LEONARDO CESAR MACHADO

Tradução de consultas XPath para predicados Prolog

Dissertação de Mestrado apresentada ao programa de pós-graduação em Computação da Universidade Federal Fluminense, como parte dos requisitos necessários à obtenção do título de Mestre em Computação.

Área de concentração: Engenharia de Sistemas e Informação.

Orientador: Prof.ª D.Sc. VANESSA BRAGANHOLO MURTA

Niterói

2016

LEONARDO CESAR MACHADO

Tradução de consultas XPath para predicados Prolog

Dissertação de Mestrado apresentada ao programa de pós-graduação em Computação da Universidade Federal Fluminense, como parte dos requisitos necessários à obtenção do título de Mestre em Computação.
Área de concentração: Engenharia de Sistemas e Informação.

Aprovada em Julho de 2016.

BANCA EXAMINADORA

Prof.ª D.Sc. Vanessa Braganholo Murta – Orientadora
UFF

Prof. D.Sc. Daniel Cardoso Moraes de Oliveira
UFF

Profª. D.Sc. Fernanda Araújo Baião Amorim
UNIRIO

Niterói
2016

A Suellen Fernanda Antunes Pereira Machado.

AGRADECIMENTOS

A Deus por renovar meu ânimo nos momentos difíceis e permitir esta conquista.

Aos meus pais, Paulo e Nadja, pela formação, apoio constante e crença de que esta conquista era possível.

À minha esposa Suellen que me acompanhou nesta jornada, sempre de forma gentil, fornecendo compreensão, apoio e incentivo nos momentos de dúvida.

À minha irmã Karina por todo o apoio e incentivo no ingresso na área da Computação.

À professora Vanessa Braganholo por ter acreditado em mim, pela excepcional orientação, incentivo e extrema compreensão.

Ao professor Leonardo Murta por ter acreditado em mim e pelo apoio durante a pós-graduação.

À professora Ana Cristina pelo apoio no ingresso à pós-graduação.

Aos professores Daniel de Oliveira e Fernanda Baião por terem aceitado participar desta banca.

Aos meus familiares e amigos, pelo suporte e compreensão nas minhas ausências.

À todos do IC, pelo aprendizado e crescimento proporcionado.

RESUMO

A necessidade da utilização de informações presentes em bases de dados semiestruturadas, especialmente XML, cresce a cada dia. O atendimento a esta demanda deve ser realizado da maneira mais rápida e satisfatória possível. Enquanto algumas abordagens de consulta sobre documentos XML utilizam técnicas dependentes de sistemas gerenciadores de banco de dados específicos para otimização do tempo de resposta, a proposta com linguagens de inferência desenvolvida por Lima *et al.* (2012) mostrou-se bastante promissora. A abordagem de Lima *et al*. propõe um método de tradução de documentos XML para fatos Prolog, propiciando assim a realização de consultas utilizando inferência. Uma avaliação *ad-hoc* mostrou que consultas Prolog sobre essas bases de fatos eram processadas mais eficientemente do que as consultas XML equivalentes. Diante disso, Santos *et al*. (2012) desenvolveram um algoritmo básico de tradução automática de consultas XPath para predicados Prolog. No entanto o algoritmo oferece somente suporte a consultas simples, não contemplando a utilização de filtros múltiplos, expressões e funções presentes na XPath. Este trabalho se baseia em de trabalhos anteriores com o objetivo principal de oferecer apoio à tradução de um conjunto mais amplo de consultas, assim como a obtenção automatizada de resultados em formato XML. O trabalho desenvolvido (XP2PL) traduz para Prolog diversas funções, expressões e filtros presentes na XPath, permitindo ao usuário maior poder de expressão no momento da criação das consultas, quando comparado com abordagens semelhantes. Experimentos foram realizados comparando os resultados obtidos por este trabalho com os dos interpretadores XPath Galax e Saxon, além do banco de dados XML nativo Sedna. As consultas do *benchmark* XPathMark foram utilizadas para analisar o poder de expressão, além de medir os tempos de resposta de cada abordagem. Os resultados alcançados foram promissores. Em relação ao desempenho, a maioria das consultas Prolog apresenta desempenho superior quando comparadas a três diferentes processadores XPath avaliados nesse trabalho, utilizando base de 10 MB.

Palavras-chave: Processamento de Consulta, Prolog, XML, XPath.

ABSTRACT

The need to use information contained in semi-structured databases, especially XML, grows every day. Fullfilling this demand should be done as quickly and satisfactorily as possible. While some query approaches to XML documents use specific database management systems techniques to optimize the response time, the proposal with inference languages by Lima et al. (2012) proved to be quite promising. Lima *et al.* translates XML documents into Prolog facts, thereby providing query capabilities using inference. An ad-hoc evaluation showed that Prolog queries over these knowledge bases run fastaer than their equivalente XML queries. Then, Santos et al. (2012) developed a basic algorithm to automatically translate XPath queries into Prolog predicates. However, the algorithm provides support only for simple queries, not contemplating the use of multiple filters, expressions and XPath functions. This work builds upon previous approaches with the main objective of supporting the translation of a set of new queries, and also to automatically obtaining results in XML format. The work developed (XP2PL) translates several XPath functions, expressions and filters to Prolog, allowing the user greater power of expression in query creation process, when compared with similar approaches. Experiments were performed comparing the results of this work with the XPath interpreters Galax and Saxon, and also with the Sedna native XML database. Queries of XPathMark benchmark were used to analyze the expression power and measure the response times of each approach. The results were promising The majority of the Prolog queries presented better performance when compared to the three different XPath processors that were evaluated in this work, using a 10 MB dataset.

Keywords: Query Processing, Prolog, XML, XPath.

LISTA DE ILUSTRAÇÕES

Figura 1: XML Schema – Composição do elemento raiz *dblp* 20
Figura 2: XML Schema – Composição do elemento *article* 21
Figura 3: XML Schema – Composição do elemento *proceedings* 21
Figura 4: XML Schema – Composição do elemento *book* 22
Figura 5: XML Schema – Composição do elemento *mastersthesis* 22
Figura 6: XML Schema – Composição do elemento *phdthesis* 23
Figura 7: XML Schema – Composição do elemento *www* 23
Figura 8: Documento XML baseado no esquema DBLP 24
Figura 9: Documento HTML e sua respectiva tradução Prolog, realizada pelo pacote *SWI-Prolog SGML/XML parser* 29
Figura 10: Documento XML de exemplo utilizado por Almendros-Jiménez *et al*. (2008) 30
Figura 11: Árvore XML que ilustra os identificadores utilizados por Almendros-Jiménez *et al*. (2008) 31
Figura 12: Fatos Prolog traduzidos pela abordagem de Almendros-Jiménez *et al*. (2008) 32
Figura 13: Regras Prolog criadas pela abordagem de Almendros-Jiménez *et al*. (2008) 32
Figura 14: Exemplo de reordenação de regras para um consulta XPath, por Almendros-Jiménez *et al*. (2008) 33
Figura 15: Etapas da abordagem XMLInference de Lima *et al.* (2012) 35
Figura 16: Fato correspondente ao elemento raiz *dblp* 36
Figura 17: Tradução do elemento complexo *article* 37
Figura 18: Tradução de elemento simples e sem atributos *title* 37
Figura 19: Exemplo de tradução do atributo *mdate* 37
Figura 20: Exemplo de tradução de um elemento misto 38
Figura 21: Criação de regras para o delimitador *sequence* 39
Figura 22: Criação de regras para delimitador *choice* 40
Figura 23: Criação de regras para delimitador *all* 40
Figura 24: Criação de regras para delimitador *sequence* com filho *choice* 41
Figura 25: Criação de regras para delimitador *choice* com filho *sequence* 42
Figura 26: Simplificação de *sequences* aninhados 42

Figura 27: Simplificação de *choices* aninhados 43
Figura 28: Regra manual *sameYearPublication* 43
Figura 29: Arquitetura da abordagem XP2PL 45
Figura 30: Interface principal do protótipo XMLInference 46
Figura 31: Composição do elemento *keywordType* 49
Figura 32: Regras automáticas para o elemento *keyword* do tipo *keywordType* 49
Figura 33: XMLInference: Execução de consulta XPath traduzida para Prolog e sua respectiva resposta 50
Figura 34: Composição do elemento *keywordType*, e seus respectivos predicados de apoio 51
Figura 35: Consulta XPath e a representação de seu filtro em formato de árvore 53
Figura 36: Exemplo de expressão matemática dentro de filtro em uma XPath e seu respectivo formato de árvore 54
Figura 37: Exemplo de correlação entre regras 55
Figura 38: Composição do elemento *keyword* e seus respectivos predicados Prolog .. 56
Figura 39: Composição do elemento *personref* e seus respectivos predicados Prolog 57
Figura 40: Exemplo de filtro em uma em uma XPath e sua respectiva tradução Prolog 57
Figura 41: Exemplo de filtro intermediário em uma XPath e sua respectiva tradução Prolog 58
Figura 42: Armazenamento do resultado de funções em variáveis 58
Figura 43: Seleção múltipla de elementos intermediários em uma XPath 59
Figura 44: Seleção de múltiplos elementos no resultado de uma XPath 59
Figura 45: Seleção de múltiplos atributos em um filtro XPath 59
Figura 46: Consulta XPath com múltipla de elementos intermediários e sua respectiva tradução Prolog 59
Figura 47: Consulta XPath com seleção de múltiplos elementos no resultado e sua respectiva tradução Prolog 60
Figura 48: XPath com operador *".."* e sua respectiva tradução Prolog 60
Figura 49: Seleção do elemento pai como resultado em uma XPath e sua respectiva tradução Prolog 61
Figura 50: Tradução de consulta com o operador *"..."* para múltiplos pais 61
Figura 51: Delimitação do elemento pai retornado em consulta XPath 62

Figura 52: Delimitação de parentesco e seleção múltipla em consulta XPath, e sua respectiva tradução Prolog....62
Figura 53: Exemplo de XPath com eixo *following*....63
Figura 54: Exemplo de XPath com eixo *preceding*....63
Figura 55: Consulta XPath com a expressão *following-sibling* e sua respectiva tradução Prolog....63
Figura 56: Consulta XPath com a expressão *preceding-sibling* e sua respectiva tradução Prolog....63
Figura 57: Regra Prolog para obtenção do *ids* dos elementos descendentes....64
Figura 58: Regra Prolog para obtenção do *ids* dos elementos ancestrais....64
Figura 59: Consulta XPath com a expressão *following* e sua respectiva tradução Prolog....65
Figura 60: Consulta XPath com a expressão *preceding* e sua respectiva tradução Prolog....65
Figura 61: Função Prolog *indexOf*()....65
Figura 62: Função Prolog *listSize*()....66
Figura 63: Consulta XPath com a função *position*() e sua respectiva tradução Prolog 66
Figura 64: Consulta XPath com as funções *position*() *last*() e sua respectiva tradução Prolog....66
Figura 65: XPath utilizando operador de união e sua respectiva tradução Prolog....67
Figura 66: XPath com a função *local-name*() e sua respectiva tradução Prolog....67
Figura 67: Consulta XPath utilizando função de processamento de texto e sua tradução Prolog....68
Figura 68: Exemplo de utilização da função *count*() e sua respectiva tradução Prolog69
Figura 69: Função Prolog *count*()....69
Figura 70: Tradução Prolog para a função *sum*()....69
Figura 71: XPath utilizando função *sum*() e sua respectiva tradução Prolog....69
Figura 72: XPath utilizando função *floor*() e sua respectiva tradução Prolog....70
Figura 73: XPath utilizando função *ceiling*() e sua respectiva tradução Prolog....70
Figura 74: Assinatura da função Prolog de impressão de elementos simples e complexos....71
Figura 75: Assinatura da função Prolog de impressão de atributos....71
Figura 76: Assinatura da função Prolog de impressão de elementos mistos....71

Figura 77: Assinatura da função Prolog de impressão do elemento raiz do documento 71
Figura 78: Exemplo de regra Prolog de impressão de elementos 72
Figura 79: Regra Prolog para impressão dos elementos filhos de *book* 72
Figura 80: Exemplo de regra Prolog para impressão de atributos 73
Figura 81: Exemplo de regra Prolog para impressão de elemento complexo 73
Figura 82: Exemplo de regra Prolog para impressão dos filhos de elemento complexo 73
Figura 83: Composição do elemento *X* (Relacionamento cíclico direto) 73
Figura 84: Composição do elemento *Y* (Relacionamento cíclico direto) 73
Figura 85: Composição do elemento *X* (Relacionamento cíclico indireto) 74
Figura 86: Composição do elemento *Y* (Relacionamento cíclico indireto) 74
Figura 87: Composição do elemento *Z* (Relacionamento cíclico indireto) 74
Figura 88: Elemento *svg:SVG* e resultado esperado da XPath *//*[namespace::svg]* ... 81
Figura 89: Tradução Prolog para a consulta //**[lang('it')]* 82
Figura 90: *Script* de execução e tomada de tempo de consultas para o interpretador Saxon 84
Figura 91: Trecho de código C++ referente à execução e tomada de tempo de consultas Prolog 85
Figura 92: Comparação de tempos de execução entre Saxon, Galax, Sedna e XP2PL, para base de fatos e documento XML D1 de 10 MB 91
Figura 93: Consulta Q12 e sua respectiva tradução Prolog 92
Figura 94: Comparação de tempos de execução entre Saxon, Galax, Sedna e XP2PL, para base de fatos e documento XML D1 de 50 MB 93
Figura 95: Comparação de tempos de execução entre Saxon, Galax, Sedna e XP2PL, para base de fatos e documento XML D1 de 100 MB 94
Figura 96: Comparação de tempos de execução entre Saxon, Sedna e XP2PL, para base de fatos e documento XML D1 de 500 MB 95
Figura 97: Comparação de tempos de execução entre Sedna e XP2PL, para base de fatos e documento XML D1 de 1 GB 96
Figura 98: Consulta Q5 e sua respectiva tradução Prolog. 97
Figura 99: Consulta XPath Q24 e sua tradução Prolog. 99
Figura 100: Consulta ao pseudo elemento *sameyearpublication* utilizando XPath e sua respectiva resposta XML 108

LISTA DE TABELAS

Tabela 1: Abrangência de funções e operadores considerando as abordagens de Jiménez *et al.* (2008), Santos *et al*. (2012) 16

Tabela 2: Funções e operadores disponíveis no algoritmo de tradução automática de consultas XPath 75

Tabela 3: Consultas utilizadas durante os experimentos para o documento D1 78

Tabela 4: Consultas utilizadas durante os experimentos para o documento D2 79

Tabela 5: Capacidade de execução de consultas para o documento D1 80

Tabela 6: Capacidade de execução das consultas para o documento D2 81

Tabela 7: Abrangência de funções e operadores deste trabalho e das abordagens Prolog concorrentes 83

Tabela 8: Tempos de execução das consultas em milissegundos utilizando documento XML de 10 MB e sua respectiva tradução Prolog 86

Tabela 9: Tempos de execução das consultas em milissegundos utilizando documento XML de 50 MB e sua respectiva tradução Prolog 87

Tabela 10: Tempos de execução das consultas em milissegundos utilizando documento XML de 100 MB e sua respectiva tradução Prolog 88

Tabela 11: Tempos de execução das consultas em milissegundos utilizando documento XML de 500 MB e sua respectiva tradução Prolog 89

Tabela 12: Tempos de execução das consultas em milissegundos utilizando documento XML de 1 GB e sua respectiva tradução Prolog 90

Tabela 13: Tempos de execução das consultas em milissegundos para a consulta Q14 – XP2PL 97

Tabela 14: Tempos de execução das consultas em milissegundos para a consulta Q23 98

Tabela 15: Tempos de execução das consultas em milissegundos para a consulta Q44 98

Tabela 16: Tempos de execução das consultas em milissegundos para a consulta Q45 98

Tabela 17: Tempos de execução em milissegundos para a consulta Q24 100

Tabela 18: Consultas por cenário aonde a abordagem XP2PL foi superior 102

Tabela 19: Comparativo de tempo de resposta com a abordagem de Almendros-Jiménez *et al*. (2008) 105

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Iterface
DTD	Document Type Definition
DBLP	Digital Bibliography & Library Project
HTML	Hyper Text Markup Language
Prolog	Programmation on Logique
XML	Extensive Markup Language
XPath	XML Path Language
XQuery	XML Query
XSD	XML Schema Definition
W3C	World Wide Web Consortium

SUMÁRIO

Capítulo 1 - Introdução 15
1.1 Motivação 15
1.2 Objetivos 16
1.3 Organização dos capítulos 17
Capítulo 2 –Linguaguem XPath 19
2.1 Introdução 19
2.2 Funções e Operadores 25
2.3 Considerações finais 27
Capítulo 3 -Trabalhos relacionados 28
3.1 Considerações finais 34
Capítulo 4 - XMLInference 35
4.1 Introdução 35
4.2 Tradução de documentos XML para Prolog 36
4.3 Criação de regras Prolog automáticas e manuais 38
4.3.1 Regras automáticas 38
4.3.2 Regras manuais 43
4.4 Considerações finais 44
Capítulo 5 – Melhorias na abordagem XMLInference 45
5.1 Introdução 45
5.2 Máquina de execução Prolog 46
5.3 Nomenclatura dos predicados Prolog 47
5.4 Algoritmo de criação da base de fatos 48
5.5 Execução de consultas Prolog 48
5.6 Ocorrências de elementos 50
5.7 Considerações finais 51
Capítulo 6 – Algoritmo de tradução e impressão 52

6.1 Introdução 52
6.1.1 Visão geral do algoritmo de tradução 52
6.1.2 *Parsing* da consulta XPath 52
6.2 Algoritmo de tradução de consultas XPath 54
6.2.1 Nomenclatura das variáveis 55
6.2.2 Montagem de predicados de elementos 56
6.2.3 Tradução de filtros, funções e operadores 57
6.3 Funções numéricas 68
6.4 Algoritmo de impressão 70
6.5 Considerações finais 74
Capítulo 7 – Avaliação experimental 76
7.1 Introdução 76
7.2 Avaliação de expressividade 76
7.2.1 Obtenção das consultas 76
7.2.2 Tradução XPath Prolog 79
7.2.3 Análise dos resultados 79
7.3 Avaliação de desempenho 82
7.3.1 Geração das bases 82
7.4 Análise dos resultados 85
7.5 Ameaças à validade 102
Capítulo 8 – Conclusão 104
8.1 Contribuições 104
8.2 Limitações 106
8.3 Trabalhos futuros 106

Capítulo 1 - INTRODUÇÃO

1.1 MOTIVAÇÃO

A linguagem XML (CLARK; DEROSE, 1999) permite representar dados semiestruturados de forma flexível. Dados armazenados no formato XML possuem alta legibilidade e independência, podendo servir como forma de comunicação entre sistemas e plataformas distintas. Obtendo proveito destas e de outras características favoráveis, cada vez mais sistemas e desenvolvedores optam por armazenar e distribuir seus dados em formato XML. Porém, consultas a grandes volumes de dados em formato XML são custosas.

