ \cup ((\overline{2}^*;\overline{1})^*;2^+;(1;2^*)^*)\rangle b) & (1) \\ \Downarrow & \\ \chi \wedge \mu X.\Big(b \wedge \big((\mu Y.b \vee \langle 1\rangle(Y \vee \mu Z.Y \vee \langle 2\rangle Z))\vee & (2) \\ ((\mu T.(b \vee \mu H.b \vee (\langle 1\rangle(H \vee \mu F.H \vee \langle 2\rangle F))) \vee \langle 2\rangle T)\vee & \\ \mu K.(\mu T.(b \vee \mu H.b \vee (\langle 1\rangle(H \vee \mu F.H \vee \langle 2\rangle F))) \vee \langle 2\rangle T)\vee & \\ (\langle \overline{1}\rangle K \vee \mu M.\langle \overline{1}\rangle K \vee \langle \overline{2}\rangle M))\big)\Big) \vee \langle 2\rangle X & \end{array}$$

différence entre les résultats des deux traductions est évidente. En fait, la complexité accrue de la translation selon l'approche du comptage global est due à l'ordre de comptage (l'ordre du document dans ce cas) qui impose une relation de précédence couvrant toute la structure XML, tandis que dans notre contexte, nous avons besoin seulement d'un ordre entre les deux noeuds satisfaisant b.
Passons maintenant au deuxième exemple qui consiste à traduire la contrainte de comptage suivante :

$$[\![\#^{<(1^+;2^+)\cup(\overline{2}^+;\overline{1}^+)>}b > 4]\!]_\chi$$

La complexité de cette formule fait que sa traduction soit volumineuse ; pour cela nous nous contentons de donner les premières pistes de traduction et de fournir un aperçu des résultats afin d'illustrer l'efficacité des deux approches à l'égard des contraintes de comptage compliquées.
La traduction préliminaire de la contrainte en question selon l'approche du comptage global donne la formule ci-dessous :

$$\chi \wedge \langle (1^+;2^+) \cup (\overline{2}^+;\overline{1}^+)\rangle\Big(b \wedge \langle path_{<<}\rangle\big(b \wedge \langle path_{<<}\rangle(b \wedge \langle path_{<<}\rangle(b \wedge \langle path_{<<}\rangle(b)))\big)\Big)$$
$$\text{avec } path_{<<} = (1;2^*)^+ \cup ((\overline{2}^*;\overline{1})^*;2^+;(1;2^*)^*)$$

Tandis que l'approche par décomposition de comptage propose une traduction qui engendre un nombre important de formules de comptage simple selon les programmes $\langle 1^+\rangle, \langle 2^+\rangle, \langle \overline{2}^+\rangle$ et $\langle \overline{1}^+\rangle$. Cette traduction a pour principe d'énumérer les différents cas possibles pour le comptage des noeuds satisfaisant b selon les chemins cités.
Dans notre cas, nous dénombrons 68 formules de comptage élémentaire ; la traduction proposée est donc une jonction entre ces différentes formules, ce qui fait qu'il nous est difficile de donner les détails de

cette traduction.
Ce qu'il faut retenir de ce deuxième exemple, c'est que l'approche du comptage global est plus efficace pour la translation des contraintes de comptage à large portée, ce qui n'est pas le cas dans le premier exemple. La synthèse des deux exemples confirme les détails de la comparaison donnée ci-dessus.

5.4.2.2 Conclusion

La complexité du comptage dépend essentiellement de trois paramètres :
- La complexité du chemin de comptage.
- La complexité de la formule propositionnelle à compter.
- Le nombre d'occurrences de comptage (la constante de comptage c).

La figure 5.5 décrit la position des deux approches proposées par rapport au problème de comptage XPath. En fait, l'approche par décomposition de comptage s'avère plus adaptée aux formules de comptage ayant des chemins primitifs, des formules propositionnelles simples et des constantes de comptage assez réduites, tandis que l'approche du comptage global est plus performante lorsqu'il s'agit de formules à large portée ayant des chemins complexes, des formules composites et des constantes de comptage à grande échelle.

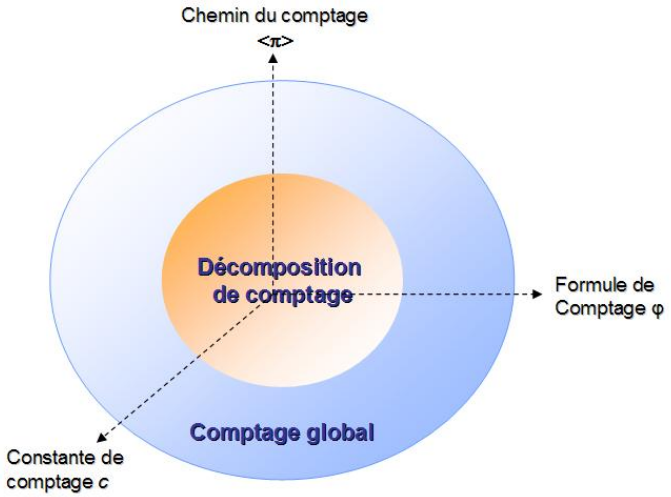


FIG. 5.5: Position des deux approches par rapport à la complexité du comptage

Chapitre 6

Algorithme de décision et implémentation

Nous présentons dans ce chapitre notre approche pour la décidabilité du fragment XPath avec contraintes de comptage. Il s'agit d'une procédure de décision qui se base sur l'algorithme établi par les travaux de Genevès[10, 11] pour la résolution efficace des problèmes d'analyse statique pour XML en s'appuyant sur la logique μ-calcul.
Notre approche consiste à adapter cet algorithme comme noyau pour notre procédure de décision en assurant au préalable la traduction des concepts XPath en PDL puis en μ-calcul. La suite de ce chapitre se décline en deux parties, d'abord nous détaillons les différentes phases de la procédure de décision, nous abordons par la suite l'étape d'implémentation.

6.1 Procédure de décision

La procédure de décision que nous proposons pour le fragment XPath avec contraintes de comptage, comprend un noyau algorithmique pour la décidabilité des formules μ-calcul, ainsi qu'une couche de traduction assurant au préalable la translation des concepts XPath en PDL puis en μ-calcul.

6.1.1 Translation des concepts XPath

La première phase de notre procédure de décision consiste à traduire les concepts XPath en PDL puis en μ-calcul. Cette traduction se fait selon les deux approches de traduction par décomposition de comptage et par comptage global détaillées dans les chapitres 5 et 6. C'est une phase préparatrice pour l'étape suivante qui porte sur l'application de l'algorithme de décision sur la formule μ-calcul obtenue afin de déterminer sa satisfaisabilité.