Neste contexto, algumas abordagens visam otimizar o tempo de resposta das consultas sobre documentos XML subdividindo a consulta inicial em subconsultas que são executadas em paralelo (ANDRADE *et al.*, 2006; FIGUEIREDO; BRAGANHOLO; MATTOSO, 2010; RODRIGUES; BRAGANHOLO; MATTOSO, 2011). Porém, essas alternativas muitas vezes continuam dependentes de sistemas gerenciadores de banco de dados especialmente criados para lidar com dados semiestruturados, ou dependem da disponibilidade de um ambiente de processamento de alto desempenho, com vários processadores.

Seguindo direção oposta, alguns outros trabalhos utilizam linguagens de inferência, mais precisamente a linguagem Prolog (BRATKO, 2001), para executar consultas XPath (BOAG *et al.*, 2010). Os trabalhos de Jiménez *et al.* (2008) e de Santos *et al*. (2012) se encaixam nesse contexto. Ambos os trabalhos assumem que o documento XML foi traduzido para Prolog e fornecem algoritmos para traduir consultas XPath para Prolog de forma automática. De fato, Lima (2012) e Santos (2015) mostram, em suas dissertações de mestrado, que em diversas situações, consultas XML processadas pelo processador Prolog têm melhor desempenho do que as consultas equivalentes processadas por um processador XML nativo.

No entanto, ambas as abordagens sofrem com baixo poder de expressão, uma vez que a gama de operadores e funções XPath suportadas pelos algoritmos de tradução de Jiménez *et al.* (2008) e de Santos *et al*. (2012) é bastante pequena. A Tabela 1 exibe um conjunto dos principais operadores e funções utilizados na linguagem XPath (FRANCESCHET, 2005), indicando quais são suportados por cada trabalho. Analisando a tabela é possível verificar que existe uma grande lacuna quanto à expressividade de consultas XPath que os algoritmos existentes atendem.

Tabela 1: Abrangência de funções e operadores considerando as abordagens de Jiménez *et al.* (2008), Santos *et al*. (2012)

Funções e Operadores	Jiménez et al. (2008)	Santos et al. (2012)
/	X	X
//	X	X
=	X	X
>	X	X
<	X	X
>=	X	X
<=	X	X
*	X	
Utilização de eixos (*parent::*, *following::*, ...)		
.		
..	X	
|	X	
Funções de posição (*position()*, *last()*, ...)		
Tramento de namespaces (*local-name()*, *name()*, ...)		
lang()		
Tratamento de strings (*substring()*, *translate()*, ...)		
Tratamento de números (*floor()*, *ceiling()*, ...)		
Funções de agregação (*sum()*, *count()*, ...)		

1.2 OBJETIVOS

A tradução das consultas XPath é uma tarefa complexa e deve levar em consideração o conjunto de elementos requisitados pelo usuário (respeitando as relações de parentesco entre os elementos), além de possíveis filtros, expressões e funções envolvidas. Conforme já mencionado, os trabalhos anteriores (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008; SANTOS; PINHEIRO; BRAGANHOLO, 2012) possuem limitações por não permitirem o uso de filtros complexos ou ainda funções e expressões mais elaboradas da XPath.

Este trabalho tem como objetivo principal suprir essas limitações, agregando a possibilidade de utilização de funções, filtros e expressões comumente utilizadas na XPath. Nós contribuímos com um algoritmo de tradução de consultas XPath para Prolog que é capaz de traduzir consultas XPath complexas. Além da tradução da consulta em si, o algoritmo também traduz a resposta da consulta traduzida Prolog novamente para XML, deixando o processo completamente transparente para o usuário.

De fato, a transparência do processo é outro objetivo importante desse trabalho. Diferentemente de outras abordagens que utilizam linguagens lógicas para consultar dados

semiestruturados, a técnica desnvolvida neste trabalho objetiva minizar quaisquer adaptações ou aprendizado extra que o usuário tenha de realizar no momento da elaboração e execução de uma consulta. Além disso, esta abordagem aperfeiçoa e unifica técnicas de trabalhos anteriores, partindo desde a tradução do documento XML até a automatização da resposta da consulta XPath.

Para avaliar a aplicabilidade do algoritmo na prática, realizamos dois tipos de experimentos. O primeiro refere-se à completude da tradução. De forma secundária, experimentos relativos à desempenho foram realizados para avaliar a escalabilidade da abordagem proposta. Bases Prolog e documentos XML de diferentes tamanhos foram utilizados em múltiplas sequências experimentos. Os tempos de resposta da abordagem descrita neste trabalho foram comparados com o desempenho de um banco de dados XML nativo e de dois interpretadores XPath.

Como contribuição prática o trabalho disponibiliza um protótipo, onde há a automatização da tradução e execução de consultas XPath, além da exibição dos resultados das consultas em formato XML.

1.3 ORGANIZAÇÃO DOS CAPÍTULOS

O restante desta dissertação descreve os esforços para alcançar os objetivos já expostos, assim como a metodologia utilizada. Ele está organizado conforme a seguir.

O Capítulo 2 apresenta a fundamentação teórica desse trabalho. O capítulo menciona as principais funções, operadores e filtros presentes na linguagem XPath.

O Capítulo 3 aborda os trabalhos relacionados destacando seus objetivos e características. No decorrer do capítulo são destacadas características semelhantes destes trabalhos em relação à esta abordagem.

O Capítulo 4 objetiva descrever a abordagem original XMLInference (LIMA *et al.*, 2012) expondo as primeiras técnicas de tradução de documentos XML para fatos Prolog, assim como a geração de regras manuais e automáticas. No decorrer do capítulo são mostrados detalhes de todas as etapas destes processos, analisando alguns casos específicos.

O Capítulo 5 descreve as melhorias e correções realizadas na abordagem XMLInference. Neste capítulo cada modificação realizada é justificada pontuando os problemas e soluções encontrados ao longo da pesquisa.

O Capítulo 6 apresenta o algoritmo de tradução proposto por este trabalho, XP2PL, apresentando as estratégias e métodos utilizados na identificação e tradução das funções, expressões e filtros utilizados em consultas XPath, além do processo automático de obtenção

dos resultados em formato XML. Ao longo do capítulo as funções, expressões e filtros XPath são subdivididos em grupos contendo seus detalhes de implementação e técnicas de simplificação.

O Capítulo 7 apresenta a avaliação experimental do algoritmo proposto. O capítulo apresenta a metodologia, objetivos e resultados alcançados.

Por fim, o Capítulo 8 traz as conclusões obtidas com o desenvolvimento da pesquisa assim como suas contribuições. O capítulo encerra enumerando possíveis trabalhos futuros que possam agregar ao tema pesquisado.

Capítulo 2 –LINGUAGUEM XPATH

2.1 INTRODUÇÃO

XPath (CLARK; DEROSE, 1999) é uma linguagem de consulta para documentos XML, baseada em expressões de caminho. Através da XPath é possível selecionar nós de um documento XML por intemédio da delimitação do caminho correspondente até o elemento alvo. Os exemplos deste capítulo utilizam um documento XML cujo esquema foi inspirado na DBLP (LEY, 2003). A DBLP armazena informações acerca de publicações na área da Computação. O esquema, apresentado nas figuras a seguir, sofreu algumas modificações por questões didáticas.

A Figura 1 apresenta a estrutura do elemento raiz *dblp*. Esse elemento tem como filhos os elementos do tipo *article*, seguido de elementos dos tipos *proceedings*, *book*, *mastersthesis*, *phdthesis* e *www*. A estrutura do elemento *article* é mostrada na Figura 2, enquanto a Figura 3 mostra a estrutura do elemento *proceedings*. O elemento *book* tem sua estrutura exibida na Figura 4, assim como a Figura 5 ilustra a composição do elemento *mastersthesis*. Já a Figura 6 exibe a estrutura do elemento *phdthesis*, e a estrutura do elemento *www* pode ser visualizada na Figura 7. Por fim, a Figura 8 exibe um exemplo de documento XML correspondente a estas estruturas.

Cada passo do caminho de uma consulta XPath é delimitado pelo operador "/". A consulta */dblp/article*, por exemplo, retorna todos os elementos *article* filhos de *dblp*. Em consultas XPath, o último elemento definido no caminho é considerado o alvo da consulta. Todas as ocorrências do elemento alvo presentes no documento XML são retornadas, sempre respeitando as condições estabelecidas e o caminho definido na consulta. Ainda há a possibilidade de seleção de elementos sem a definição explícita do seu caminho anterior através da utilização do operador "//". A consulta *//authors* seleciona diretamente todos os elementos *authors* do documento, sem a necessidade de informar seu caminho completo. Este operador ainda permite realizar a seleção de elementos que possam estar em caminhos distintos em um documento XML. Como exemplo, podemos considerar a consulta *//title*. É possível observar na Figura 8 que o elemento *title* está presente na composição de diversos elementos como *article* e *book*. Desta forma, esta consulta retorna todas as ocorrências do elemento *title* presentes no documento, independente dos respectivos caminhos.