6.1.2 Algorithme de décision

Une fois la formalisation du problème XPath en μ-calcul faite, il reste à décider de la satisfaisabilité de la formule en question en se basant sur l'algorithme de décision établi par les travaux de Genevès [10, 12].
Cet algorithme présente deux propriétés importantes : D'abord il est établi pour la logique μ-calcul modal sans alternance dont la richesse expressive est égale à l'expressivité de la logique du second ordre (WS2S), en plus il admet une complexité exponentielle simple et il est efficace sur le plan pratique.
L'idée de cet algorithme consiste à déterminer un arbre binaire fini comme modèle pour la formule à satisfaire en énumérant et construisant les différents arbres possibles tout en vérifiant l'existence d'un noeud qui satisfait la formule en question.

6.1.2.1 Définitions préliminaires

Avant d'entamer les détails de l'algorithme de décision, nous introduisons quelques définitions préliminaires liées aux sous-formules composant la formule ψ à résoudre. En fait le statut de vérité de ψ peut être déterminé par le statut de quelques-unes de ses sous-formules.
Pour cela nous commençons par définir la Fisher-Ladner closure $cl(\psi)$ qui représente l'ensemble des sous formules de ψ où les points fixes sont déroulés une fois, ensuite nous nous intéressons particulièrement à un sous-ensemble $Lean(\psi) \subseteq cl(\psi)$ qui comprend essentiellement les modalités d'accès aux noeuds et les propositions qu'ils satisfont :

$$Lean(\psi) = \{\langle a\rangle\top \mid a \in \{1, 2, \overline{1}, \overline{2}\}\} \cup \sum(\psi) \cup \{\langle a\rangle\varphi \mid \langle a\rangle\varphi \in cl(\psi)\}$$

À partir de la spécification du $Lean(\psi)$ nous introduisons la définition des ψ-types $t \subseteq Lean(\psi)$ comme des sous-ensembles représentant les entités de base pour la construction des arbres binaires finis, objet de notre algorithme.

6.1.2.2 Algorithme

L'algorithme que nous adoptons pour la décidabilité des requêtes XPath avec contraintes de comptage a pour principe de construire un arbre binaire fini de ψ-types en se basant sur la méthode des tableaux qui est une méthode analytique dont les règles de calcul s'appuient sur la décomposition syntaxique de la formule à satisfaire.
La construction de l'arbre binaire s'appuie sur le calcul d'un plus petit point fixe et se fait selon l'approche bottom-up (de bas en haut) : à partir des feuilles figurant dans l'ensemble $Lean(\psi)$, l'algorithme construit itérativement les noeuds du tableau (les ψ-types) en ajoutant tous les noeuds parents possibles des noeuds courants, l'algorithme s'arrête lorsqu'un point fixe est atteint. À la fin, le point fixe contient toutes les racines de tous les arbres établis. Si la formule ψ est présente dans une racine, alors elle est satisfaisable et l'arbre de la racine en question peut être facilement extrait et employé comme un modèle

satisfaisant la requête XPath.
La figure ci-dessous donne un aperçu de l'algorithme de décision ; la fonction *update* permet d'incrémenter la construction de l'arbre binaire en y insérant les noeuds parents possibles, tandis que la fonction *present* permet de vérifier la présence de la formule à satisfaire dans les arbres construits.

T $\longleftarrow \emptyset$
repeat
T' $\longleftarrow$ ***T***
T $\longleftarrow$ update(T')
if present(ψ,T) ***then***
return "ψ est satisfaisable"
until T=T'
return "ψ est insatisfaisable"

Pour avoir plus de détails concernant le principe de cet algorithme ainsi que ses caractéristiques (complétude, exactitude, complexité..) il faut se référer aux travaux de Genevès[10].

6.2 Implémentation et test

La procédure de décision proposée a été mise en application. Un compilateur prend des expressions de XPath comme entrée, et les traduit en formules PDL puis en formules μ-calcul selon l'approche par décomposition de comptage. La formule relative à une requête XPath particulière est donc composée, traduite puis résolue via le décideur de satisfaisabilité déjà mis en place par les travaux de Genevès[10, 9]portant sur l'analyse statique pour XML. Ce décideur de satisfaisabilité implémente l'algorithme de décision détaillé dans la première section.
Dans ce qui suit, nous nous concentrons sur le compilateur XPath et nous décrivons les techniques utilisées pour son implémentation.

6.2.1 Vue d'ensemble

Nous présentons une vue d'ensemble des idées et des techniques qui ont guidé la réalisation de notre compilateur XPath. Il s'agit de choix conceptuels, de la méthodologie de développement et des solutions techniques adoptées.

6.2.1.1 Méthodologie de développement

Pour réaliser notre compilateur XPath, nous avons opté pour une approche basée sur le processus unifié ***eXtreme Programming*** guidé par la modélisation objet en ***UML***.

eXtreme Programming C'est une approche itérative et incrémentale permettant de développer le compilateur selon des incréments élémentaires dont chacun remplit un sous-ensemble des exigences et des fonctionnalités requises. À la fin de chaque incrément, nous testons la validité des tâches réalisées afin de procéder à d'éventuelles corrections et rectifications.
Le choix de cette méthodologie de développement est justifié par le fait qu'elle soit adaptée aux équipes réduites avec des besoins changeants, ce qui est le cas dans ce travail.

Modélisation UML La modélisation UML porte essentiellement sur l'élaboration de diagrammes de classe permettant de piloter la réalisation du compilateur durant les différentes phases du processus de développement. Pour chaque unité fonctionnelle cohérente nous créons un diagramme de classe décrivant la structure à réaliser.

Programmation Java Le codage de notre compilateur XPath se fait en Java. Dans ce paragraphe, nous sommes loin de faire une comparaison entre les langages de programmation, ce choix est exigé par l'état de l'art de notre sujet. En fait nous avons eu recours à ce langage de programmation pour adhérer aux solutions existantes puisque les parsers implémentant la spécification XPath 2.0 ainsi que l'algorithme de satisfaisabilité sur lequel se base notre travail sont codés en Java.

6.2.1.2 Saxon vs. PsychoPath ?

La réalisation de notre compilateur passe inévitablement par l'analyse syntaxique des requêtes XPath, pour cela nous devons intégrer dans notre implémentation un parseur XPath 2.0. Les études que nous avons effectué autour des solutions existantes ont aboutit à deux choix : SAXON et PsychoPath.

SAXON C'est la première implémentation Java de XPath 2.0 supportant les schémas XML[19]. Ce parseur est réalisé par Mike Kay qui est un rédacteur de la spécification XPath 2.0 et reste le mieux placé pour fournir un processeur XPath performant.
SAXON existe en deux versions ; une version commerciale qui fournit un support complet de la norme XPath 2.0, et une version open source qui implémente partiellement cette spécification et qui ne supporte pas les schémas XML.

PsychoPath C'est la première version open source d'un processeur XPath 2.0 en Java qui supporte les schémas XML (ce que seule la version payante de SAXON sait faire). L'avantage majeur de ce parseur est qu'il est conçu d'une manière flexible et modulable permettant son extension par d'autres fonctionnalités comme la traduction des expressions XPath en PDL dans notre cas.

La comparaison empirique entre SAXON et PsychoPath se fait essentiellement selon les critères de performance et de support des schémas XML. L'évaluation des deux parseurs montre que SAXON est fortement optimisé et qu'il est plus rapide que PsychoPath, en plus il est compatible avec la version XPath 1.0. Néanmoins nous devons comparer la version open source de SAXON avec PsychoPath, et dans ce cas ce dernier est plus avantageux puisqu'il supporte les schémas XML. En résumé, nous avons opté pour PsychoPath comme processeur XPath 2.0 supportant notre compilateur.

6.2.1.3 Patron conceptuel *Visitor*[8]

Contexte et problématique Les requêtes XPath forment évidemment des arbres dont les noeuds sont les expressions de la requête et les arcs sont les références entre ces expressions. Ces arbres sont par ailleurs étiquetés : tous les arcs ont une étiquette qui est l'identifiant utilisé dans la source pour se référer à la cible ; et typés : chaque expression possède un type pouvant être attribué à plusieurs expressions simultanément.
Le problème du parcours de ces arbres XPath est omniprésent dans la réalisation de notre compilateur : étant donné un arbre d'expressions XPath de différents types, comment parcourir cet arbre et générer une traduction à chaque noeud qui dépend de sa typologie, et ce d'une manière qui soit efficace, simple et ouverte ?
Un problème supplémentaire émerge dans ce contexte qui est la prise en compte des modifications et des extensions. Ce problème apparaît explicitement lorsqu'on souhaite insérer de nouveaux types d'expressions qui agissent sur l'ensemble des types déjà existants et qui sont structurellement reliés entre eux, trois réponses possibles à cet égard : transformation de la structure elle-même, transformation d'une partie de la structure qui est liée aux nouveaux types ou transformation vers une autre structure ?

Solution Nous proposons une solution inspirée des recommandations W3C pour XPath 2.0[18] qui s'appuie sur l'utilisation du patron conceptuel «Visitor». Ce patron constitue la solution la plus classique et la plus utilisée au problème général du parcours typé de graphes d'objets, il s'appuie sur la construction

d'itérateurs spécifiques à chaque type d'objet parcouru permettant ainsi de personnaliser simplement les actions et les stratégies de parcours d'un graphe d'objet. La structure de ce patron se résume en deux parties : celle des classes visitées qui sont dans notre contexte les types d'expressions XPath, et celle des classes visiteurs qui ont pour rôle de rajouter des fonctionnalités supplémentaires (non prévues au préalable) aux classes visitées. Dans notre cas, il s'agit de fournir une traduction PDL pour chaque type d'expression visitée. Nous revenons sur les détails d'implémentation du patron Visitor dans la section suivante.
La méthodologie de programmation par le patron conceptuel «Visitor» est proposée comme outil de base pour la réalisation du processeur PsychoPath. Notre tâche se limite donc à étendre la structure existante en créant notre propre visiteur qui permet de fournir la formule équivalente en PDL pour chaque type d'expression visité.
Cette solution a pourtant un certain nombre de défauts bien connus : elle produit un code fortement couplé et peu évolutif ce qui engendre des difficultés lors de l'adjonction de nouveaux types d'expressions XPath qui ne figurent pas dans la spécification 2.0 ni dans le parseur PsychoPath tels que les chemins conditionnés, les chemins itératifs et la clôture réflexive sur les chemins. Ce problème peut être résolu en procédant à la modification de la structure implémentant le patron Visitor, réalisée par une composition sans relation entre les nouveaux types et les types déjà existants.

6.2.2 Détails de l'implémentation

Cette section est organisée comme suit. Dans le premier paragraphe, nous présentons la structure du compilateur en termes de paquetages réalisés constituant chacun une unité fonctionnelle cohérente. Cette structure étend celle du processeur PsychoPath. Le deuxième paragraphe est consacré à la présentation des fonctionnalités implémentées, nous détaillons les différentes étapes de satisfaisabilité d'une requête XPath.

6.2.2.1 Paquetages

Pour réaliser notre compilateur XPath dans les règles d'art, nous avons fixé comme objectif de le concevoir d'une façon modulable et flexible assurant un couplage faible entre les différents paquetages, ce qui permet d'une part de satisfaire l'exigence d'extensibilité, et d'autre part de préciser la responsabilité de chaque paquetage et de l'évaluer indépendamment des autres.
Notre implémentation comprend sept paquetages préfixés '***fr.inrialpes.wam.xpath***', la figure 6.1 donne une vue d'ensemble de ces paquetages en mettant l'accent sur la nature des relations qui les joignent. Nous donnons par la suite la description de chaque paquetage et de son contenu en termes de classes Java en précisant le rôle de chaque classe.

Le paquetage *ast* L'analyse d'une expression XPath produit un arbre dont les noeuds reflètent les différents types d'expressions XPath qui existent. Ceci mène à la définition du paquetage 'ast' qui introduit la notion de syntaxe abstraite dénommée AST[20] (Abstract Syntax Tree) représentant les diverses expressions de XPath. Ce paquetage est déjà inclut dans le parseur PsychoPath, et il contient l'ensemble des types spécifiés dans la norme XPath 2.0.
Néanmoins il reste à définir les types d'expressions qui font partie du fragment XPath considéré dans notre travail de recherche et qui ne figurent dans la spécification 2.0 tels que les chemins conditionnés, les chemins itératifs et la clôture réflexive sur les chemins.
Pour cela nous avons définis notre propre paquetage 'ast' dans lequel nous avons crée trois classes faisant partie de la structure parcourue par le visiteur et implémentant la méthode 'accept' du pattern Visitor :

- XPathClosure : correspond aux expressions de clôture réflexive sur les chemins.
- XPathIteration : représente les expressions d'itération sur les chemins.
- ConditionalXPath : définit les expressions des chemins conditionnés.