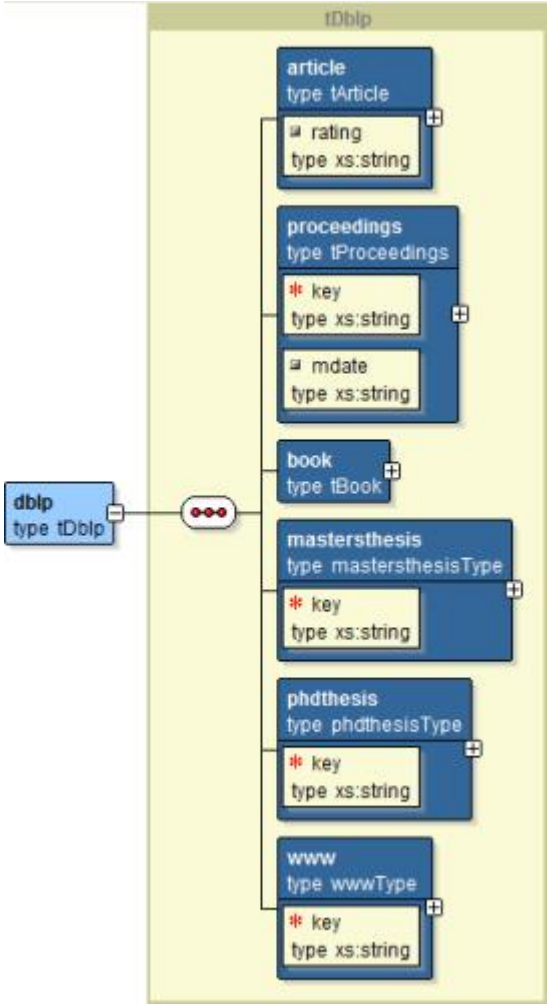


Figura 1: XML Schema – Composição do elemento raiz *dblp*

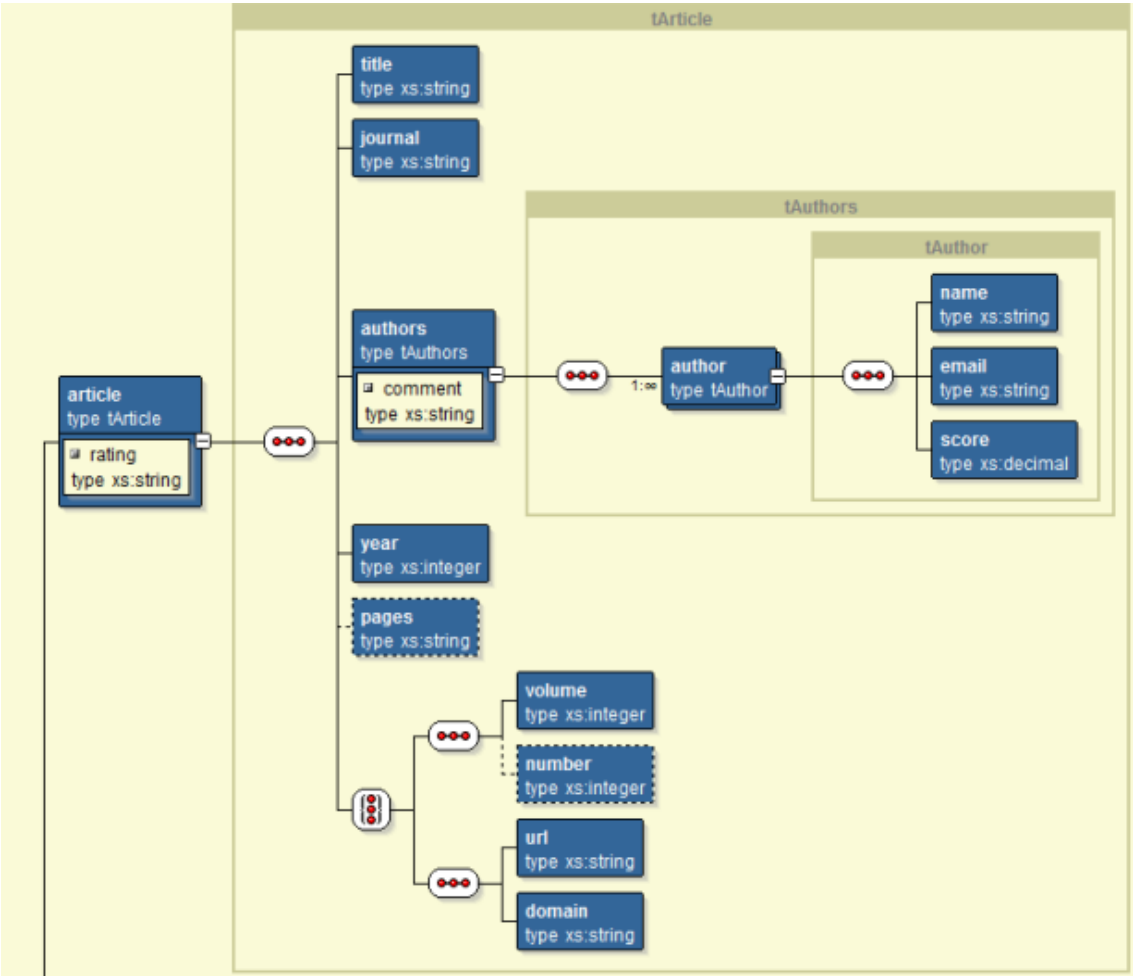


Figura 2: XML Schema – Composição do elemento *article*

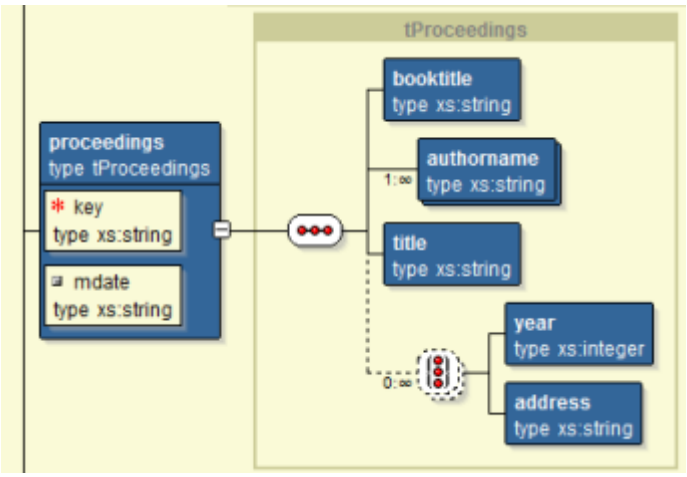


Figura 3: XML Schema – Composição do elemento *proceedings*

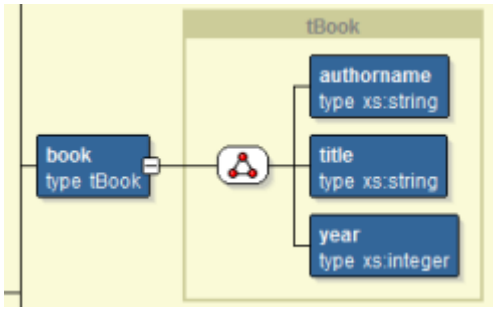


Figura 4: XML Schema – Composição do elemento *book*

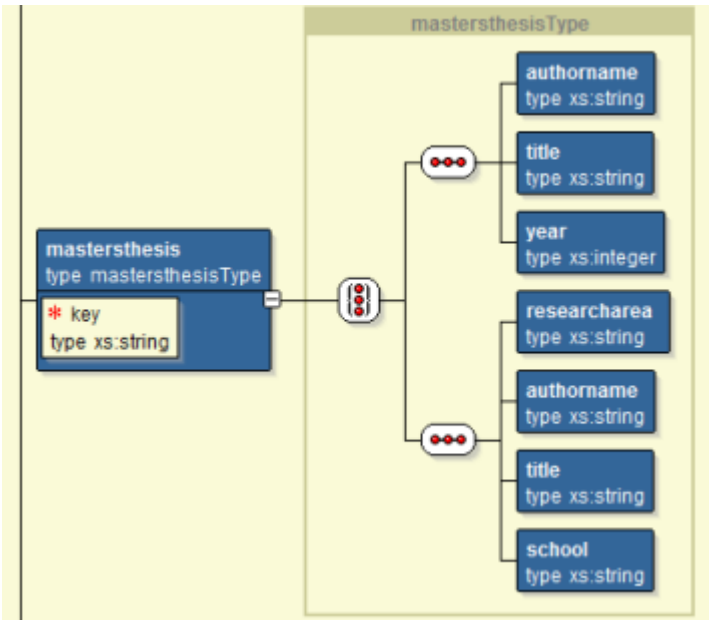


Figura 5: XML Schema – Composição do elemento *mastersthesis*

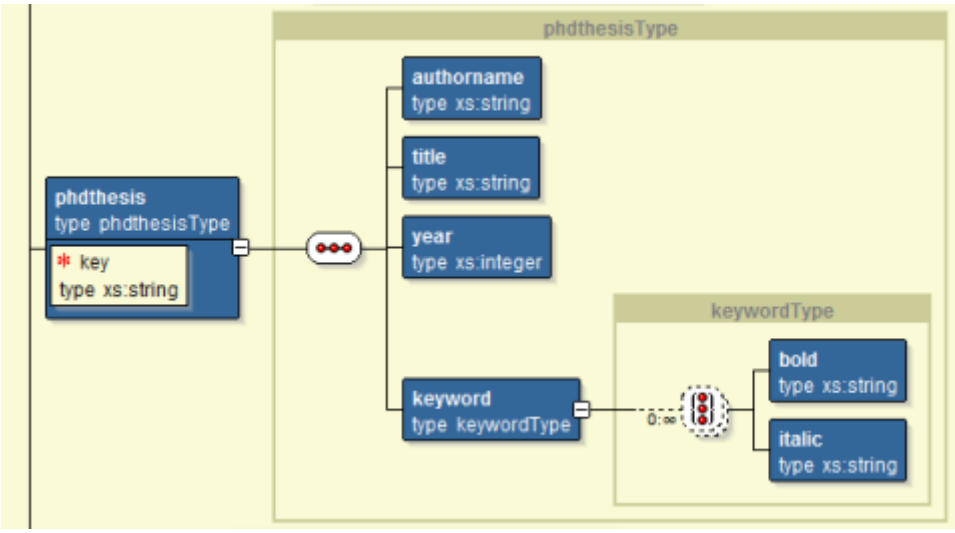


Figura 6: XML Schema – Composição do elemento *phdthesis*

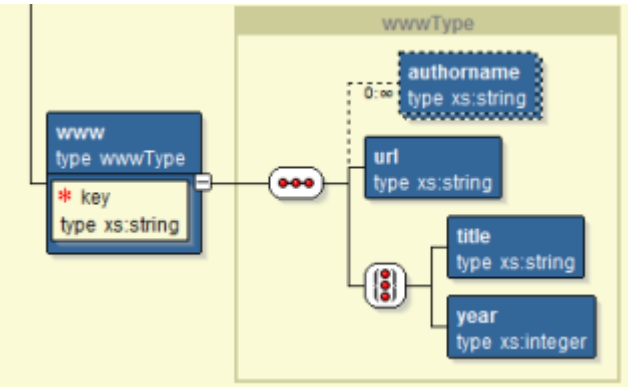


Figura 7: XML Schema – Composição do elemento *www*

```
1 <dblp>
2  <article rating="A">
3   <title>XPath Translation</title>
4   <journal>J. Sem. Data</journal>
5   <year>2016</year>
6   <authors comment="none">
7    <author>
8     <name>Leo Cesar</name>
9     <email>Cesar@dblp.com</email>
10    <score>9.20</score>
11   </author>
12   <author>co-author
13    <name>Cesar Mac</name>
14    <email>Mac@dblp.com</email>
15    <score>9.50</score>
16   </author>
17  </authors>
18  <volume>24</volume>
19  <number>1</number>
20 </article>
21 <proceedings key="journals/ws/2016-04" mdate="2016-04-01">
22  <booktitle>The Semistructured Data Challenge 2016</booktitle>
23  <authorname>Leo Cesar</authorname>
24  <authorname>Cesar Mac</authorname>
25  <title>Journal of Semistructured Data</title>
26  <year>2016</year>
27  <address>R. Edmundo March, s/n - Boa Viagem, Niteroi, Rio de Janeiro, Brazil</address>
28 </proceedings>
29 <book>
30  <authorname>Semistructured Data Author</authorname>
31  <title>Semistructured Data</title>
32  <year>2016</year>
33 </book>
34 <mastersthesis key="mastersthesis/l/LeoCesar">
35  <authorname>Leo Cesar</authorname>
36  <title>Translation XPath to Prolog</title>
37  <year>2016</year>
38 </mastersthesis>
39 <phdthesis key="phdthesis/m/MacCesar">
40  <authorname>mac Cesar</authorname>
41  <title>Translation XQuery to Prolog</title>
42  <year>2016</year>
43  <keyword>Prolog</keyword>
44  <keyword>Translation <bold>XQuery</bold> </keyword>
45 </phdthesis>
46 <www key="homepages/l/LeoCesar">
47  <authorname>Leo Cesar</authorname>
48  <url>http://www.leohomepage.com</url>
49  <title>Leo Home Page</title>
50 </www>
51 <www key="homepages/m/MachadoLeo">
52  <authorname>Machado Leo</authorname>
53  <url>http://www.machadohomepage.com</url>
54  <year>2016</year>
55 </www>
56 </dblp>
```

Figura 8: Documento XML baseado no esquema DBLP

Referências a atributos também podem ser utilizadas através do operador “@”. A consulta *//article/@rating* seleciona todos os atributos *rating* de elementos *article*. A seleção de elementos alvo, ou de elementos envolvidos em um passo do caminho, pode ser restringida pela presença de filtros. Os filtros em uma consulta XPath são delimitados pela ocorrerência da abertura e fechamento de colchetes (“[...]”). Como exemplo, podemos citar a consulta *//book[year = ‘2016’]*, aonde o objetivo é a seleção dos elementos *book* que possuam o elemento filho *year* com valor ‘2016’. A presença de um filtro atua somente no nível do caminho atual, não alterando seu contexto. Desta forma é possível realizar filtros intermediários à definição do caminho de uma XPath. A consulta */dblp/article[@rating = ‘A’]/title*, busca retornar os elementos *title* que possuam os elementos *dblp* e *article* como ancestrais. A aplicação do filtro *[@rating = ‘A’]* restringe que os elementos *article* pais de *title* devam possuir o atributo rating com o valor ‘A’. A caraterística booleana da aplicação de filtros em uma XPath permite a utilização de vários operadores de comparação (>, ≥, <, ≤, *OR*, *AND*, *NOT*), além de diversas funções.

2.2 FUNÇÕES E OPERADORES

Conforme já mencionado, durante a definição de uma consulta XPath há a possibilidade de utilização de diversos tipos operadores e funções, que podem delimitar o conjunto final de elementos resposta. Segundo a definição formal da W3C (CLARK; DEROSE, 1999; MICHAEL KAY, 2014), há um grande conjunto de operadores e funções passíveis de utilização em uma XPath. Maiores detalhes podem ser encotrados no *site* da W3C (https://www.w3.org/TR/xpath-functions-30 e https://www.w3.org/TR/xpath).

Operações com operadores e eixos relativos à posição. A XPath permite realizar referências posicionais a um ou mais elementos em uma consulta. Todos os elementos de um determinado nível do caminho podem ser referenciados através do operador "*". A consulta */dblp/*/tile*, busca obter todos os elementos *title* que tenham como ancestral o elemento *dblp*, omitindo seus possíveis elementos pais. De forma semelhante também é possível referenciar um nível anterior do caminho através do operador "..". A consulta *//title/..*, por exemplo, seleciona os elementos pais do elemento *title*.

Os eixos *preceding::* e *following::* podem ser utilizados para respectivamente selecionar elementos anteriores ou posteriores a um determinado elemento de referência. A consulta *//authors/author[@comment = 'none']/ following::author* exemplifica o uso do eixo *following::*, onde os demais elementos *author* posteriores ao primeiro elemento *author*, com o

atributo *comment* igual a *'none'*, são retornados. Os eixos *preceding-sibling::* e *following-sibling::* realizam operação semelhante, considerando somente os elementos irmãos ao elemento de referência.

Operações numéricas. Funções numéricas podem ser utilizadas em uma consulta XPath, transformando conteúdos numéricos de um determinado elemento. Exemplos desse tipo de função são *abs(), ceiling(), floor(), round(), round-half-to-even(),* etc.. A consulta *//article[sum(volume + number) > 30]*, por exemplo, filtra o resultado final retornando somente os elementos *article* que possuam a soma dos conteúdos dos elementos *volume* e *number* maior que 30. Operações matemáticas básicas também podem ser realizadas através de funções como *numeric-add(), numeric-subtract(), numeric-multiply(), numeric-divide(), numeric-mod(),* assim como operações mais complexas como *pi(), exp(), log(), pow(), sqrt(),* etc..

Funções de processamento de texto. Funções que atuam sobre conteúdo textual também são disponibilizadas pela linguaguem. Funções como *concat(), string-join(), substring(), string-lenght(), normalize-space(), normalize-unicode(), upper-case(), lower-case()* e *translate()* podem ser utilizadas no momento da construção de uma consulta XPath. A consulta *//book[string-lenght(title) > 10]*, por exemplo, considera somente os elementos *book* que contenham filhos *title* com mais de 10 caracteres.

Referências a URI. A linguaguem XPath permite ainda realizar referências a valores de URI (*Uniform Resource Identifier*), através das funções *resolve-uri(), encode-for-uri(), iri-to-uri()* e *escape-html-uri().*

Operações booleanas. Expressões e funções booleanas (*boolean-equal(), boolean-less-than(), boolean-greater-than(), boolean(), not()*) podem ser utilizadas para verificar e comparar o conteúdo de elementos em um documento XML.

Operações com intervalos de tempo e datas. Processamento de conteúdos que envolvem intervalos de tempo e datas, estão disponíveis através de funções como *yearMonthDuration-less-than(), yearMonthDuration-greater-than(), dayTimeDuration-less-than(), dayTimeDuration-greater-than(), duration-equal(), years-from-duration(), months-from-duration(), days-from-duration(), hours-from-duration(), minutes-from-duration(), seconds-*

from-duration(), dateTime-equal(), dateTime-less-than(), dateTime-greater-than(), date-equal(), etc..

Operações com *QNames*. Operações relativas à *qualified names* estão disponíveis através de funções com *QName-equal(), prefix-from-QName(), local-name-from-QName(), namespace-uri-from-QName(), namespace-uri-for-prefix().*

Operações envolvendo valores base64 e hexadecimais. Comparações de valores em binário no formato base64 e hexadecimal binário podem ser aplicadas pelas funções *hexBinary-equal()* e *base64Binary-equal().*

Operações imediatas sobre um elemento. Diversas funções podem ser aplicadas diretamente sobre um determinado elemento do documento XML. Estas funções podem retornar o nome do elemento com ou sem o respectivo *namespace* (*local-name(), name()*), seu contexto de linguaguem (*lang()*), ou ainda realizar verificações como *node-before(), node-after(), root(), has-children(),* etc..

Operações com coleções. Uma coleção de elementos ou valores pode ser manipulada por funções como *head(), tail(), remove(), reverse(),* etc..

Operações relativas ao contexto. Funções relativas ao contexto posicional estão presentes como *position()* e *last(),* assim como contextos temporais (*current-dateTime(), current-date(), current-time(), implicit-timezone()*) e realtivos a URI (*static-base-uri()*).

2.3 CONSIDERAÇÕES FINAIS

Este capítulo apresenta brevemente as principais características da linguagem de consulta XPath. A XPath fornece alto poder de expressão ao disponibilizar uma gama de funções, operadores e possibilidades de filtros para realização de consultas sobre documentos XML. Conforme já mencionado, as abordagens que traduzem consultas XPath para Prolog sofrem com a falta de recursos relativos à expressividade no momento da criação das consultas. Desta forma, o próximo capítulo aborda com maiores detalhes os trabalhos relacionados, destacando suas principais características.

Capítulo 3 -TRABALHOS RELACIONADOS

A utilização de linguagens de programação lógica no âmbito de consultas a bases de dados semiestruturadas é um tema pesquisado há alguns anos. O aumento da oferta de informações, sobretudo dentro do contexto *Web*, fez crescer o interesse acerca da otimização do processo de obtenção de informações deste tipo, especialmente utilizando documentos XML. De fato, diversos trabalhos utilizam linguagens lógicas para manipular documentos XML.

May (2001) desenvolveu uma linguagem de manipulação de documentos XML nomeada de XPathLog. Similar à linguagem Prolog em termos de utilização, a XPathLog possibilita a realização de consultas, criação de novos elementos, ou a formação de ciclos através da adição de *links* entre elementos contidos no documento XML. Ronem e Shmueli (2007) utilizam Datalog para transformar documentos XML em termos e representar sua estrutura contida em DTDs, no que o autor chama de *regular types*. XCentric (COELHO; FLORIDO, 2007) é uma abordagem em Prolog que transforma dados semiestruturados em termos e possibilita a realização de consultas do tipo *pattern queries*. Schatten e Ivkovic (2012) propuseram uma linguagem semelhante à XPath (*RPE- Regular Path Expressions*) utilizando Prolog na definição da gramática e na representação de dados semiestruturados, como por exemplo XML e JSON. Apesar destas abordagens basearem-se na linguagem de consulta XPath, é importante mencionar que sua utilização impõe uma curva de aprendizado do usuário. A ausência de um algoritmo de tradução XPath não permite uma experiência totalmente transparente ao usuário.

A utilização em conjunto das linguagens XPath e Prolog pôde ser analisada em alguns trabalhos. O pacote *SWI-Prolog SGML/XML parser* (WIELEMAKER, 2009), presente na ferramenta SWI-Prolog (JAN WIELEMAKER *et al.*, 2012), implementa funcionalidades de *parser* de arquivos SGML, XML e HTML, além da execução de algumas consultas básicas XPath. A tradução dos arquivos é representada em forma de listas conforme exibe a Figura 9. As consultas XPath devem ser imputadas por meio de um predicado Prolog de assinatura *xpath(+DOM, +Spec, ?Content),* aonde são passados como argumento a lista que representa o documento (*DOM*), a consulta XPath (*Spec*), e a variável que armazenará o resultado (*Content*). Segundo sua documentação, a utilização de filtros deve ser expressa através de parêntesis ao invés de colchetes, como por exemplo, na consulta *xpath(DOM, //book(@genre=thiller), Book).*

Documento HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
 <head>
  <title>Demo</title>
 </head>
 <body>
  <h1 align=center>This is a demo</h1>
  Paragraphs in HTML need not be closed.
  This is called `omitted-tag' handling.
 </body>
</html>
```

Tradução Prolog:

```
[ element(html,
        [],
        [ element(head,
                [],
                [ element(title,
                        [],
                        [ 'Demo'
                        ])
                ]),
          element(body,
                [],
                [ '\n',
                  element(h1,
                        [ align = center
                        ],
                        [ 'This is a demo'
                        ]),
                  '\n\n',
                  element(p,
                        [],
                        [ 'Paragraphs in HTML need not be closed.\n'
                        ]),
                  element(p,
                        [],
                        [ 'This is called `omitted-tag\' handling.'
                        ])
                ]) ]) ].
```

Fonte: (WIELEMAKER, 2009)

Figura 9: Documento HTML e sua respectiva tradução Prolog, realizada pelo pacote *SWI-Prolog SGML/XML parser*

Entretando a abordagem que mais possui características similares às deste trabalho é a de Almendros-Jiménez *et al.* (2008). Neste trabalho os autores também propõem um processo de transformação de documentos XML em base de fatos conectados por identificadores, além da utilização de um esquema (obtido pela leitura do próprio documento) para criação de regras que serão utilizadas durante a execução de consultas XPath. Apesar dos trabalhos possuírem semelhanças, a execução da proposta de Almendros-Jiménez *et al.* (2008) se dá de maneira diferente, conforme detalhado a seguir.

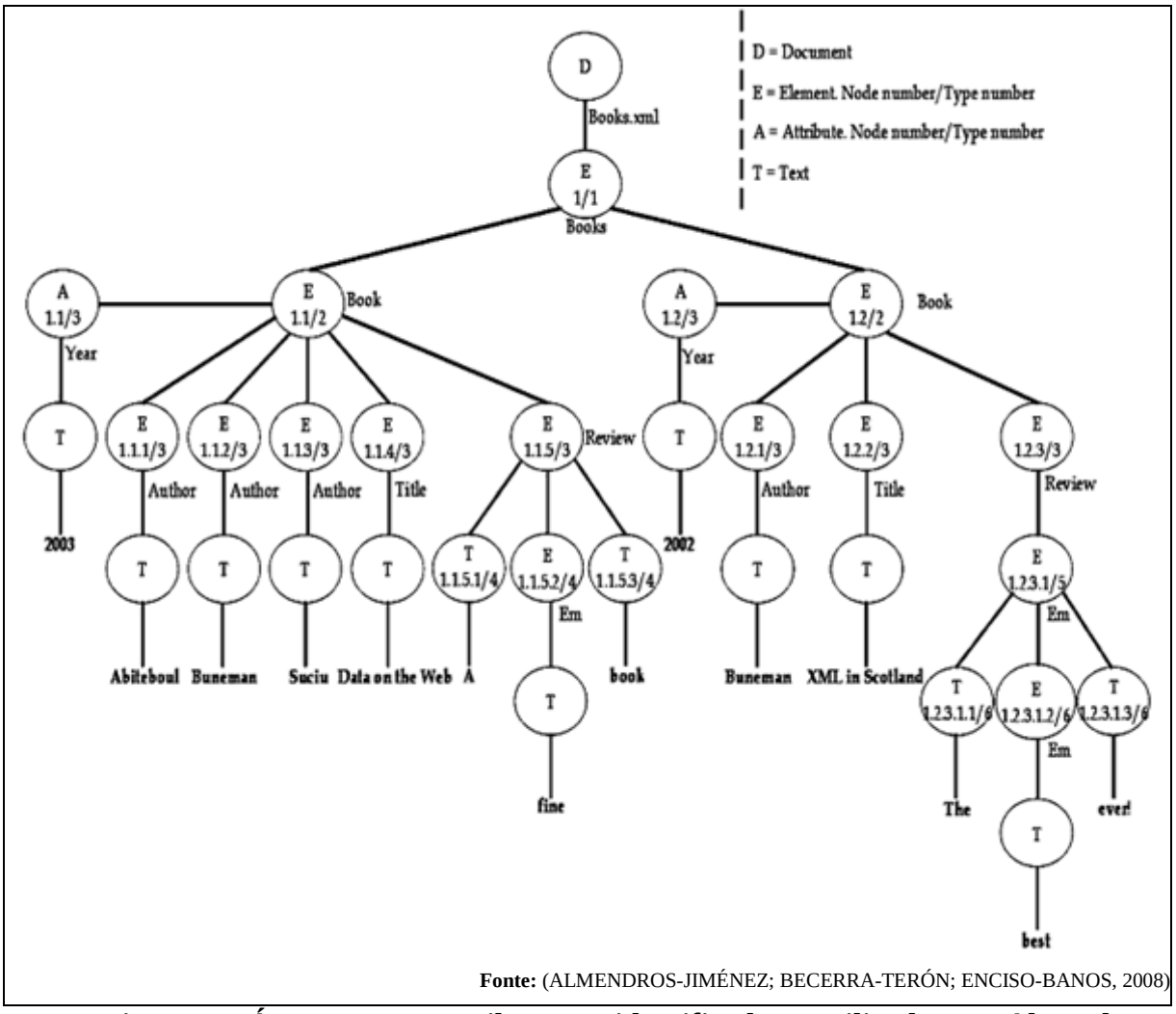
A tradução do documento XML realizada por Almendros-Jiménez *et al.* (2008), considera o uso de identificadores de nome *nodenumber* e *typenumber*. O identificador *nodenumber* representa o identificador do nó a ser traduzido, iniciando com o número 1 para o elemento raiz. O identificador *nodenumber* dos elementos descendentes do nó raiz é descrito como o valor do *nodenumber* pai acrescido de um ponto, que representa uma mudança de nível, e o valor numérico que indica a ordem de aparição no documento. Já o identificador *typenumber* auxilia na distinção da ocorrência de elementos com mesmo nome, mas estruturalmente diferentes. O valor do *typenumber* é composto por $l + n + 1$ onde l é o *typenumber* do elemento pai e n é o número de elementos de mesmo nome que estão à esquerda do elemento em questão, nomeados pelos autores de fracamente distintos. A Figura 10 exibe o documento XML utilizado nos exemplos do trabalho de Almendros-Jiménez *et al.* (2008) e a Figura 11 exibe a árvore XML que ilustra os identificadores utilizados por sua abordagem.

```
<books>
  <book year="2003">
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <title>Data on the Web</title>
    <review>A<em>fine</em> book.</review>
  </book>
  <book year="2002">
    <author>Buneman</author>
    <title>XML in Scotland</title>
    <review><em>The <em>best</em> ever!</em></review>
  </book>
</books>
```

Fonte: (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008)

Figura 10: Documento XML de exemplo utilizado por Almendros-Jiménez *et al.* (2008)

Conforme pode ser observado na Figura 12, a tradução do documento XML em fatos contempla os identificadores *nodenumber* (em formato de lista) e *typenumber* descritos anteriormente. Para as ocorrências de valores dos elementos mistos os autores utilizam o termo *unlabeled*. A criação das respectivas regras ocorre de maneira encadeada, de forma a facilitar a obtenção dos resultados das consultas. Conforme ilustra a Figura 13, cada regra de elementos não terminais é construída segundo a seguinte representação:



Fonte: (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008)

Figura 11: Árvore XML que ilustra os identificadores utilizados por Almendros-Jiménez *et al*. (2008)

$tag(tagtype(Tag_{i1},...,Tag_{it}, [Att_1 ,...,Att_n]),NodeTag, k):-$

$tag_{i1}(Tag_{i1}, [NodeTag_{i1}|NodeTag], r),...,$

$tagit(Tag_{it}, [NodeTag_{it}|NodeTag], r),$

$att1 (Att_1 ,NodeTag, r),...,$

$attn (Att_n ,NodeTag, r).$

onde:

- *tag* é o nome do elemento corrente;
- *tagtype* é o nome da função que irá recuperar os valores do documento XML;
- $\{tag_{ij} \mid ij \in \{1,...,s\}, 1 \leq j \leq t\}$ é o conjunto de *tags* dos elementos filhos do elemento corrente;
- $att_1,...,att_n$ representam os nomes dos atributos;
- $Tag_{i1},...,Tag_{it}$ e $Att_1 ,...,At_n$ são variáveis, uma para cada *tag*/atributo;

- $NodeTag_{i1},...,NodeTag_{it}$ são variáveis para representar o *nodenumber* do elemento pai;
- *NodeTag* é a variável que representa a parte final do *nodenumber* corrente;
- *k* é o *typenumber* corrente;
- *r* é *typenumber* dos elementos filhos.

```
year('2003', [1, 1], 3).
author('Abiteboul', [1, 1, 1], 3).
author('Buneman', [2, 1, 1], 3).
author('Suciu', [3, 1, 1], 3).
title('Data on the Web', [4, 1, 1], 3).
unlabeled('A', [1, 5, 1, 1], 4).
em('fine', [2, 5, 1, 1], 4).
unlabeled('book.', [3, 5, 1, 1], 4).
year('2002', [2, 1], 3).
author('Buneman'. [1, 2, 1], 3).
title('XML in Scotland', [2, 2, 1], 3).
unlabeled('The', [1, 1, 3, 2, 1], 6).
em('best', [2, 1, 3, 2, 1], 6).
unlabeled('ever!', [3, 1, 3, 2, 1], 6).
```

Fonte: (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008)

Figura 12: Fatos Prolog traduzidos pela abordagem de Almendros-Jiménez *et al*. (2008)

```
books( bookstype( Books, []), NodeBooks,1 ) :-
        book( Books, [NodeBook|NodeBooks] ,2).
book( booktype( Author, Title, Review, [Year]), NodeBook ,2) :-
        author( Author, [NodeAuthor|NodeBook] ,3),
        title(Title, [NodeTitle|NodeBook] ,3),
        review( Review, [NodeReview|NodeBook] ,3),
        year( Year, NodeBook,3).
review( reviewtype( Unlabeled,Em,[]),NodeReview,3):-
        unlabeled( Unlabeled,[NodeUnlabeled|NodeReview],4),
        em( Em,[NodeEm|NodeReview] ,4).
Review( reviewtype( Em,[]),NodeReview,3) :-
        em( Em,[NodeEm|NodeReview],5).
 em( emtype( Unlabeled, Em,[]),NodeEms,5) :-
        unlabeled( Unlabeled,[NodeUnlabeled|NodeEms],6),
        em( Em, [NodeEm|NodeEms],6).
```

Fonte: (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008)

Figura 13: Regras Prolog criadas pela abordagem de Almendros-Jiménez *et al*. (2008)

As consultas XPath executadas pela abordagem de Almendros-Jiménez *et al.* (2008) utilizam o conjunto de regras criadas anteriormente, armazenadas na memória principal, e a base de fatos armazenada em disco. Um subconjunto de regras é eleito para solucionar cada consulta XPath, nomeado pelos autores de especialização do esquema. Índices e reordenação do conjunto de regras são utilizados para aperfeiçoar o processo de obtenção dos resultados. A

Figura 14 exibe uma consulta XPath aonde há a ocorrência de filtros utilizando o atributo *year* e o elemento *title*. A técnica de Almendros-Jiménez *et al.* (2008) reordena as regras que compõem a solução da consulta, de forma que as regras que contêm os elementos filtrados sejam executadas primeiro.

```
/book [@year = 2002 and title =“DataontheWeb”]/author
```

```
book(booktype(Author,Title,Review,[Year]),NodeBook,2):-
    year(Year,NodeBook,3),
    title(Title,[NodeTitle|NodeBook],3),
    author(Author,[NodeAuthor|NodeBook],3).
```

Fonte: (ALMENDROS-JIMÉNEZ; BECERRA-TERÓN; ENCISO-BANOS, 2008)

Figura 14: Exemplo de reordenação de regras para um consulta XPath, por Almendros-Jiménez *et al*. (2008)

A abordagem de Almendros-Jiménez *et al.* (2008) consegue traduzir somente consultas com operadores “//”, “/../”, “*” e filtros básicos. Seus respectivos experimentos consideraram consultas relativamente simples e documentos de tamanho máximo de 1MB. Os tempos de resposta considerando as etapas de tradução, execução, reconstrução dos resultados e etc., são ruins, considerando a baixa complexidade das consultas e tamanho dos documentos utilizados. A consulta */books* para um documento de 512 KB, por exemplo, é executada em 2 minutos e 54 segundos. Além disso, não há comparação do desempenho da abordagem proposta com outras abordagens de processamento de consultas XML. Posteriormente Almendros-Jiménez (2009) substituiu a abordagem anterior de tradução de documentos XML e consultas XPath pelo já citado pacote *SWI-Prolog SGML/XML parser* (WIELEMAKER, 2009), focando desta vez em consultas XQuery.

Já os trabalhos de Almendros-Jiménez *et al.* (2011; 2014) tratam da introdução de lógica *fuzzy* em consultas XPath, introduzindo novos operadores que permitem consultas mais flexíveis. Almendros-Jiménez *et al.* (2012) apresenta a linguagem lógica TOY para consulta a documentos XML. As consultas realizadas pela TOY necessitam que o documento XML tenha sido previamente convertido para uma representação própria da linguagem. Ainda que semelhantes, as consultas suportadas pela linguagem TOY exigem que as consultas XPath originais sejam adaptadas, o que faz com que a abordagem não seja totalmente transparente para o usuário.

Lima *et al.* (2012) lidou com o tema de obtenção de informações de documentos XML utilizando bases de fatos Prolog. Em uma avaliação *ad-hoc*, os resultados obtidos pelos autores foram satisfatórios quando comparados com outros processadores de consulta XML nativos. Ainda sobre o arcabouço construído por Lima *et al*. (2012), Santos *et al.* (2012) desenvolveu um algoritmo que possibilitava executar consultas XPath simples, constituídas

dos operadores “/”,”//” e filtros básicos. As consultas eram traduzidas para Prolog e executadas sobre uma base fatos. Santos (2015) também utiliza a abordagem de Lima *et al.* (2012) durante sua análise acerca da performance de consultas XQuery e Prolog sobre documentos XML. Os resultados obtidos por Santos (2015) mostram a viabilidade de utilização da linguagem Prolog para execução de consultas sobre documentos XML.

As abordagens de Lima *et al.* (2012) e Santos (2015) não utilizaram nenhuma técnica de tradução automática para as consultas. Além disso, a abordagem de Santos *et al.* (2012) ainda que funcional, considera apenas um pequeno subconjunto da linguagem XPath.

3.1 CONSIDERAÇÕES FINAIS

De forma geral, a ausência de uma abordagem que permitisse uma maior expressividade na confecção de consultas XPath (automaticamente traduzidas para Prolog), além da respectiva obtenção automática de resultados em formato XML, e a ausência de experimentos com documentos de grande porte, incentivaram o desenvolvimento deste trabalho.

Diferentemente de outras abordagens relacionadas, este trabalho não utiliza linguagem de consulta própria. A consulta de informações sobre as bases de dados semiestruturadas pode ser realizada na linguagem XPath, bastante difundida. Desta forma o usuário é poupado da tarefa de aprendizado e adataptação à ferramenta. Outro diferencial é a possibilidade acerca da utilização de documentos XML de grandes tamanhos, sem a necessidade de configuração de índices e estruturas de apoio.

Como os trabalhos de Lima *et al.* (2012) e seus respectivos sucessores serviram de arcabouços para esta abordagem, o próximo capítulo detalha o XMLInference, nome dado por Lima *et al.* (2012) ao seu trabalho.

Capítulo 4 - XMLINFERENCE

4.1 INTRODUÇÃO

A abordagem proposta por Lima *et al.* (2012), chamada de XMLInference, obteve bons tempos de resposta em consultas (traduzidas manualmente) executadas sobre bases de fatos Prolog. Por isso, ela foi escolhida para ser usada como base do trabalho proposto nessa dissertação. Basicamente, a proposta original do XMLInference possibilita ao usuário realizar consultas Prolog em bases de fatos que foram previamente convertidas pela abordagem, a partir de documentos XML. Uma característica diferencial da técnica XMLInference é a possibilidade de inferir conhecimento por meio do uso de regras automáticas ou manuais, inseridas previamente por um usuário especialista. Nesse capítulo, detalhamos a abordagem de tradução de documentos XML para fatos Prolog, uma vez que esse conhecimento será necessário para entender o algoritmo de tradução de consultas proposto nessa dissertação.

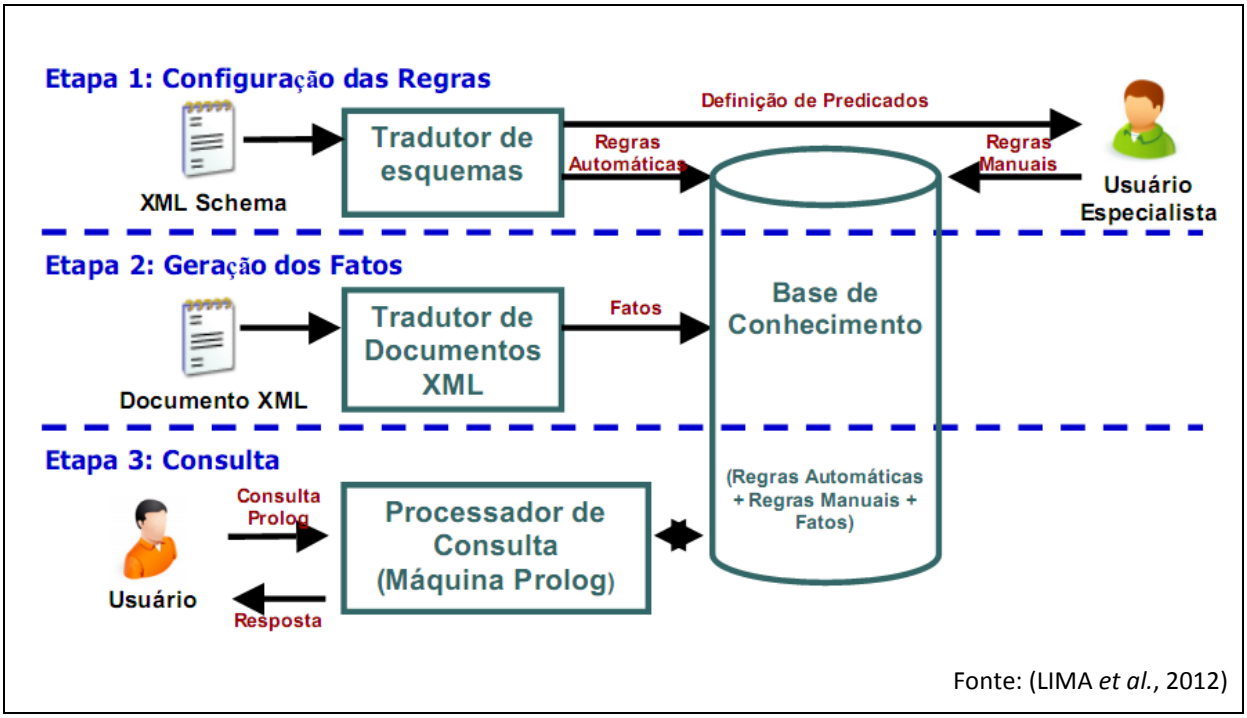


Figura 15: Etapas da abordagem XMLInference de Lima *et al.* (2012)

A técnica XMLInference pode ser subdividida em três etapas principais. A primeira consiste na tradução do XML Schema do documento em regras automáticas Prolog. Ainda nesta etapa o usuário que detém conhecimento acerca do esquema pode introduzir regras manuais que possam agregar conhecimento ao usuário final. A segunda etapa realiza o processo de tradução do documento XML para uma base de fatos Prolog. Por fim, a terceira

etapa é constituída na realização de consultas em linguagem Prolog pelo usuário final. As consultas são realizadas sobre a base de conhecimento, que contém as regras automáticas e manuais, além dos fatos. A Figura 15 resume as etapas descritas anteriormente.

As seções seguintes detalham as duas primeiras etapas da abordagem XMLInference. Para melhor compreensão, a tradução de documentos XML é abordada antes do processo de criação de regras automáticas e manuais.

4.2 TRADUÇÃO DE DOCUMENTOS XML PARA PROLOG

A tradução de documentos XML para bases de fatos Prolog consiste na geração de diversos fatos que correspondem aos elementos presentes no documento. Cada elemento é transformado em um predicado Prolog, com seu respectivo conteúdo adicionado como uma constante. Para representar as relações de parentesco entre os elementos do documento XML, a técnica de tradução criou um mecanismo de interligação entre os fatos. Identificadores foram adicionados ao fato para representar a relação de pai e filho entre os elementos. A versão inicial do algoritmo de tradução de Lima *et al*. (2012) concebia identificadores próprios somente para os fatos que correspondiam a elementos que possuíam filhos no documento. Posteriormente, Santos *et al.* (2015) modificou o algoritmo para contemplar um identificador próprio para todo fato criado. Esta modificação permitiu a ordenação dos fatos que representam elementos folha do documento, preservando assim sua ordem original. O processo de tradução de documentos XML pode ser subdividido entre cinco possíveis tipos de tradução, apresentados a seguir. Os exemplos deste capítulo utilizam o mesmo documento XML e respectivo esquema do Capítulo 2.

Tradução do elemento raiz

O elemento raiz possui um tratamento especial no momento de sua tradução. Como é o único que não possui um elemento pai, seu respectivo fato possui apenas um único identificador. Este *id* possui o papel de realizar a ligação do elemento raiz com seus filhos. A Figura 16 exibe o fato correspondente ao elemento raiz *dblp*.

```
dblp(1).
```

Figura 16: Fato correspondente ao elemento raiz *dblp*

Tradução de elementos complexos

Elementos complexos possuem outros elementos como filhos. Sua respectiva tradução é constituída do nome do elemento, o identificador do elemento pai, além do seu próprio identificador. Seus elementos filhos serão traduzidos de acordo com sua classificação (simples ou complexos), utilizando o valor do identificador do elemento complexo pai em sua formação. A Figura 17 mostra a tradução do elemento complexo *article*, presente na linha 2 do documento XML do Capítulo 2.

```
article(1, 2).
```

Figura 17: Tradução do elemento complexo *article*

Tradução de elementos folha

Elementos folha sem atributos possuem somente seu respectivo valor como filho na árvore XML. Sua tradução é constituída do nome do elemento, o identificador do elemento pai, seu próprio identificador para fins de ordenação, e o valor correspondente. A Figura 18 exibe um exemplo de tradução do elemento simples sem atributos *title*, presente na linha 3 do documento XML do Capítulo 2.

```
title(2, 5, 'XPath Translation').
```

Figura 18: Tradução de elemento simples e sem atributos *title*

Tradução de elementos com atributos

Na tradução proposta por Lima *et al*. (2012), atributos são considerados filhos dos elementos que os contém. Desta forma a tradução de fatos para atributos é representada pelo seu respectivo nome, o identificador do elemento pai, seu próprio identificador e o valor correspondente. Apesar dos atributos contidos em um elemento não possuírem ordem, o identificador do atributo pode ser utilizado para manter a ordem original em que o atributo aparece no documento XML. A Figura 19 apresenta um exemplo de tradução para o atributo *mdate*, presente na linha 21 do documento XML do Capítulo 2.

```
mdate(26, 29, '2016-04-01').
```

Figura 19: Exemplo de tradução do atributo *mdate*

Tradução de elementos mistos

A tradução de elementos mistos é semelhante às demais traduções já apresentadas. A diferença principal é que o nome do predicado Prolog gerado é igual para todos dos elementos mistos. A nomenclatura não compromete os resultados já que os identificadores é que realizam o papel de informar o posicionamento correto do conteúdo. Sua formação é composta pelo termo *xml/mixedElement*, o identificador do elemento pai, seu próprio identificador e o conteúdo que representa. A Figura 20 exibe um exemplo da tradução de um elemento misto (linha 12 do documento XML).

```
xml/mixedElement(12, 13, 'co-author').
```

Figura 20: Exemplo de tradução de um elemento misto

4.3 CRIAÇÃO DE REGRAS PROLOG AUTOMÁTICAS E MANUAIS

A criação de regras Prolog permite a realização de inferências acerca do documento a ser traduzido. Na abordagem de Lima *et al*. (2012), estas regras são subdivididas em automáticas e manuais. As regras automáticas, geradas sem supervisão, são construídas durante o processo de *parse* do XML Schema referente ao documento desejado. Já as regras manuais podem ser inseridas por algum usuário que possua conhecimento específico da estrutura do documento. Os dois processos são detalhados a seguir.

4.3.1 REGRAS AUTOMÁTICAS

Durante o processo de criação das regras automáticas o algoritmo correspondente constrói diferentes regras em função da definição estrutural de cada elemento do esquema, considerando os delimitadores de grupo *sequence*, *choice* e *all* (FALLSIDE; WALMSLEY, 2004).

Regras para elementos complexos com elementos filhos simples

A criação de regras para elementos com filhos não complexos é a mais simples de ser realizada, pois utiliza as traduções básicas para cada delimitador de grupo já citado.

Regras para *sequence*: O delimitador de grupo *sequence* especifica a ordem e cardinalidade dos elementos contidos no mesmo. A regra do elemento em questão é constituída por uma variável correspondente ao seu identificador, seguidos de variáveis correspondentes ao conteúdo de cada elemento filho que possui. A Figura 21 exibe um exemplo correspondente de regra para o elemento *author*.

Esquema XML:

```
1 <xs:complexType name="tAuthor" mixed="true">
2   <xs:sequence >
3     <xs:element name="name"  type="xs:string"/>
4     <xs:element name="email" type="xs:string"/>
5   </xs:sequence>
6 </xs:complexType>
```

Regra Prolog:

```
author(ID, NAME, EMAIL):- name(ID, IDNAME, NAME), email(ID, IDEMAIL, EMAIL).
```

Figura 21: Criação de regras para o delimitador *sequence*

Regras *choice*: O delimitador de grupo *choice* especifica que na composição do elemento em questão, poderá haver somente a ocorrência de um dos elementos filhos por vez. Desta forma o algoritmo de criação de regras automáticas constrói regras distintas considerando cada elemento dentro do delimitador *choice*. O diferencial destas regras é que as mesmas contam com um identificador de opção que possibilita informar a qual elemento pertence o valor retornado. A Figura 22 contém um exemplo do esquema que descreve o elemento *proceedings*, e as regras criadas para o delimitador *choice*, linhas 6 a 9, com delimitadores de opção *name* e *address*. Durante a criação de regras automáticas, Lima *et al.* (2012) desconsideram delimitadores de grupo que contenham o atributo *minOccurs=0*, pelo motivo de não saber se há de fato a ocorrência destes elementos no documento. Os atributos relativos à cardinalidade foram removidos do trecho do XML Schema da Figura 22 para manter a coerência com a técnica de criação de regras original.

Regras *all*: O delimitador de grupo *all* permite que o elemento complexo em questão possua como filho qualquer combinação dos elementos dentro do delimitador, e em qualquer ordem. O delimitador *all* ainda prevê que todos os elementos dentro do grupo devam ser elementos simples. A construção da regra do elemento complexo considera uma variável para sua identificação, um identificador de opção para o elemento filho correspondente, seguido de outra variável para o conteúdo. A Figura 23 exibe um exemplo de regras para o delimitador *all*.

Como exemplo de utilização das regras da Figura 23, se utilizarmos o documento de exemplo da, a consulta *book(_, FILHO, CONTEUDO)*, retornará os seguintes resultados:

FILHO= *authorName*, CONTEUDO= '*Semistructured Data Author*'

FILHO= *title*, CONTEUDO= '*Semistructured Data*'

FILHO= *year*, CONTEUDO= '2016'

Esquema XML:

```
1  <xs:complexType name="tProceedings">
2     <xs:sequence>
3        <xs:element name="booktitle" type="xs:string"/>
4        <xs:element name="authorname" type="xs:string"/>
5        <xs:element name="title" type="xs:string"/>
6        <xs:choice>
7           <xs:element name="year" type="xs:integer"/>
8           <xs:element name="address" type="xs:string"/>
9        </xs:choice>
10    </xs:sequence>
11    <xs:attribute name="key" type="xs:string" use="required"/>
12    <xs:attribute name="mdate" type="xs:string" use="optional"/>
13 </xs:complexType>
```

Regras Prolog:

```
proceedings(ID, name, NAME) :- name(ID, IDNAME, NAME).

proceedings(ID, address, ADDRESS) :- name(ID, IDADDRESS, ADDRESS).
```

Figura 22: Criação de regras para delimitador *choice*

Esquema XML:

```
1 <xs:complexType name="tBook">
2  <xs:all>
3   <xs:element name="authorName" type="xs:string"/>
4   <xs:element name="title" type=" xs:string"/>
5   <xs:element name="year" type=" xs:integer"/>
6  </xs:all>
7 </xs:complexType>
```

Regras Prolog:

```
book(ID, authorName, AUTHORNAME):- authorName(ID, IDAUTHORNAME, AUTHORNAME).
book(ID, title, TITLE):- title(ID, IDTITLE, TITLE).
book(ID, year, YEAR):- title(ID, IDYEAR, YEAR).
```

Figura 23: Criação de regras para delimitador *all*

Regras para elementos complexos com filhos complexos

A criação de regras para elementos com filhos complexos segue os mesmos conceitos descritos anteriormente. Porém, como os elementos filhos também são complexos, é necessário realizar alguns tratamentos especiais para que as regras geradas sejam válidas.

Regras *sequence* com filho *choice*: A criação de regras para elementos que possuam o delimitador *sequence* com filhos *choice* corresponde à união das regras de ambos os grupos. A Figura 24 mostra que o elemento *www* possui 3 filhos, sendo os 2 primeiros *authorName* e *url*, e o último variando entre *title* e *year*. Desta forma serão criadas duas regras distintas para

cada opção. Semelhantemente à criação da regra anterior para o delimitador *choice*, um identificador de opção é adicionado.

Esquema XML:

```
1  <xs:complexType name="wwwType">
2    <xs:sequence>
3      <xs:element name="authorName" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
4      <xs:element name="url" type="xs:string" />
5      <xs:choice>
6        <xs:element ref="title" type="xs:string"/>
7        <xs:element ref="year" type="xs:integer" />
8      </xs:choice>
9    </xs:sequence>
10   <xs:attribute name="key" type="xs:string" use="required"/>
11 </xs:complexType>
```

Regras Prolog:

```
www(ID, title, AUTHORNAME, URL, TITLE):- title(ID, IDTITLE, TITLE),
                          authorName(ID, IDAUTHORNAME, AUTHORNAME),
                          url(ID, IDURL, URL).
www(ID, year, AUTHORNAME, URL, YEAR):- year(ID, IDYEAR, YEAR),
                           authorName(ID, IDAUTHORNAME, AUTHORNAME),
                           url(ID, IDURL, URL).
```

Figura 24: Criação de regras para delimitador *sequence* com filho *choice*

Regras *choice* com filho *sequence*: A criação de regras para elementos que possuam o delimitador *choice* com filhos *sequence* utiliza identificadores para cada ocorrência do delimitador *sequence*. A Figura 25 apresenta um exemplo deste tipo de tradução. Já que não há elementos que possam ser utilizados como identificadores de opção, identificadores sintéticos (“s1” e “s2”) são inseridos para diferenciar cada conjunto *sequence*.

Propriedade do elemento neutro

De acordo com a abordagem de Lima *et al.* (2012), outras combinações tais como *sequences* com filhos *sequence*, e *choice* com filhos *choice* são resolvidas com a propriedade do elemento neutro. A propriedade considera que os casos citados acima podem ser considerados como um único *sequence* ou *choice*. A Figura 26 exibe um exemplo de uma definição alternativa do elemento raiz *dblp*, utilizando *sequences* aninhados, e de como os mesmos podem ser considerados como um único *sequence* durante o processo de criação de regras. De forma semelhante, a Figura 27 ilustra um exemplo de uma nova definição do elemento *www* com *choices* aninhados e sua respectiva simplificação.

Esquema XML:

```
1 <xs:complexType name="mastersthesisType" >
2   <xs:choice>
3     <xs:sequence>
4       <xs:element name="authorName" type="xs:string"/>
5       <xs:element name="title" type="xs:string"/>
6       <xs:element name="year" type="xs:integer"/>
7     </xs:sequence>
8     <xs:sequence>
9       <xs:element name="researchArea" type="xs:string"/>
10       <xs:element name="authorName" type="xs:string"/>
11       <xs:element name="title" type="xs:string"/>
12       <xs:element name="school" type="xs:string"/>
13     </xs:sequence>
14   </xs:choice>
15   <xs:attribute name="key" type="xs:string" use="required"/>
16 </xs:complexType>
```

Regras Prolog:

```
mastersthesis(ID, s1, AUTHORNAME, TITLE, YEAR):- authorname(ID, IDAUTHORNAME, AUTHORNAME),
                                                 title(ID, IDTITLE, TITLE),
                                                 year(ID, IDYEAR, YEAR).
mastersthesis(ID, s2, RESEARCHAREA, AUTHORNAME, TITLE, SCHOOL):-
                                                 researchArea(ID, IDRESEARCHAREA, RESEARCHAREA),
                                                 authorname(ID, IDAUTHORNAME, AUTHORNAME),
                                                 title(ID, IDTITLE, TITLE),
                                                 school (ID, IDSCHOOL, SCHOOL).
```

Figura 25: Criação de regras para delimitador *choice* com filho *sequence*

Esquema XML (*sequences* aninhados):

```
1 <xs:complexType  name="tDblp">
2   <xs:sequence>
3     <xs:sequence>
4       <xs:element name="article" type="tArticle"/>
5       <xs:element name="proceedings" type="tProceedings"/>
6     </xs:sequence>
7     <xs:element name="book" type="tBook"/>
8     <xs:element name="mastersthesis" type ="mastersthesisType" >
9     <xs:element name="phdthesis" type="phdthesisType"/>
10     <xs:element name="www" type="wwwType">
11   </xs:sequence>
12 </xs:complexType>
```

Esquema XML (apenas 1 *sequence*):

```
1 <xs:complexType  name="tDblp" >
2   <xs:sequence>
3     <xs:element name="article" type="tArticle"/>
4     <xs:element name="proceedings" type="tProceedings"/>
5     <xs:element name="book" type="tBook"/>
6     <xs:element name="mastersthesis" type ="mastersthesisType" >
7     <xs:element name="phdthesis" type="phdthesisType"/>
8     <xs:element name="www" type="wwwType">
9   </xs:sequence>
10 </xs:complexType>
```

Figura 26: Simplificação de *sequences* aninhados

Esquema XML (choices aninhados):

```
1 <xs:complexType name="wwwType">
2   <xs:choice>
3     <xs:element name="authorName" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
4     <xs:element name="url" type="xs:string" />
5     <xs:choice>
6       <xs:element name="title" type="xs:string"/>
7       <xs:element name="year" type="ixs:nteger" />
8     </xs:choice>
9   </xs:choice>
10  <xs:attribute name="key" type="xs:string" use="required"/>
11 </xs:complexType>
```

Esquema XML (apenas 1 *choice*):

```
1 <xs:complexType name="wwwType">
2   <xs:choice>
3     <xs:element name="authorName" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
4     <xs:element name="url" type="xs:string" />
5     <xs:element name="title" type="xs:string"/>
6     <xs:element name="year" type="ixs:nteger" />
7   </xs:choice>
8   <xs:attribute name="key" type="xs:string" use="required"/>
9 </xs:complexType>
```

Figura 27: Simplificação de *choices* aninhados

4.3.2 REGRAS MANUAIS

Conforme mencionado anteriormente, as regras manuais são inseridas por usuários especialistas do documento a ser consultado. A inserção de regras manuais permite realizar inferências que muitas vezes não são possíveis somente com a utilização de regras automáticas. Outra característica é a possibilidade de produzir meta-informações que agreguem conhecimento ao usuário. Dado o esquema do Capítulo 2, podemos considerar a obtenção de informações a respeito de pesquisadores que publicaram um artigo no mesmo ano. Tal informação não é possível de ser obtida consultando somente as regras automáticas criadas pelo algoritmo. Desta forma, um usuário especialista do documento pode implementar tal regra, que poderá ser utilizada posteriormente por outros usuários. A Figura 28 exibe a regra *sameYearPublication*, que informa se dois pesquisadores publicaram algum artigo no mesmo ano.