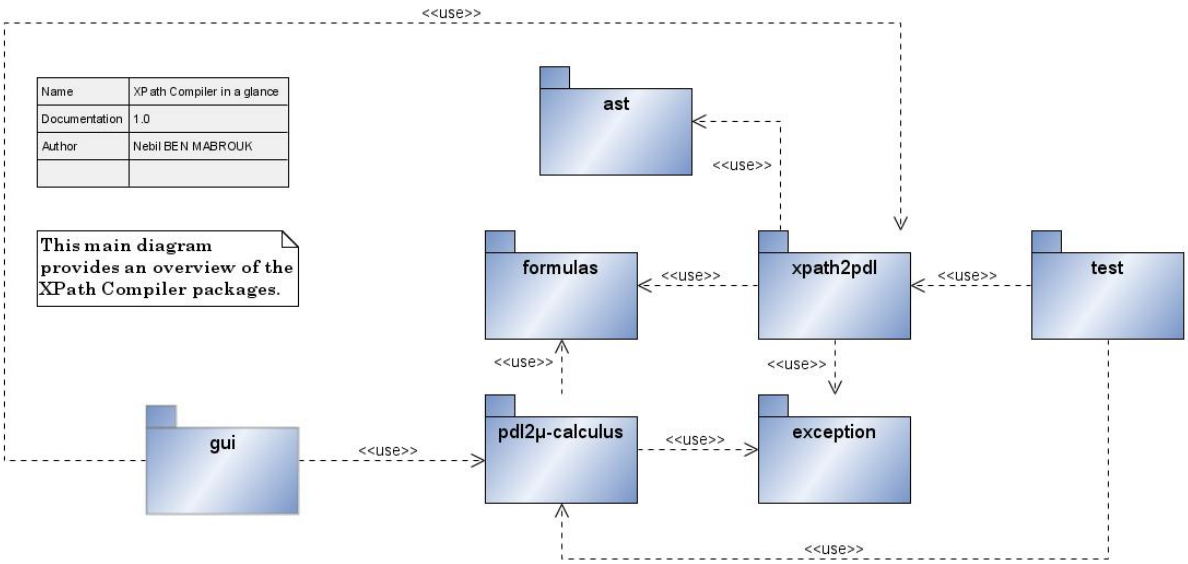


FIG. 6.1: Vue d'ensemble du compilateur XPath

La figure 6.2 montre la manière avec laquelle ces classes sont codés à travers l'exemple de la classe XPathClosure.

```
package fr.inrialpes.wam.xpath.ast;

import org.ucl.xpath.ast.XPathNode;

import fr.inrialpes.wam.xpath.pdl.MainVisitor;

public class XPathClosure extends XPathNode{

    /**
     * Support for Visitor interface.
     * @return Result of Visitor operation.
     */

    public Object accept(MainVisitor v) {
        return v.visit(this);
    }
}
```

FIG. 6.2: Code Java de la classe XPathClosure

Le paquetage *formulas* Ce paquetage fournit les classes nécessaires à la création des formules PDL et μ-calcul relatives aux requêtes XPath, il contient une interface 'LogicalFormula' représentant les formules génériques des logiques modales. Cette interface est implémentée par deux classes 'PDLFormula' et 'μ-calculusFormula' contenant, outre la définition des opérateurs et des symboles logiques de ces deux formalismes, les méthodes nécessaires pour la création des différents types de formules selon ces deux logiques. La figure 6.3 récapitule le contenu de ce paquetage.

Le paquetage *xpath2pdl* C'est l'un des paquetage principaux de notre compilateur XPath, il réalise la première étape de la procédure de décision portant sur la traduction des concepts XPath en PDL. La figure 6.4 donne un aperçu des classes contenu dans ce paquetage, une grande partie des détails est cachée pour la clarté du diagramme.
La classe principale de ce paquetage 'XPath2PDL'assure trois fonctions primordiales :

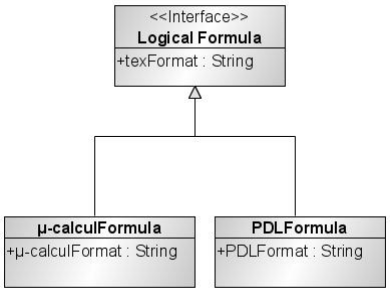


FIG. 6.3: Le paquetage “formulas”

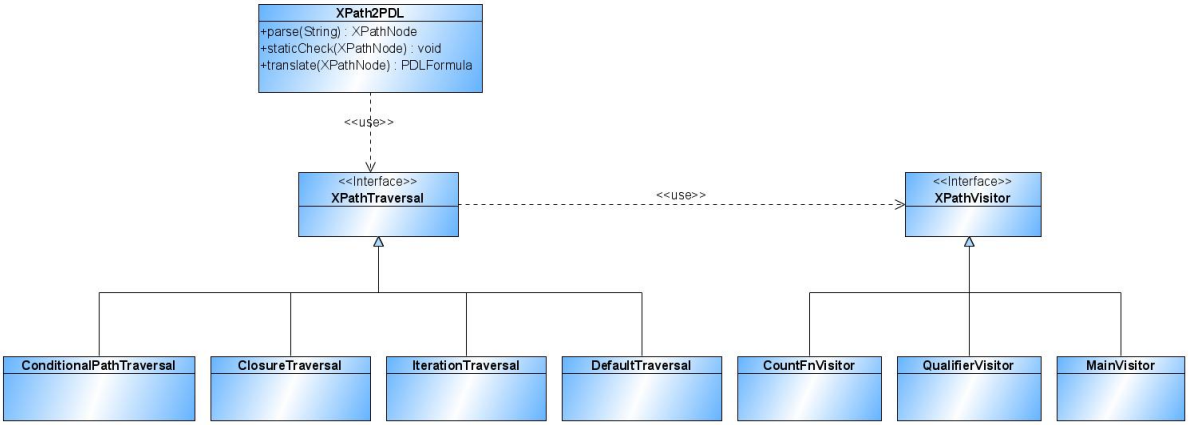


FIG. 6.4: Le paquetage “xpath2pdl”

✓ ‘Parsing[16]’ des requêtes XPath

L’expression de XPath 2.0 doit être ‘parsée’et représentée comme un arbre de syntaxe abstraite (AST). Cette étape fait appel au parseur PsychoPath comme l’indique la figure 6.5.

```
public XPathNode parseXPath(String query)throws ParsingException{

    XPathParser _parser = new JFlexCupParser();
    XPathNode _xp = null;

    long startParsingTime = System.currentTimeMillis();

    _xp =  _parser.parse(query);

    long stopParsingTime = System.currentTimeMillis();
    this._parsingtime = stopParsingTime - startParsingTime;

    return _xp;
}
```

FIG. 6.5: ‘Parsing’ des requêtes XPath

✓ Analyse statique des requêtes XPath

L’expression XPath 2.0 doit être statiquement analysée pour vérifier sa validité structurale. Cette étape est assurée par PsychoPath qui définit une classe implémentant le modèle de visiteur pour traverser et vérifier l’AST résultant de l’étape précédente. L’invocation de cette classe est montré par la figure 6.6.