```
sameYearPublication(AUTHOR1, AUTHOR2, YEAR) :- year(IDARTICLE, _, YEAR),
        authors(IDARTICLE, IDAUTHORS), author(IDAUTHORS, IDAUTHOR),
        name(IDAUTHOR, _, AUTHOR1), year(IDARTICLE2, _, YEAR),
        authors(IDARTICLE2, IDAUTHORS2), author(IDAUTHORS2, IDAUTHOR2),
        name(IDAUTHOR2, _, AUTHOR2).
```

Figura 28: Regra manual *sameYearPublication*

4.4 CONSIDERAÇÕES FINAIS

Este capítulo descreve o XMLInference, apresentando o processo de tradução de documentos XML e a criação de regras manuais e automáticas. Os bons resultados alcançados por Lima *et al.* (2012), e outros trabalhos que também utilizaram suas técnicas (SANTOS; PINHEIRO; BRAGANHOLO, 2012; SANTOS, 2015), motivaram a utilização de sua abordagem como arcabouço para o desenvolvimento deste trabalho de tradução automática de consultas XPath para Prolog. Desta forma, o próximo capítulo trata das melhorias realizadas ao longo da pesquisa na abordagem XMLInference.

Capítulo 5 – MELHORIAS NA ABORDAGEM XMLINFERENCE

5.1 INTRODUÇÃO

Durante o decorrer da pesquisa, modificações foram realizadas na abordagem escolhida para desenvolver este trabalho – XMLInference (LIMA *et al.*, 2012) – com o objetivo de evoluir ou mesmo corrigir alguns pontos. A Figura 29 exibe em alto nível, a nova arquitetura da abordagem XMLInference para realização de consultas XPath. A etapa 1, que anteriormente travava da criação de regras automáticas e manuais, passou a ser responsável pela criação das regras de impressão, que geram os resultados em formato nativo XML, além de outras regras e predicados auxiliares.

A etapa 2 continua sendo responsável pela geração dos fatos, porém através de um novo algoritmo que possui um desempenho superior ao da versão anterior. Na etapa 3 reside o foco deste trabalho. Após o usuário submeter uma consulta XPath, o algoritmo de tradução de consultas realiza as operações de *parsing* e construção da consulta Prolog equivalente. Após a construção, o algoritmo executa a respectiva consulta Prolog na base de conhecimento, exibindo o resultado no formato nativo XML.

Toda esta nova abordagem recebeu o nome de XP2PL (abreviatura de XPath2Prolog). Ao longo deste capítulo são discutidos maiores detalhes sobre as modificações realizadas, assim como algumas situações encontradas e suas respectivas soluções.

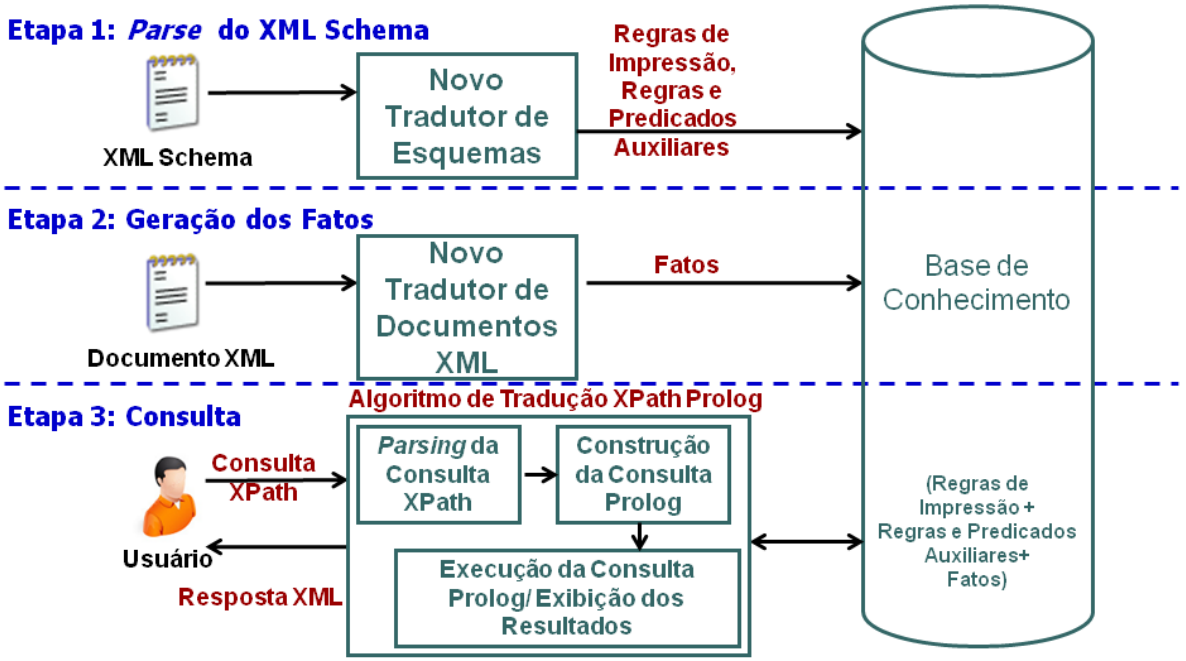


Figura 29: Arquitetura da abordagem XP2PL

5.2 MÁQUINA DE EXECUÇÃO PROLOG

A solução XMLInference original utilizava a máquina Java TuProlog (DENTI; OMICINI; RICCI, 2001) para execução de consultas. Com a evolução da complexidade das consultas executadas durante o período de desenvolvimento e testes dessa dissertação, percebeu-se que a máquina de execução TuProlog não possuía a robustez necessária, principalmente quando o tamanho das bases de fatos aumentava.

Desta forma optou-se pela utilização da máquina de execução Prolog SWI-Prolog (JAN WIELEMAKER *et al.*, 2012). Além da sua grande aceitação no meio acadêmico, a máquina de execução desenvolvida em C++ proporciona bom desempenho na execução de consultas e possui documentação de fácil acesso.

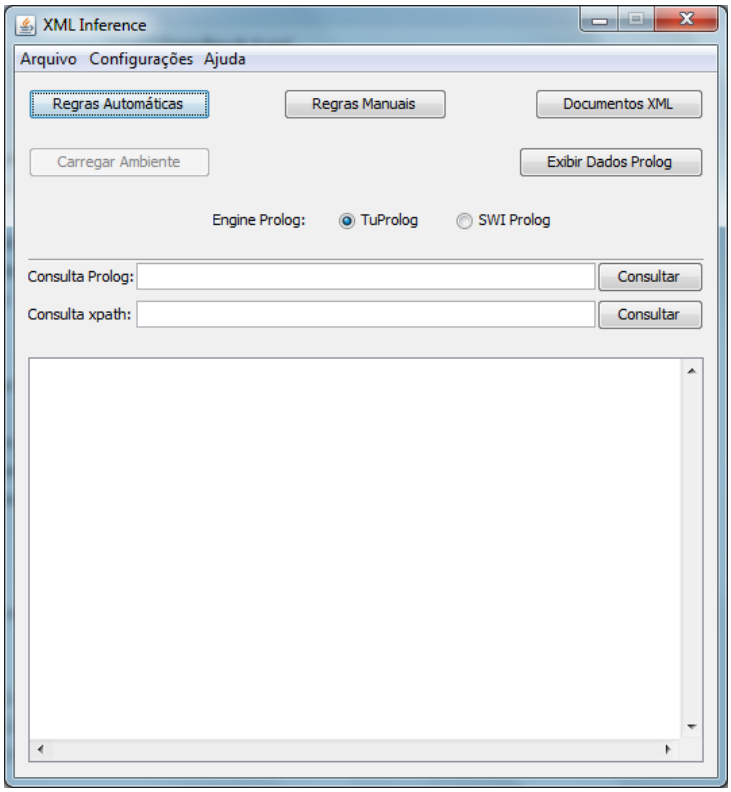


Figura 30: Interface principal do protótipo XMLInference

Apesar das características favoráveis, a execução de consultas através da máquina de execução SWI-Prolog necessita que o seu pacote esteja instalado na máquina do usuário, diferentemente da máquina de execução TuProlog disponibilizada em formato "*.jar*". Com o objetivo de proporcionar uma maior liberdade ao usuário, as duas máquinas de execução foram disponibilizadas no protótipo. Durante a execução de consultas o usuário é capaz de selecionar a máquina de execução desejada. A Figura 30 exibe a interface principal do protótipo exibindo os *radiobuttons* seletores das máquinas de execução.

5.3 NOMENCLATURA DOS PREDICADOS PROLOG

A utilização de diversos documentos XML durante os testes de tradução de consultas XPath evidenciaram problemas relacionados à nomenclatura dos predicados contidos nas bases de fatos Prolog. Elementos que possuíam seu *namespace* como prefixo de seus respectivos nomes como, por exemplo, *svg:rect*, eram traduzidos de forma literal (com o caractere ':') na base de fatos. Este tipo de nomenclatura não é aceito nas máquinas de execução Prolog utilizadas, e necessitou ser alterado. Para elementos com *namespaces* prefixados, o algoritmo de tradução automática acrescenta o caractere "-" ao início e término do *namespace*. Esta ação permite indentificar com exatidão a ocorrência de *namespaces*, já que elementos XML não podem possuir seus nomes iniciados com "-". Desta forma o elemento citado anteriormente teria sua nomenclatura traduzida como *-svg-rect*.

Outro ponto observado foi acerca da nomenclatura de atributos. Alguns dos documentos utilizados continham elementos com atributos que possuíam a mesma nomenclatura de outros elementos XML. Esta característica conferia a presença de fatos com mesmas assinaturas de predicado, porém com semânticas diferentes (como por exemplo entre um elemento e um atributo), o que ocasionava respostas indevidas quando consultados. Para sanar o problema, a nomenclatura dos atributos foi modificada, conferindo o nome do elemento dono do atributo, mais a expressão *_attribute_* ao predicado. Como exemplo, podemos citar um atributo *id* pertencente a um elemento *item*. Anteriormente, sua tradução poderia ser descrita como *id(56, 57, 'item1')*. Caso houvesse a presença de um elemento de nome *id*, sua tradução possuiria a mesma assinatura, como por exemplo, *id(60, 61, 'id1')*. A consulta Prolog *id(IDPAI, IDFILHO, VALOR)*, traria como resposta elementos e atributos de nome *id*. Com a nova abordagem, o atributo *id* passou a ser traduzido como *item_attribute_id(56, 57, 'item1')*. É importante observar que durante a impressão dos resultados, elementos que tiveram sua nomenclatura modificada serão impressos conforme sua nomenclatura original.

Por fim, os elementos mistos também tiveram sua nomenclatura modificada. Originalmente o predicado que representava os elementos mistos possuía a nomenclatura definida como *xml/mixedElement*. No entanto, com a troca para a máquina de execução SWI-Prolog, o caractere "/" passou a ser um problema, pois não é aceito pelo *parser* do SWI. Desta forma o predicado passou a ser nomeado como *xmlMixedElement*.

5.4 ALGORITMO DE CRIAÇÃO DA BASE DE FATOS

O algoritmo de tradução de documentos XML para bases de fatos Prolog do XMLInference foi implementado de forma correta pelos trabalhos anteriores, porém sua performance e robustez até então não tinha sido posta à prova. Como este trabalho também tem como objetivo avaliar o desempenho do algoritmo de tradução proposto, realizando experimentos com documentos XML de diversos tamanhos, foi necessário executar inúmeros processos de tradução de documentos.

O algoritmo anterior utilizava classes que agrupavam informações referentes a atributos, valores de elementos, etc.. Os objetos destas classes ainda utilizavam referências às instâncias filhas e ancestrais, armazenando (temporariamente) assim partes das subárvores do documento processado. Estas características oneravam o algoritmo, principalmente quando executado para documentos acima de 10 MB, acarretando eventuais estouros de memória. Como o foco deste trabalho passa pela tradução de grandes documentos XML, se fez necessária a criação de um novo algoritmo de tradução de documentos. O novo algoritmo continua utilizando a abordagem SAX (XML-DEV COMMUNITY, 2002) de leitura de documentos XML durante o *parse*, porém não faz uso de estruturas de dados complexas.

Seu funcionamento faz uso de uma simples estrutura de pilha aonde são guardados os identificadores utilizados na construção dos fatos, além de uma estrutura simples, armazenada brevemente, que guarda os atributos processados antes do evento de início de um elemento. Todas as funções de abertura e término de elementos, assim como seus atributos, foram refeitas para favorecer o desempenho do processo de geração dos fatos. Outro ponto observado e corrigido nesta nova versão foi em relação aos atributos do elemento raiz que não eram considerados no algoritmo anterior. A nova abordagem mudou drasticamente o desempenho do processo de tradução. Documentos XML que levavam dezenas de minutos para serem traduzidos, puderam ser processados em poucos minutos. Também não foram mais evidenciados problemas relacionados a estouro de memória. Os tamanhos dos documentos testados com o novo algoritmo variaram seus tamanhos de 100 KB à até 1GB.

5.5 EXECUÇÃO DE CONSULTAS PROLOG

Inicialmente a base Prolog utilizada para execução das consultas XPath traduzidas possuía seu conteudo formado pelos fatos Prolog e as respectivas regras automáticas produzidas pela leitura do XML Schema. No entanto, em alguns casos havia conflito entre a estrutura dos fatos e a estrutura das regras automáticas, o que provocava problemas semelhantes aos relacionados à nomenclatura dos fatos. A Figura 31 mostra a composição do

elemento *keywordType*. Podemos observar que o elemento é misto e que possui como possíveis filhos os elementos *bold* ou *emph*. A presença de elementos filhos pode não ocorrer devido à presença do atributo *minOccurs= 0*. As respectivas regras automáticas para o elemento *keyword* do tipo *keywordType* são descritas na Figura 32. Ambas as regras objetivam resgatar o identificador do elemento pai de *keyword* (*IDPARENT*), seu próprio identificador (*IDKEYWORD*) e o valor de um elemento filho (*BOLD* ou *EMPH*).

Já o algoritmo de tradução de documentos XML de Lima *et al.* (2012) considera que a estrutura dos possíveis fatos para o elemento *keyword* são *keyword(IDPARENT, IDKEYWORD)*, para o caso de ocorrências de elementos filhos, e *keyword(IDPARENT, IDKEYWORD, KEYWORDVALUE)*, para o caso do elemento *keyword* não possuir elementos filhos. Quando considerados em conjunto, o fato *keyword(IDPARENT, IDKEYWORD, KEYWORDVALUE)* pode possibilitar a geração de conflito com alguma das regras automáticas da Figura 32 , pela razão de possuírem a mesma assinatura.

```
1 <xs:complexType name="keywordType" mixed="true">
2   <xs:choice minOccurs="0" maxOccurs="unbounded">
3     <xs:element name="bold" type="boldType"/>
4      <xs:element name="emph" type="emphType"/>
5   </xs:choice>
6 </xs:complexType>
```

Figura 31: Composição do elemento *keywordType*

```
keyword(IDPARENT, IDKEYWORD, BOLD) :- bold(IDKEYWORD, IDBOLD, BOLD).
keyword(IDPARENT, IDKEYWORD, EMPH) :- emph(IDKEYWORD, IDEMPH, EMPH).
```

Figura 32: Regras automáticas para o elemento *keyword* do tipo *keywordType*

O foco deste trabalho não envolve a utilização das regras criadas de forma automática, pois para a resolução de uma consulta XPath somente se faz necessária a presença da base de fatos acompanhada de algumas funções de apoio. Desta forma, as regras automáticas passaram a ser desconsideradas durante a execução de consultas XPath traduzidas para Prolog.

Outra modificação no protótipo XMLInference foi em relação aos resultados exibidos em sua interface. Nos trabalhos anteriores o resultado Prolog era exibido de forma nativa na interface. Este trabalho desenvolveu uma técnica de impressão dos resultados das consultas em formato nativo XML, a qual será detalhada em capítulo posterior. Esta técnica possibilitou a exibição dos resultados de consultas XPath inseridas no protótipo pelo usuário. A Figura 33 exibe um exemplo de consulta XPath que foi traduzida para Prolog e executada na máquina

de execução SWI-Prolog. Conforme mencionado, sua respectiva resposta é exibida de forma transparente ao usuário.

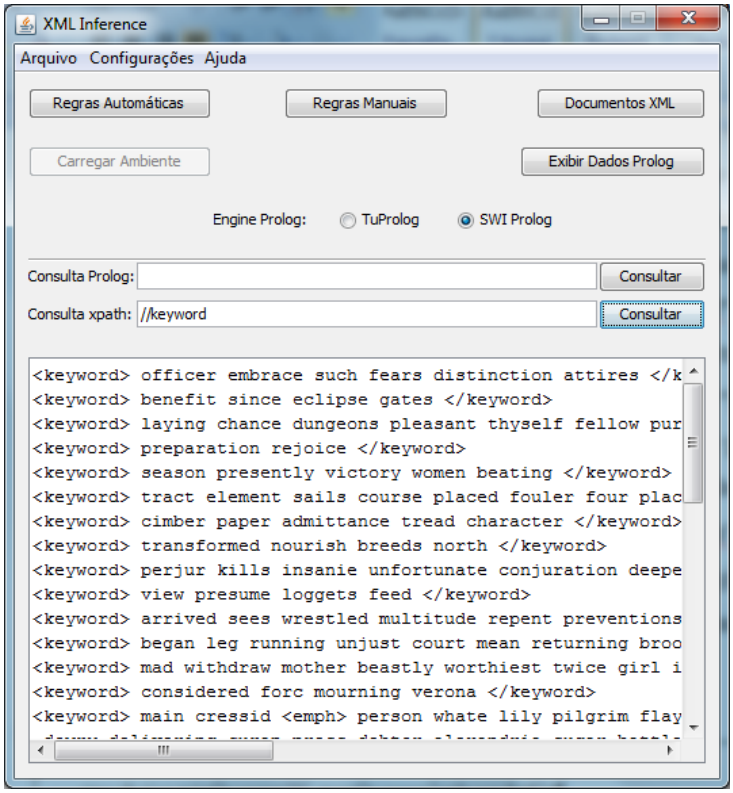


Figura 33: XMLInference: Execução de consulta XPath traduzida para Prolog e sua respectiva resposta

5.6 OCORRÊNCIAS DE ELEMENTOS

Conforme mencionado no capítulo anterior, durante a criação das regras automáticas, a abordagem original de Lima *et al*. (2012) desconsidera delimitadores de grupo e elementos que contenham o atributo *minOccurs=0*. A motivação é que como não há obrigatoriedade da existência de elementos que possuem este atributo, suas respectivas regras automáticas poderiam não funcionar corretamente, já que contêm referência explícita ao suposto predicado. Um problema semelhante poderia acontecer no contexto das traduções de consultas XPath para Prolog. O algoritmo de tradução de consultas XPath utiliza o XML Schema para montagem da consulta principal e construção de regras auxiliares, como, por exemplo, a impressão de elementos em formato XML. Desta forma, a não existência de um elemento que esteja declarado dentro de uma estrutura *choice* ou ainda que possua o atributo *minOccurs =0* poderia acarretar na geração de uma consulta Prolog inválida.

Para solucionar o problema, algumas medidas foram adotadas. Em primeiro lugar, a leitura do XML Schema se tornou pré-requisito para a execução do processo de tradução do

documento XML. Esta medida tem como objetivo conhecer a *priori* quais elementos podem se enquadrar nos casos de eventual não ocorrência, descritos anteriormente. De posse destas informações, a segunda medida pôde ser executada. Esta medida objetiva construir predicados de apoio, específicos para cada elemento reconhecido anteriormente, para sua posterior inserção na base de fatos. A Figura 34 exibe um exemplo destes predicados para um elemento que possui o atributo um delimitador *choice* com o atributo *minoccurs = 0*.

```
1 <xs:complexType name="keywordType" mixed="true">
2   <xs:choice minOccurs="0" maxOccurs="unbounded">
3     <xs:element name="bold" type="xs:string"/>
4      <xs:element name="italic" type="xs:string"/>
5   </xs:choice>
6 </xs:complexType>
```

```
bold(_,_,_) :- false.
italic(_,_,_) :- false.
```

Figura 34: Composição do elemento *keywordType*, e seus respectivos predicados de apoio

Para causar o mínimo de impacto, a construção destes tipos de predicados é realizada de forma semelhante aos fatos já traduzidos do documento. Sua nomenclatura segue as mesmas regras de criação de fatos descritas anteriormente, seguida dos respectivos identificadores e valor (caso possua), declarados de forma anônima. O valor *false* é atribuído a estes predicados de modo que sua presença não crie qualquer tipo de impacto nas respostas produzidas pela execução das consultas Prolog. Caso os elementos envolvidos sejam complexos, predicados de apoio para os seus respectivos elementos filhos também são criados, assim como os atributos envolvidos. A adoção destas medidas também possibilita a criação de regras automáticas para elementos que possuam o atributo *minoccurs = 0*.

5.7 CONSIDERAÇÕES FINAIS

Este capítulo apresenta inúmeras melhorias e correções realizadas no protótipo XMLInference. Todas as modificações realizadas no decorrer da pesquisa foram importantes para possibilitar o desenvolvimento do processo de tradução, execução e obtenção de resultados das consultas XPath em Prolog. O próximo capítulo trata destes temas, que fazem parte do foco principal deste trabalho.

Capítulo 6 – ALGORITMO DE TRADUÇÃO E IMPRESSÃO

6.1 INTRODUÇÃO

Este capítulo tem como objetivo descrever em alto nível.o algoritmo responsável por traduzir consultas XPath para Prolog e imprimir os resultados de forma automatizada, foco da abordagem X2PL. As técnicas de tradução de expressões, operadores e funções são agrupadas e descritas de acordo com suas características em comum. Todos os exemplos descritos neste capítulo utilizam o documento XML, inspirado na DBLP (LEY, 2003), e seu respectivo XML Schema descritos no Capítulo 2.

6.1.1 VISÃO GERAL DO ALGORITMO DE TRADUÇÃO

Uma vez que o processo de tradução do documento XML foi realizado, a base de fatos Prolog pode ser consultada. Ao receber uma consulta XPath como parâmetro de entrada, o algoritmo se encarrega do processo de análise e fragmentação dos seus elementos, respeitando as ocorrências de filtros, expressões e funções utilizadas. Após esta etapa, O algoritmo de tradução realiza a tarefa de tradução de cada fragmento, reescrevendo-os em linguagem Prolog. Cada predicado ou conjunto de predicados é unido pelos operadores Prolog E "," ou OU ";", de acordo com o contexto.

Em seguida ao processo de tradução da consulta XPath, o algoritmo constroi automaticamente os predicados Prolog responsáveis pela impressão dos elementos resposta resultantes da consulta. Estes predicados são unificados à consulta Prolog principal, e enviados à máquina de execução Prolog. A saída gerada pela execução da consulta Prolog corresponde aos resultados da respectiva consulta, já impressos em formato XML.

6.1.2 *PARSING* DA CONSULTA XPATH

Dada uma consulta XPath de entrada, o processo de tradução realiza o processo de *parse*, separando a consulta em *tokens*, utilizando para tal os caracteres "*/*" ou "//". Os *tokens* são processados e colocados em uma lista de acordo com a ordem de aparecimento na consulta. Como um exemplo, na consulta */dblp/book/title* os *tokens* são representados pelos elementos descritos no caminho da consulta: *dblp*, *book* e *title*.

Durante o *parse* de uma consulta XPath, além da identificação dos elementos presentes no caminho, há o processamento de possíveis filtros inseridos pelo usuário. O *parser* dos filtros é de suma importância, pois pode determinar diretamente os elementos retornados pela consulta XPath, ou ainda, pode delimitar o conjunto de elementos

intermediários relacionados com o próximo nó do caminho. De forma semelhante ao *parser* dos elementos do caminho da XPath, os filtros são processados pela identificação de um intervalo de colchetes "[...]". Durante seu processamento, a ocorrência de operadores ("=", ">", "≥", "<", "≤", *and* e *or*) é representada no algoritmo em formato de árvore, com operandos esquerdo e direito.

A Figura 35 exemplifica um filtro aplicado diretamente no conjunto de elementos que serão retornados pela consulta *//article[@rating = 'A']*. Nesse caso, somente os elementos *article* que contêm o atributo *rating* com valor "A" serão retornados. A representação em formato de árvore do filtro desta consulta possui o operador "=" como raiz, acompanhado do operando esquerdo *@rating* e do operando direto "A".

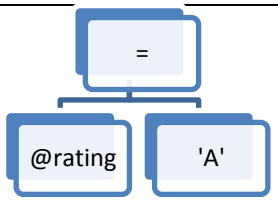
//article[@rating = 'A']	

Figura 35: Consulta XPath e a representação de seu filtro em formato de árvore

Já a consulta */dblp/proceedings[@mdate = '2016-04-01']/booktitle*, por exemplo, aplica um filtro intermediário na definição do caminho XPath, limitando que os elementos *booktitle* do caminho, devem possuir um elemento pai *proceedings* com atributo *mdate* de valor "*2016-04-01*". De forma semelhante ao exemplo anterior, o respectivo filtro é processado e estruturado em forma de árvore, para sua posterior inserção na já mencionada lista de elementos processados. A identificação de um filtro permite distingui-los dos demais elementos da lista no momento da tradução da consulta XPath.

O processo de realização do *parser* de filtros possui ainda definição recursiva, já que em uma mesma consulta XPath é possível definir filtros aninhados. A consulta */dblp/article[authors[@comment = 'none']/author/name='Leo Cesar']* possui filtros com esta característica. O elemento *article* retornado pela XPath deve possuir o elemento filho *authors*, que por sua vez deve possuir o atributo *comment* de valor "*none*", além do elemento descendente *name*, filho de*author*, com valor "Leo Cesar". Neste contexto a identificação e processamento dos filtros mais internos são realizados primeiro, facilitando o processo posterior de construção da consulta Prolog.

Ainda sobre o processamento de filtros, a linguagem XPath disponibiliza uma série de funções pré-definidas. De maneira semelhante, o algoritmo de *parser* de filtros prevê um número considerável de funções presentes na XPath, as quais são detalhadas nas próximas

seções. O processamento de funções presentes em filtros pode ser complexo, já que os argumentos de entrada de uma função podem ser o resultado da aplicação de outras funções. Novamente o algoritmo de *parser* utiliza recursividade, aonde cada argumento é processado em separado, preservando a ordem das chamadas durante a tarefa de construção da consulta Prolog equivalente.

A consulta */dblp[string-length(translate(proceedings/address," ",""))* > *30]*, por exemplo, contém a função *translate*() como argumento da função *string-length*(). No momento da verificação dos argumentos da função *string-length*(), há a identificação e *parse* da função *translate*() que é inserida na lista de estruturas processadas da consulta. Desta forma, no momento da construção da consulta Prolog, o resultado da função *translate*() será traduzido primeiro e seu resultado será o argumento da tradução da função *string-length*().

Da mesma forma é possível combinar o uso de funções com operadores lógicos ou mesmo com operações matemáticas, conforme exibe a Figura 36. Novamente a representação em formato de árvore é utilizada pelo algoritmo para estruturar os elementos envolvidos. A representação presente na Figura 36 possui o operador “>” como elemento raiz, com o operando direito “8” e o operando esquerdo “-”. Este último tratando-se de um operador, possui como operando esquerdo o resultado da função *number*(*volume*) e como operando direito o resultado da função *number*(*number*).

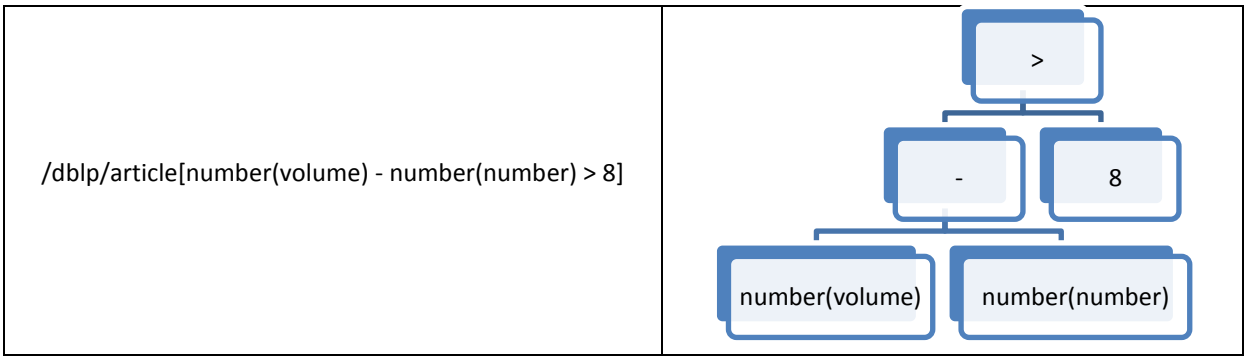


Figura 36: Exemplo de expressão matemática dentro de filtro em uma XPath e seu respectivo formato de árvore

6.2 ALGORITMO DE TRADUÇÃO DE CONSULTAS XPATH

Após a atividade de decomposição da consulta XPath, a próxima etapa é a construção da consulta Prolog. Iterações são realizadas sobre a lista que contém as estruturas que foram processadas na etapa de *parsing*. Cada elemento presente na lista é verificado e sua respectiva tradução para um ou mais predicados Prolog procede de acordo com sua representação, ou

seja, se o elemento apenas representa uma parte do caminho, ou se representa um filtro, função, etc.

A construção do predicado Prolog de todos os elementos é auxiliada pelo arquivo XML Schema do respectivo documento, consumido na etapa de tradução do documento XML. Conforme já mencionado, o XML Schema fornece informações sobre cada elemento passível de ser encontrado no documento XML, além da sua definição (simples ou complexo) e composição. As subseções a seguir detalham o algoritmo de tradução.

6.2.1 NOMENCLATURA DAS VARIÁVEIS

A representação de um nó XML durante a tradução de uma consulta XPath é composta basicamente por um identificador para o elemento pai (com exceção do elemento raiz), um segundo identificador para o próprio elemento e uma variável, caso ele seja um elemento simples. A nomenclatura das variáveis que representam a composição do predicado de um elemento segue um padrão determinado. A padronização é importante para que haja o casamento dos identificadores dos elementos que compõem o caminho pré-definido pelo usuário na consulta. A nomenclatura da variável que representa o primeiro identificador é iniciada pelo termo *"ID"* seguido pelo nome do elemento pai do elemento corrente. Em seguida a variável que exerce o papel do segundo identificador é iniciada novamente por *"ID"* seguido pelo nome do próprio elemento. Por fim, caso seja necessário, a última variável é descrita somente pelo nome do elemento. A Figura 37 exemplifica a importância da correta nomenclatura dos elementos para ligação do caminho definido por uma consulta XPath. As variáveis que definem o próprio identificador do elemento em questão devem ser as mesmas que identificam o elemento pai do predicado subsequente. Note que as variáveis *IDDBLP* e *IDARTICLE* serão unificadas pelo Prolog, numa operação com semântica equivalente a uma junção, restando *IDAUTHORS* como variável não unificada. Nesse caso, a consulta retorna todas as opções de valores de *IDAUTHORS* para cada combinação válida de *IDDBLP* e *IDARTICLE*.

Consulta XPath:
/dblp/article/authors
Consulta Prolog:
dblp(IDDBLP), article(IDDBLP, IDARTICLE), authors(IDARTICLE, IDAUTHORS).

Figura 37: Exemplo de correlação entre regras

6.2.2 MONTAGEM DE PREDICADOS DE ELEMENTOS

Durante a montagem da consulta Prolog, há a verificação se o nó que representa o elemento em questão é o nó raiz, ou ainda se o mesmo é um elemento simples ou complexo de acordo com o XML Schema associado. A montagem de predicados de elementos simples objetiva construir predicados que sejam correspondentes aos gerados no processo de criação da base de fatos, seguindo o padrão de nomenclatura descrito na seção anterior. Porém para casos em que o elemento possa assumir mais de uma formação dentro da base de fatos, há um tratamento especial.

A Figura 38 exibe a composição do nó *keyword* e sua respectiva tradução Prolog em um caminho dentro de uma consulta XPath. O nó complexo *keyword*, do tipo *keywordType*, apresenta em sua composição possíveis ocorrências de elementos filhos dentro de uma estrutura *choice*. O atributo *minOccurs* com valor zero indica que a presença dos elementos filhos não é obrigatória, além do atributo *mixed* indicar que esse é um elemento misto. O predicado Prolog *keyword(IDFATHER, IDKEYWORD)* representa os fatos na hipótese do nó possuir elementos filhos (as ocorrências de seus valores serão representados pelos elementos mistos *XMLMixedElement*). Já o predicado *keyword(IDFATHER, IDKEYWORD, KEYWORD)* representa fatos aonde o nó *keyword* não possui descendentes. Como a priori não é possível delimitar qual das ocorrências faz parte efetiva da resposta da consulta XPath definida pelo usuário, os dois predicados são utilizados separando a execução de ambos pelo operador Prolog OU *";"*.

Composição:

```
1 <xs:complexType name="keywordType" mixed="true">
2    <xs:choice minOccurs="0" maxOccurs="unbounded">
3      <xs:element name="bold" type="xs:string"/>
4      <xs:element name="italic" type="xs:string"/>
5    </xs:choice>
6 </xs:complexType>
```

Regra Prolog:

```
keyword(IDTEXT, IDKEYWORD, KEYWORD); keyword(IDTEXT, IDKEYWORD).
```

Figura 38: Composição do elemento *keyword* e seus respectivos predicados Prolog

De forma semelhante, elementos complexos sem filhos são representados da mesma maneira que o exemplo anterior. A Figura 39 exibe o nó *personref* e sua respectiva tradução Prolog em um caminho dentro de uma consulta XPath.

O predicado Prolog *personref(IDFATHER, IDPERSONREF)* representa o elemento *personref* quando não há a ocorrência de conteúdo associado, assim como o predicado

personref(IDFATHER, IDPERSONREF,PERSONREF) representa o elemento *personref* quando há a ocorrência de conteúdo. Ambos os predicados são separados pelo operador Prolog OU ";".

Composição:

```
1 <xs:complexType name="personrefType">
2     <xs:attribute name="person" use="required" type="xs:NCName"/>
3 </xs:complexType>
```

Regra Prolog:

```
personref(IDBIDDER, IDPERSONREF, PERSONREF); personref(IDBIDDER, IDPERSONREF).
```

Figura 39: Composição do elemento *personref* e seus respectivos predicados Prolog

6.2.3 TRADUÇÃO DE FILTROS, FUNÇÕES E OPERADORES

Ainda durante a fase de construção da consulta Prolog, estruturas que não representam elementos (ou seja, filtros, funções ou operadores lógicos) são tratados de forma diferenciada pelo algoritmo de tradução.

Conforme já mencionado, a utilização de filtros objetiva restringir o resultado final ou ainda reduzir o número de elementos intermediários presentes no caminho definido pela consulta XPath. O algoritmo de tradução de consultas verifica os elementos que fazem parte do mesmo filtro, e em seguida analisa o seu conteúdo em busca de funções e operadores utilizados. Após o processamento do conteúdo do filtro corrente, seu retorno é agregado à consulta principal através da utilização do operador Prolog E *","*, aproveitando o poder de expressão lógica da linguagem. A Figura 40 exibe uma consulta XPath com a utilização de filtro simples sobre o elemento *article* e sua respectiva tradução Prolog.

Consulta XPath:

```
//article[@rating = 'A']
```

Consulta Prolog:

```
article(IDNOPARENT, IDARTICLE), RATING = 'A', article_attribute_rating(IDARTICLE, IDRATING, RATING).
```

Figura 40: Exemplo de filtro em uma em uma XPath e sua respectiva tradução Prolog

Caso a consulta utilize filtros intermediários ou aninhados em uma consulta XPath, há a necessidade da atualização das referências ancestrais utilizadas durante a tarefa de montagem de predicados. A Figura 41 exibe a utilização de um filtro intermediário dentro de uma consulta XPath. Ao final do filtro as referências ancestrais apontam para os elementos

internos ao mesmo. A cada demarcação de início de filtro em uma consulta, o algoritmo de tradução preserva as referências ancestrais correntes e as atualiza ao final do mesmo.

Consulta XPath:
/dblp/mastersthesis[@key = 'mastersthesis/l/LeoCesar']/authorname
Consulta Prolog:
dblp(IDDBLP), mastersthesis(IDDBLP, IDMASTERSTHESIS), KEY = 'mastersthesis/l/LeoCesar', mastersthesis_attribute_key(IDMASTERSTHESIS, IDKEY, KEY), authorname(IDMASTERSTHESIS, IDAUTHORNAME, AUTHORNAME).

Figura 41: Exemplo de filtro intermediário em uma XPath e sua respectiva tradução Prolog

Conforme descrito anteriormente, o algoritmo de tradução proposto prevê a utilização de vários operadores e funções empregadas pelo usuário. Os elementos criados na tarefa de *parse* da consulta podem conter expressões delimitadas pelos operadores básicos aritméticos (soma, subtração, multiplicação e divisão) e comparações lógicas (“>”, “<”, “=”, “≥”, “≤”, *and* e *or*). O algoritmo de tradução processa o conteúdo das estruturas de forma recursiva, remontando a árvore de execução original. A cada chamada, caso haja a presença de funções ou outros operadores específicos, o algoritmo encaminha os mesmos a sub-rotinas especializadas. A estrutura de dados proveniente da fase de *parse* da consulta XPath original permite a reconstrução de funções e filtros de forma recursiva, respeitando assim a ordem de execução de cada termo. Esta característica permite a ocorrência de funções que possuam outras funções como argumento. Cada execução preliminar possui seu valor armazenado em variáveis genéricas, as quais são utilizadas como argumento de entrada da próxima função, como pode ser observado na Figura 42. O detalhamento de funções e operadores é realizado a seguir.