✓ Translation des requêtes XPath en PDL

Cette fonction représente l’objectif principal du paquetage ‘xpath2pdl’, elle assure la traduction des

```
public void staticChecking(XPathNode _xp)throws StaticCheckException{

    StaticContext _sc = new DefaultStaticContext();
    StaticChecker name_check = new StaticNameResolver(_sc);

    long startCheckingTime = System.currentTimeMillis();

    name_check.check(_xp);

    long stopCheckingTime = System.currentTimeMillis();
    this._checkingtime = stopCheckingTime - startCheckingTime;
}
```

FIG. 6.6: Analyse statique des concepts XPath

concepts XPath en PDL en faisant appel aux classes 'Traversal'invoquant à leur tour les classes visiteur permettant de parcourir l'arbre AST et de fournir une translation PDL pour chaque type de noeud. Ceci est montré par la figure 6.7.

```
public PDLFormula translate(XPath _xp)throws PDLTranslationException{

    DefaultTraversal _traversal = new DefaultTraversal();
    PDLFormula _formula = new PDLFormula();

    long startTranslationTime = System.currentTimeMillis();

    _formula = _traversal.traverse(_xp);

    long stopTranslationTime = System.currentTimeMillis();
    this._translationtime = stopTranslationTime-startTranslationTime;

}
```

FIG. 6.7: Translation des concepts XPath en PDL

La suite des classes composant le paquetage 'xpath2pdl' se décline en deux catégories, à savoir les classes 'Traversal' et les classes 'Visitor' :

- Les classes 'Traversal' : Ce sont les classes implémentant l'interface 'XPathTraversal' et qui ont pour rôle de sélectionner une stratégie de parcours selon la typologie de l'expression XPath considérée. S'il s'agit d'une expression XPath 2.0, nous faisons appel à la classe 'DefaultTraversal' pour parcourir cette expression, sinon nous invoquons les classes 'ClosureTraversal', 'IterationTraversal' ou 'ConditionalXPathTraversal' au cas ou l'expression est respectivement une clôture réflexive sur un chemin, une itération de chemin ou un chemin conditionné.

- Les classes 'Visitor' : Ce sont des classes réalisant le modèle visiteur et implémentant l'interface 'XPathVisitor'. Elles sont invoquées par les classes 'Traversal' pour visiter les différents types de noeuds composant l'expression XPath en considération. Si l'expression visitée est une expression de chemin nous faisons appel à la classe 'MainVisitor' qui engendre une interprétation PDL de l'expression en question selon la sémantique de navigation, sinon si le noeud visité est un qualificateur nous la traduisons en PDL selon la sémantique de sélection. Un visiteur additionnel 'CountFnVisitor' a été crée pour prendre en charge les traitements spécifiques à la traduction des contraintes de comptage.
  L'ensemble des visiteurs utilise la classe 'PDLFormula' pour générer les formules PDL relatives aux requêtes XPath considérées.

Le paquetage *pdl2μ-calculus* Ce paquetage réalise la deuxième phase de notre procédure de décision qui porte sur la translation en μ-calcul des formules PDL résultantes de la phase précédente. Il comprend essentiellement deux classes :

- 'PDLType' : Cette classe permet de déterminer la nature de la formule PDL à traduire en se basant sur les expressions régulières de l'API javax.regex. Une formule PDL peut être une proposition, un chemin, une jonction de formule, une contrainte de comptage ou encore une contrainte modulo. Cette étape constitue une phase préparatrice pour la traduction des formules PDL en μ-calcul.
- 'PDL2μ-calculus' : C'est la classe principale du paquetage 'pdl2μ-calculus', elle invoque la classe 'PDLType' pour déterminer la nature de la formule PDL à traduire, ensuite elle fait appel à la classe 'μ-calculusFormula' pour produire la formule μ-calcul équivalente.

Le paquetage *exception* Ce paquetage est dédié à la gestion des exceptions dans le comportement de notre compilateur XPath, il définit quatre exceptions principale qui étendent la classe 'java.lang.exception' , à savoir :

- ParsingException : gère les erreurs engendrées par le parsing des expressions XPath.
- StaticCheckException : définit le comportement de notre compilateur XPath au cas ou l'analyse statique d'une expression échoue.
- PDLTranslationException : manipule l'occurrence des exceptions lors de la translation d'un concept XPath en PDL.
- μ-calculusTranslationException : manipule l'occurrence des exceptions lors de la translation d'un concept XPath en μ-calcul.

La seule exigence à laquelle doivent répondre ces exceptions c'est de générer un message d'erreur compréhensible par l'utilisateur. Ceci facilite la signalisation des erreurs et indique les raisons spécifiques de chaque erreur.

Le paquetage *gui* Notre compilateur offre une interface graphique conviviale et bien définie pour tester la satisfaisabilité des requêtes XPath. Cette interface permet à l'utilisateur de tester séparément les différentes étapes de notre procédure de décision en présentant chaque étape dans un onglet a part contenant les composants graphiques nécessaires pour la saisie des entrées et l'affichage des résultats relatifs à l'étape en question.
Ceci est dû à la nature séquentielle et facilement décomposable de notre procédure de décision et dû aussi au temps étendu que nous avons passé itérativement sur la conception et l'évaluation de notre compilateur. Cette interface présente un autre avantage qui est l'affichage des formules PDL et μ-calcul sous format algébrique aussi que sous format TeX pour convenir aux besoins des utilisateurs familiarisés avec ces deux formats d'écritures de formules logiques. Des figures montrant les détails de l'interface graphique sont présentées plus tard dans la section dédiée aux tests.

Le paquetage *test* Un paquetage additionnel a été crée pour englober tout le code d'essai. La raison principale pour laquelle nous avons séparé le code du compilateur du code des essais était de faciliter l'utilisation du compilateur au cas où quelques utilisateurs sont simplement intéressés par l'exécution des différentes phases de la procédure de décision sans accompagnement de tests. En outre nous croyons qu'il y a plus d'ordre dans la structuration du code de cette façon, facilitant probablement le développement. La figure 6.5 montre le code d'un test unitaire portant sur la translation d'une requête XPath en PDL.

6.2.3 Test et illustration

Cette section est dédié à l'évaluation finale de notre approche pour la satisfaisabilité des requêtes XPath avec contraintes de comptage. Nous essayons de déterminer ce qui a été accompli et ce qui a besoin toujours d'amélioration dans notre compilateur. Pour cela, nous allons tester les différentes fonctionnalités de ce compilateur en mettant l'accent sur le bon déroulement et les performances de chacune

```
package fr.inrialpes.wam.xpath.test;

import junit.framework.TestCase;

public class PDLTranslationTest extends TestCase {

    public static void main(String[] args) {
        junit.textui.TestRunner.run(PDLTranslationTest.class);
    }

    public PDLTranslationTest(String arg0) {
        super(arg0);
    }

    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /*
     * Test method for 'fr.inrialpes.wam.xpath.pdl.XPath2PDL.performXPath(String)'
     */
    public void testPerformXPath() {

        XPath2PDL xpath2pdl = new XPath2PDL();

        try {
            xpath2pdl.performXPath("self::BOOK/child::AUTHOR");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

FIG. 6.8: Test unitaire de la translation des concepts XPath en PDL

d'elles.

Les figures 6.9, 6.10 et 6.11 montrent l'exécution séquentielle de notre procédure de décision pour la requête XPath "child : :A [*count* (child : :B) > 2]".
Cette exécution comprend trois phases : d'abord la translation de la requête en question en formule PDL, détaillée par la figure 6.9 qui montre la formule PDL résultante sous format algébrique et sous format TeX, elle affiche également les durées relatives au 'parsing', à l'analyse statique et à la translation de cette requête.
La deuxième phase de notre procédure porte sur la translation de la formule PDL obtenue en μ-calcul, cette phase est illustré par la figure 6.10 qui montre la formule μ-calcul résultante toujours sous format algébrique et sous format TeX, elle affiche également la durée de la translation de cette formule.
La phase finale de notre procédure de décision consiste à établir la satisfaisabilité de la requête XPath considérée, cette phase n'est pas implémentée dans notre compilateur XPath mais elle fait appel au 'solveur' détaillé au début de ce chapitre, la figure 6.11 illustre le déroulement de cette étape.
Il est à déduire que notre compilateur est peu mûr pour être un gagnant dans les performances, mais nous croyons qu'il a quelque chose à offrir quand à la pertinence de la traduction des concepts XPath en μ-calcul, formant ainsi une base solide pour la décidabilité des requêtes XPath avec contraintes de comptage.

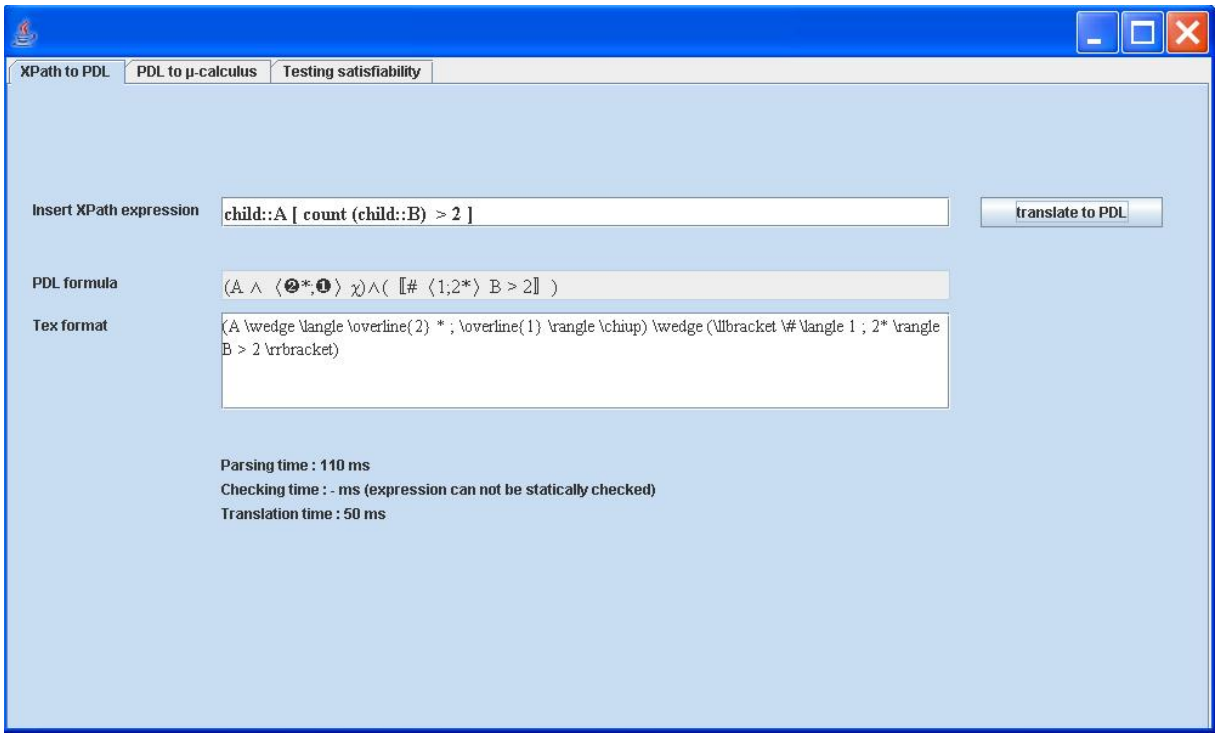


FIG. 6.9: Translation d'une requête XPath en PDL

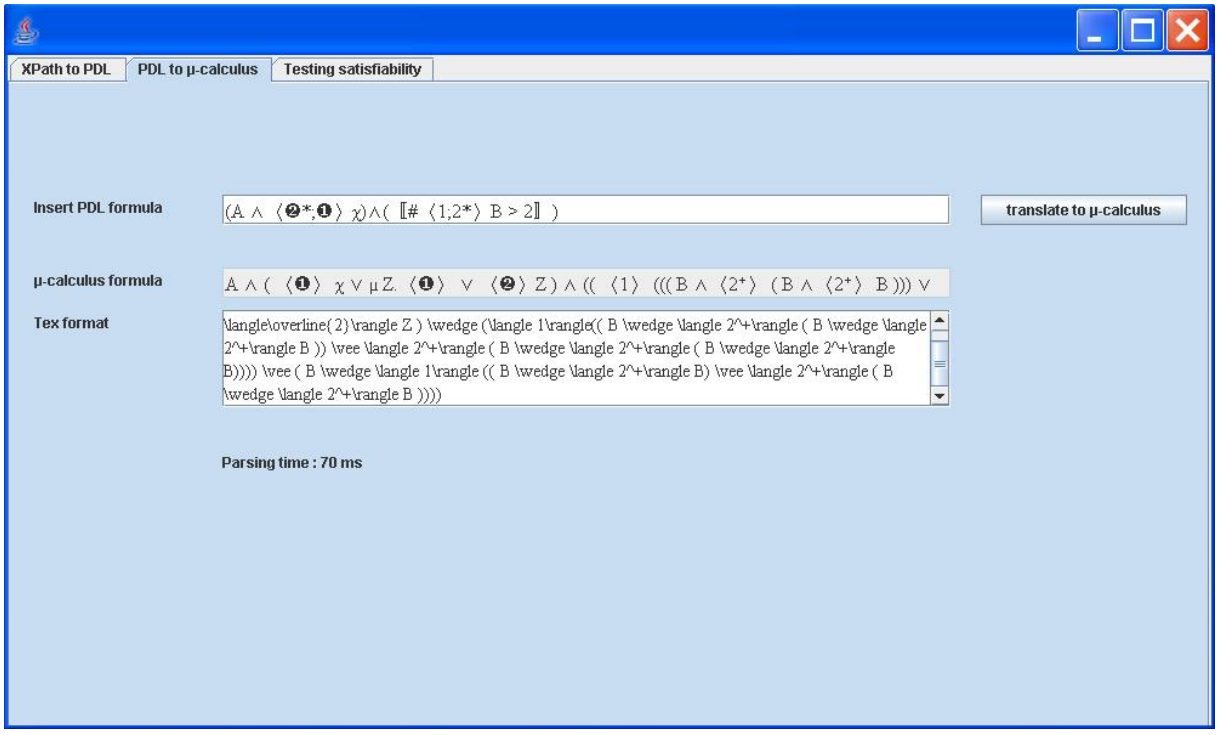


FIG. 6.10: Translation de la formule PDL en μ-calcul

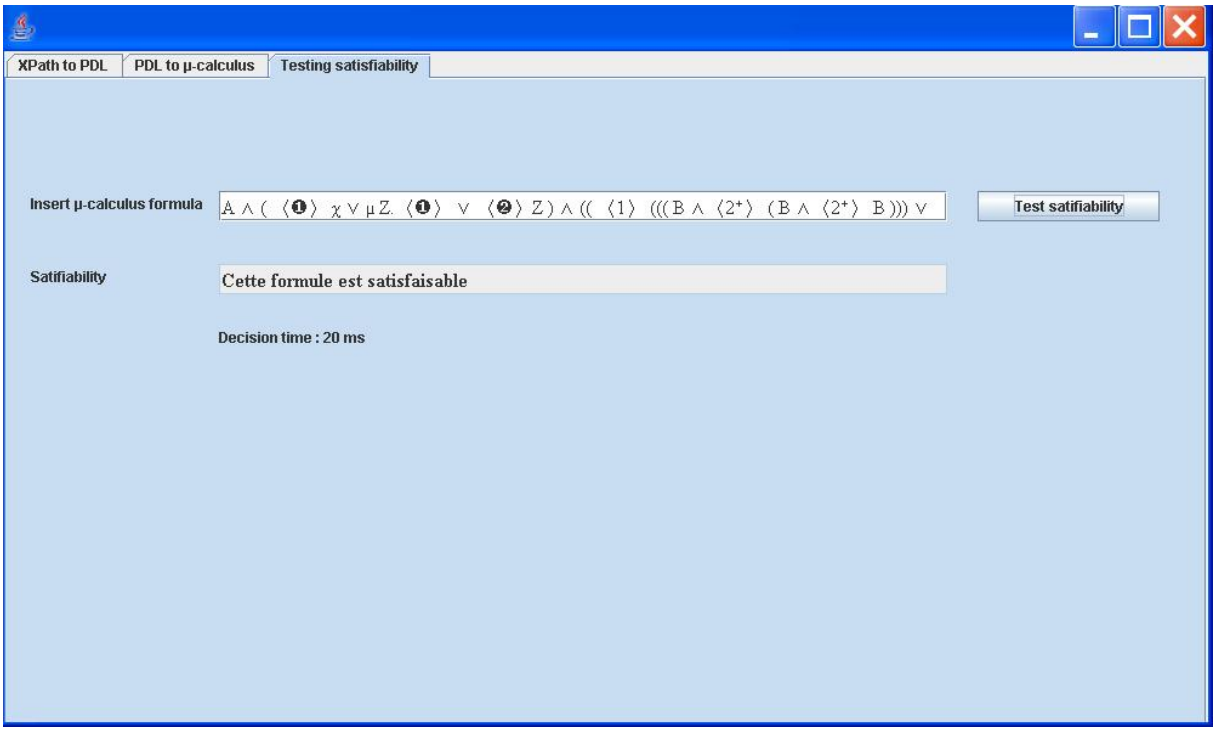


FIG. 6.11: Satisfaisabilité de la requête XPath

Chapitre 7

Conclusion

Notre conclusion comprend quatre parties. Dans la première partie nous rappelons les objectifs et les finalités qui ont motivé ce travail de recherche. La deuxième partie est dédiée à la synthèse du travail réalisé et la troisième partie présente les principales contributions de notre travail. Enfin, nous concluons par les perspectives ouvertes de ce mémoire.

7.1 Rappel des objectifs

Ce travail part de l'idée que le problème de décidabilité des requêtes XPath avec contraintes de comptage doit être résolu pour pouvoir manipuler avec sûreté et efficacité les structures de données dans le monde XML.
Dans cette perspective nous avons fixé comme objectifs, tout d'abord, l'identification d'une logique appropriée avec une expressivité suffisante pour supporter à la fois les contraintes de comptage et les langages d'arbres réguliers permettant la navigation et la sémantique de sélection de noeuds selon les spécificités du langage XPath ; ensuite, la résolution de manière efficace le problème de satisfaisabilité de cette logique qui permet de déterminer si une formule donnée de la logique admet un document XML qui la satisfasse.
En résumé, nos objectifs se récapitulent par la définition d'une logique offrant un meilleur compromis entre l'expressivité et la complexité [6], et qui est étroitement liée au contexte de XPath.

7.2 Travail réalisé

Nous sommes partis de l'état de l'art de notre sujet pour étudier les techniques de pointe existantes et les travaux de recherche reliés à nos objectifs. Dans le chapitre 2 nous avons introduit quelques fondations théoriques et formalismes utilisés dans la suite de ce travail tout en soulignant leur relation avec nos objectifs, et ceci au fur et à mesure que leurs concepts sous-jacents ont été présentés.
Grâce aux leçons tirées des investigations précédemment menées, nous avons réussi à établir la contribution principale de ce mémoire. Le chapitre 3 présente le fragment XPath avec contraintes de comptage à considérer dans ce travail et propose une logique d'arbres finis spécifiquement conçue pour ce fragment XPath. Cette logique définit le PDL comme langage de haut niveau assurant une translation préliminaire des concepts XPath, et la logique μ-calcul représentant le langage de base pour la décidabilité du fragment XPath considéré.
Les chapitre 4 et 5 proposent deux approches différentes et complémentaires pour l'interprétation des contraintes de comptage. Chaque approche répond d'une manière différente aux exigences de monotonie et du comptage ordonné imposées par les propriétés intrinsèques du comptage.
Enfin une procédure de décision pour tester la satisfaisabilité de la logique considérée a été proposée,

ainsi que les techniques pour son implantation. En outre, des tests ont été menés avec une implémentation complète du système, qui s'avère être assez intéressant au niveau de l'interprétation proposée pour les concepts XPath avec contraintes de comptage.

7.3 Principales contributions

La contribution principale de ce travail est une nouvelle logique conçue pour XPath avec contraintes de comptage, dérivée de PDL et du μ-calcul modal. Cette logique est assez expressive pour supporter, outre la navigation multi-directionnelle dans les arbres finis, les contraintes de comptage XPath ainsi que d'autres types d'expressions avancées qui ne figurent pas dans la spécification XPath 2.0 tels que les chemins conditionnés, les chemins itératifs et la clôture réflexive sur les chemins.
Une autre contribution de ce mémoire est de montrer comment compiler linéairement les contraintes de comptage XPath dans la logique proposée. Cette logique prend en considération un fragment XPath assez large et soutient la plupart des expressions XPath.
Un autre avantage de notre travail est que la logique considérée est une sous-logique de formalismes existants (PDL et μ-calcul) ce qui permet de tirer profit des résultats déjà établis pour ces logiques.
La principale application de ce travail est un compilateur qui interprète les requêtes XPath en PDL puis en μ-calcul afin de décider de leur satisfaisabilité via le "solveur" présenté dans les travaux de Genevès[10]. Ce compilateur utilise directement les résultats décrits dans ce mémoire.

7.4 Perspectives

La logique et les approches d'interprétation proposées dans ce travail de recherche constituent des résultats importants offrant une base solide pour une étude plus approfondie de la décidabilité des requêtes XPath avec contraintes de comptage.
Notre travail a atteint la majorité des objectifs définis au départ, néanmoins plusieurs optimisations demeurent possibles couvrant différents aspects du sujet.
D'abord un raffinement itératif s'impose à court terme afin d'améliorer les traductions proposées pour quelques concepts XPath constituant une limitation de notre travail actuel. En plus, il y a d'autres améliorations possibles qui portent sur l'adoption d'une approche de comptage parmi les deux approches proposées dans notre travail, ou encore aller plus loin dans l'optimisation en proposant une approche unifiée à partir de la synthèse de ces deux approches. Nous avons énuméré également un ensemble de tâches évidentes qui devraient être apportées au projet telles que la preuve de la "soundness and correctness" de notre procédure de décision et l'établissement de sa complexité algorithmique.
Les perspectives à long terme de notre travail consistent à élargir le fragment XPath considéré pour couvrir en premier lieu les contraintes de comptage au niveau des expressions et des chemins, ensuite augmenter progressivement les concepts XPath couverts jusqu'à atteindre la totalité du langage XPath.
Enfin, étendre l'expressivité du langage XPath pour égaler celle de l'arithmétique de Presburger reste un défis à relever dans une stade avancée de ce travail de recherche.

Bibliographie

[1] Abiteboul and S. Vianu. regular path queries with constraints. *Journal of Computer and System Sciences*, 1999. [citée dans p. 34]

[2] T. Accary-Barbier and S. Calabretto. *XML*. Techniques de l'ingénieur, 2005. [citée dans p. 10]

[3] L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx, and M. de Rijke. pdl for ordered trees. *NWO*, Mai 2004. [citée dans p. 29]

[4] Piero A. Bonatti, C. Lutz, A. Murano, and Moshe Y. Vardi. the complexity of enriched μ-calculi. *ICALP*, 2006. [citée dans p. 23]

[5] D. Colazzo, G. Ghelli, P. Manghi, and C. Sartiani. types for path correctness of xml queries. *Proceedings of the ninth ACM SIGPLAN international conference on Functional programming*, 2004. [citée dans p. 36]

[6] A. Dawar, P. Gardner, and G Ghelli. Expressiveness and complexity of graph logic. Technical report, Imperial College, 2004. [citée dans p. 75]

[7] S. Demri and D. Lugiez. complexity of modal logics with presburger constraints. Technical report, Laboratoire Spécification et Vérification, Octobre 2006. [citée dans p. 19]

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. [citée dans p. 66]

[9] P. Genevès. a satisfiability solver for xml and xpath decision problems. 2006. `http://wam.inrialpes.fr/software/xml-calculus/`. [citée dans p. 65]

[10] P. Genevès. *Logics for XML*. PhD thesis, Institut National Polytechnique de Grenoble, 2006. [citée dans p. 9, 20, 23, 63, 64, 65, 76]

[11] P. Genevès and N. Layaïda. a decision procedure for xpath containment. Technical Report 5867, INRIA, 2006. [citée dans p. 63]

[12] P. Genevès and Vion-Dury. logic-based xpath optimization. *Proceedings of the 2004 ACM Symposium on Document Engineering*, 2004. [citée dans p. 64]

[13] M. Marx. conditional xpath. *Proceeding of the International Conference on Database Theory*, 2004. [citée dans p. 25]

[14] V. Rusu. analyzing automata with presburger arithmetic and uninterpreted function symbols. Technical Report 4100, INRIA, Janvier 2001. [citée dans p. 28]

[15] H. Seidl, T. Shweintick, A. Muscholl, and P. Habermehl. counting in trees for free. *ICALP*, 2004. [citée dans p. 16]

[16] The World Wide Web Consortium (W3C). Document Object Model (DOM). `http://www.w3.org/DOM/`. [citée dans p. 69]

[17] The World Wide Web Consortium (W3C). eXtended Markup Language (XML). http://www.w3.org/XML/. [citée dans p. 9]

[18] The World Wide Web Consortium (W3C). XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20/. [citée dans p. 9, 66]

[19] The World Wide Web Consortium (W3C). XML Schema, Parts 0, 1, and 2. http://www.w3.org/XML/Schema. [citée dans p. 66]

[20] The World Wide Web Consortium (W3C). XQuery 1.0 and XPath 2.0 Data Model. http://www.w3.org/TR/xpath-datamodel/. [citée dans p. 67]

[21] The World Wide Web Consortium (W3C). XQuery 1.0 and XPath 2.0 Formal Semantics. http://www.w3.org/TR/xquery-semantics/. [citée dans p. 38]

[22] The World Wide Web Consortium (W3C). XQuery 1.0 and XPath 2.0 Functions and Operators. http://www.w3.org/TR/xquery-operators/. [citée dans p. 13]

[23] S. Dal Zilio and D. Lugiez. xml schema, tree logic and sheaves automata. Technical Report 4631, INRIA, Novembre 2002. [citée dans p. 17]