```
sum(LIST,RESULT1), myfloor(RESULT1, RESULT2) , RESULT2 >= 18
```

Figura 42: Armazenamento do resultado de funções em variáveis

Referência a múltiplos nós

A seleção de múltiplos nós (“*”) pode ser utilizada de inúmeras maneiras em uma consulta XPath. É possível realizar a seleção de elementos intermediários dentro de um caminho XPath (Figura 43), ou ainda selecionar como resultado qualquer nó filho (Figura 44). Semelhantemente, é permitido efetuar menções a atributos de forma genérica (Figura 45). A

técnica de tradução identifica as ocorrências destas referências agrupando-as de forma sequencial e submetendo-as à sub-rotina apropriada.

```
/dblp/*/title
```

Figura 43: Seleção múltipla de elementos intermediários em uma XPath

```
/dblp/book/*
```

Figura 44: Seleção de múltiplos elementos no resultado de uma XPath

```
//proceedings[@*]
```

Figura 45: Seleção de múltiplos atributos em um filtro XPath

Para as referências a elementos intermediários do caminho, a sub-rotina citada identifica o elemento imediatamente **anterior** ao operador *"*"*, para que, de forma recursiva, seja possível analisar os seus possíveis elementos filhos. Para cada elemento filho, um novo predicado é construído, realizando a tentativa de casamento de um dos identificadores filhos do elemento do nível referenciado, com o imediato elemento **posterior** ao operador *"*"*. A separação de cada possível conjunto de predicados é realizada pelo operador OU *";"*. Todos os identificadores do elemento em questão são coletados com a função Prolog *findall*, para posterior tentativa de casamento. A Figura 46 exibe um exemplo desta natureza. Na sua tradução Prolog, cada filho do elemento *dblp* possui seu próprio predicado, de forma que haja a tentativa de casamento dos seus identificadores com o predicado do elemento posterior ao operador "*" (destaque em negrito). Na hipótese do operador *"*"* ser utilizado ao final de uma consulta, a sub-rotina somente busca todos os possíveis filhos do último elemento definido conforme pode ser visualizado na Figura 47, em negrito.

Consulta XPath:

```
/dblp/*/title
```

Consulta Prolog:

```
dblp(IDDBLP), findall(IDWILD, (IDWILD = IDARTICLE, article(IDDBLP, IDARTICLE));
(IDWILD = IDPROCEEDINGS, proceedings(IDDBLP, IDPROCEEDINGS));
(IDWILD = IDBOOK, book(IDDBLP, IDBOOK));
(IDWILD = IDMASTERSTHESIS, mastersthesis(IDDBLP, IDMASTERSTHESIS));
(IDWILD = IDPHDTHESIS, phdthesis(IDDBLP, IDPHDTHESIS));
(IDWILD = IDWWW, www(IDDBLP, IDWWW)), LISTWILD),
 member(IDWILD, LISTWILD), title(IDWILD, IDTITLE, TITLE).
```

Figura 46: Consulta XPath com múltipla de elementos intermediários e sua respectiva tradução Prolog

Consulta XPath:
/dblp/book/*
Consulta Prolog:
dblp(IDDBLP), book(IDDBLP, IDBOOK),
findall(IDNODEORDER, (IDNODEORDER = IDYEAR, year(IDBOOK, IDNODEORDER,_);
IDNODEORDER = IDAUTHORNAME, authorname(IDBOOK, IDNODEORDER, _); IDNODEORDER = IDTITLE,
title(IDBOOK, IDNODEORDER, _)), LISTNODEORDER).

Figura 47: Consulta XPath com seleção de múltiplos elementos no resultado e sua respectiva tradução Prolog

Operações relativas à posição

A linguagem de consulta XPath permite ao usuário realizar operações que fazem referências posicionais a um nó desejado. A abordagem proposta permite a utilização de expressões: ".", "..", eixos (*axes*): *parent*::, *following*, *following-sibling*, *preceding*, *preceding-sibling*, além das funções *position*() e *last*().

A expressão *".."* seleciona o elemento atual dentro de uma consulta XPath. A Figura 48 exibe a utilização da expressão dentro de um filtro, além da sua respectiva tradução Prolog. O fato do algoritmo de tradução guardar a referência ao elemento anterior do caminho permite a identificação do elemento referenciado dentro do filtro. A função *mynumber()* foi criada para converter valores armazenados como texto em valores numéricos.

Consulta XPath:
//volume[. > 23]
Consulta Prolog:
volume(IDNOPARENT, IDVOLUME, VOLUME),
mynumber(VOLUME, VOLUME1), VOLUME1 @> 23

Figura 48: XPath com operador *"."* e sua respectiva tradução Prolog

A expressão *"..."* seleciona o elemento pai do elemento atual em uma consulta XPath. A Figura 49 mostra uma consulta XPath na qual deseja-se obter como resultado o elemento pai do elemento *journal*, além da sua respectiva tradução Prolog. O XML Schema inspirado na DBLP prevê que o elemento *journal* possui o elemento *article* como pai. Durante o processo de tradução, o algoritmo efetua buscas recursivas *bottom-up* remontando o caminho da árvore até o elemento desejado, conforme pode ser visualizado na Figura 49. A regra correspondente ao elemento pai é inserida antes da regra correspondente ao elemento filho. A relação de parentesco é efetuada pela nomenclatura das variáveis identificadoras. Na hipótese

em que mais de um elemento possa possuir o mesmo tipo de elemento filho definido no XML Schema, todas as tentativas de casamento serão realizadas.

Consulta XPath:
//journal/..
Consulta Prolog:
findall(IDREFPARENT, (
(IDREFPARENT = IDARTICLE, article(IDDBLP, IDARTICLE),
journal(IDARTICLE, IDJOURNAL, JOURNAL))), LISTREFPARENT).

Figura 49: Seleção do elemento pai como resultado em uma XPath e sua respectiva tradução Prolog

A Figura 50 exemplifica a obtenção dos elementos que possam realizar o papel de pai do elemento *title*. Assim como no operador *"*"*, os identificadores dos elementos pais, que realizaram o casamento desejado, são armazenados para posterior utilização.

Consulta XPath:
//title/..
Consulta Prolog:
findall(IDREFPARENT, (
(IDREFPARENT = IDARTICLE, article(IDDBLP, IDARTICLE), title(IDARTICLE, IDTITLE, TITLE));
(IDREFPARENT = IDPROCEEDINGS, proceedings(IDDBLP, IDPROCEEDINGS),
title(IDPROCEEDINGS, IDTITLE,TITLE));
(IDREFPARENT = IDBOOK, book(IDDBLP, IDBOOK), title(IDBOOK, IDTITLE,TITLE));
(IDREFPARENT = IDMASTERSTHESIS, mastersthesis(IDDBLP, IDMASTERSTHESIS),
title(IDMASTERSTHESIS, IDTITLE,TITLE));
(IDREFPARENT = IDPHDTHESIS, phdthesis(IDDBLP, IDPHDTHESIS),
title(IDPHDTHESIS, IDTITLE,TITLE));
(IDREFPARENT = IDWWW, www(IDDBLP, IDWWW),
title(IDWWW, IDTITLE,TITLE))), LISTREFPARENT).

Figura 50: Tradução de consulta com o operador "..*"* para múltiplos pais

O eixo *parent::* faz referência ao ancestral imediato do elemento atual em uma consulta XPath. Quando especificado ao final da consulta (Figura 51), permite retornar os elementos pais do último elemento presente no caminho.

```
//title/parent::book
```

Figura 51: Delimitação do elemento pai retornado em consulta XPath

Também há a possibilidade de ser utilizado dentro de um filtro, conforme exibe a Figura 52. Esta última forma lida somente com o poder da expressão lógica, retornando os elementos que possuem o pai especificado. Neste último exemplo o algoritmo de tradução avalia quais pais devem ser considerados na consulta para que as devidas regras sejam criadas. Cada possível casamento é separado novamente pelo operador OU *";"*. No caso de utilização prévia do operador *"*"* na mesma consulta, o algoritmo tratará sequencialmente seu uso colocando todas as relações de parentesco possíveis, para que, posteriormente ao processar a expressão *parent::*, os pais que não fazem parte da expressão lógica tenham seus identificadores invalidados na consulta. A Figura 52 exemplifica esta hipótese, aonde os elementos pais que não são *masterthesis* ou *phdthesis* têm seus respectivos identificadores associados ao valor zero (em negrito).

Consulta XPath:

```
/dblp/*/title[parent::mastersthesis or parent::phdthesis]
```

Consulta Prolog:

```
IDARTICLE is 0, IDPROCEEDINGS is 0, IDBOOK is 0, IDWWW is 0,
dblp(IDDBLP), findall(IDWILD, (IDWILD= IDARTICLE, article(IDDBLP, IDARTICLE));
(IDWILD = IDPROCEEDINGS, proceedings(IDDBLP, IDPROCEEDINGS));
(IDWILD = IDBOOK, book(IDDBLP, IDBOOK));
(IDWILD = IDMASTERSTHESIS, mastersthesis(IDDBLP, IDMASTERSTHESIS));
(IDWILD = IDPHDTHESIS, phdthesis(IDDBLP, IDPHDTHESIS));
(IDWILD = IDWWW, www(IDDBLP, IDWWW)), LISTWILD) , member(IDWILD, LISTWILD),
title(IDWILD, IDTITLE, TITLE).
```

Figura 52: Delimitação de parentesco e seleção múltipla em consulta XPath, e sua respectiva tradução Prolog

Os eixos *following*, *following-sibling*, *preceding* e *preceding-sibling* da XPath também são processados no algoritmo de tradução Prolog. O eixo *following* faz referência aos elementos posteriores ao elemento atual (Figura 53), enquanto o eixo *preceding* aponta para os elementos anteriores (Figura 54). Os eixos *following-sibling* e *preceding-sibling* possuem comportamento semelhante, porém restringem seu campo de ação aos elementos irmãos do elemento atual. Estas expressões podem referenciar múltiplos elementos, com a utilização do operador *"*"*, ou ainda especificar um único elemento. Atualmente o algoritmo de tradução

automática prevê suporte apenas ao uso destas expressões quando há a referência explícita ao elemento desejado.

A tradução dos eixos que fazem referência aos elementos irmãos é construída de forma mais simples do que as traduções que fazem referência aos elementos ancestrais e descendentes. Dada a referência a um determinado elemento, a função Prolog *findall* é utilizada para coletar os identificadores dos seus irmãos. A condição para que os elementos irmãos sejam posteriores ou anteriores ao referido elemento é realizada pelos operadores de maior “>” ou menor “<”. O conjunto dos elementos posteriores é representado pelos elementos que possuem os *ids* maiores que o *id* do elemento atual referenciado. Em contrapartida os elementos que possuem ids menores, representam os irmãos anteriores. As funções criadas *preceding()* e *following()* auxiliam a obtenção dos *ids* para posterior comparação. A Figura 55 e Figura 56 exibem exemplos de tradução dos mesmos.

```
/dblp/article/authors/author/following::author
```

Figura 53: Exemplo de XPath com eixo *following*

```
/dblp/article/authors/author/preceding::author
```

Figura 54: Exemplo de XPath com eixo *preceding*

Consulta XPath:

```
/dblp/www/following-sibling::www
```

Consulta Prolog:

```
findall(IDWWW, (dblp(IDDBLP), www(IDDBLP, IDWWW)), FOLLOWINGLIST), following(FOLLOWINGLIST,
FOLLOWINGFINALLIST), member(IDWWW1, FOLLOWINGFINALLIST), www(IDDBLP, IDWWW1),
www(IDDBLP, IDWWW2), IDWWW1 < IDWWW2.
```

Figura 55: Consulta XPath com a expressão *following-sibling* e sua respectiva tradução Prolog

Consulta XPath:

```
/dblp/www/preceding-sibling::www
```

Consulta Prolog:

```
findall(IDWWW, (dblp(IDDBLP), www(IDDBLP, IDWWW)), PRECEDINGLIST),
preceding(PRECEDINGLIST, PRECEDINGFINALLIST), member(IDWWW1, PRECEDINGFINALLIST),
www(IDDBLP, IDWWW1), www(IDDBLP, IDWWW2), IDWWW1 > IDWWW2.
```

Figura 56: Consulta XPath com a expressão *preceding-sibling* e sua respectiva tradução Prolog

Já os eixos *preceding* e *following*, apesar de possuírem características semelhantes, possuem um processo de tradução mais complexo. O resultado da expressão *preceding* não pode conter elementos que possam ser ancestrais do elemento referenciado, assim como o resultado da expressão *following* não pode conter elementos descentes do mesmo. Como esses elementos sempre têm *id* maior do que o *id* do pai, somente o simples critério de verificação do identificador do elemento não é suficiente.

Para a resolução do problema, o algoritmo de tradução criou funções que buscam os identificadores dos elementos descendentes ou ancestrais do referido elemento. Conforme já mencionado, durante o *parse* do XML Schema são construídas funções individuais para cada elemento do documento XML. A função que busca os identificadores ancestrais tem seu nome iniciado por *obtainAncestrals_*, enquanto a que busca os descendentes é iniciada por *obtainDescendents_*. Ambas terminam com o nome do respectivo elemento. A Figura 57 e Figura 58 exibem um exemplo de cada função, para o elemento *author*. Conforme pode ser observado na Figura 59 e Figura 60, após a obtenção dos identificadores necessários, o próximo passo é identificar do total de elementos candidatos, quais estão aptos a serem retornados. Estas verificações são ralizadas com os operadores de comparação maior “>” ou “<”, juntamente com a função criada de nome *notMember()*. A função *notMember()* verifica se o identificador não pertence a lista de identificadores retornados pelas funções *obtainAncestrals_* e *obtainDescendents_*.

```
obtainDescendents_author(IDELEMENT, LIST2):-
        author(IDPARENT, IDELEMENT), findall(LIST4, (
        (name(IDELEMENT, IDCHILD, _), obtainDescendents_name(IDCHILD, LIST4));
        (email(IDELEMENT, IDCHILD, _), obtainDescendents_email(IDCHILD, LIST4))), LIST5),
        flatten(LIST5, LIST6), addHeadList(IDELEMENT, LIST6, LIST2).
```

Figura 57: Regra Prolog para obtenção do *ids* dos elementos descendentes

```
obtainAncestrals_author(IDELEMENT, LIST):- author(IDPARENT, IDELEMENT, _),
        obtainAncestrals_authors(IDPARENT, LIST2), addTailList(IDPARENT, LIST2, LIST).
```

Figura 58: Regra Prolog para obtenção do *ids* dos elementos ancestrais

Consulta XPath:
/dblp/article/authors/author/following::author
Consulta Prolog:
dblp(IDDBLP), article(IDDBLP, IDARTICLE), authors(IDARTICLE, IDAUTHORS), author(IDAUTHORS, IDAUTHOR), obtainDescendents_author(IDAUTHOR, LISTPF), author(IDAUTHORS1, IDAUTHOR1), IDAUTHOR < IDAUTHOR1, notMember(IDAUTHOR1, LISTPF).

Figura 59: Consulta XPath com a expressão *following* e sua respectiva tradução Prolog

Consulta XPath:
/dblp/article/authors/author/preceding::author
Consulta Prolog:
dblp(IDDBLP), article(IDDBLP, IDARTICLE), authors(IDARTICLE, IDAUTHORS), author(IDAUTHORS, IDAUTHOR), obtainAncestrals_author(IDAUTHOR, LISTPF), author(IDAUTHOR1S, IDAUTHOR1), IDAUTHOR > IDAUTHOR1, notMember(IDAUTHOR1, LISTPF).

Figura 60: Consulta XPath com a expressão *preceding* e sua respectiva tradução Prolog

Por fim, as funções *position*() e *last*() permitem realizar filtros em função da posição do elemento atual, ou ainda obter o último elemento filho de um elemento. A tradução dinâmica destas funções é realizada com o auxílio de rotinas criadas previamente. A função nativa *findall* permite recuperar todas as ocorrências de um determinado tipo de fato Prolog. Esta função é utilizada em conjunto com a função própria *indexOf*() (Figura 61), que permite recuperar um elemento de uma posição específica em uma lista. Desta forma, para recuperar um elemento de uma determinada posição se faz necessário colher todas as ocorrências de seus identificadores, respeitando as devidas relações de parentesco, e armazená-los em uma lista. Posteriormente esta lista será percorrida para obtenção do identificador que está localizado na posição desejada.

```
indexOf([Element|_], Element, 1):- !.
indexOf([_|Tail], Element, Index):- indexOf(Tail, Element, Index1), !, Index is Index1 + 1.
```

Figura 61: Função Prolog *indexOf*()

A função *last*() utiliza os mesmos princípios, porém faz uso de outra função própria, de nome *listSize*(). Esta rotina recupera o tamanho da lista de identificadores retornada por *findall*(). Desta forma é possível recuperar o último identificador de uma lista, passando como argumento o seu tamanho.

A Figura 63 exibe um exemplo de tradução para uma consulta que utiliza a função *position*() e a Figura 64 mostra um exemplo de tradução para uma consulta com ambas as funções *position*() e *last*().

```
listSize([H|T], LENGTH) :- listSize([H|T], LENGTH,0).
listSize([H|T], LENGTH, TMPLENGTH):- NEWTMPLENGTH is TMPLENGTH +1,
        listSize(T, LENGTH, NEWTMPLENGTH).
listSize([], LENGTH, TMPLENGTH) :-  LENGTH = TMPLENGTH, !.
```

Figura 62: Função Prolog *listSize*()

Consulta XPath:

```
/dblp/www[position() = 3]
```

Consulta *Prolog*:

```
dblp(IDDBLP), www(IDDBLP, IDWWW),
findall(IDSEARCH1, www(IDDBLP, IDSEARCH1), LIST1 ),
indexOf(LIST1, IDWWW, RESULT1), RESULT1 = 3.
```

Figura 63: Consulta XPath com a função *position*() e sua respectiva tradução Prolog

Consulta XPath:

```
/dblp/www[position() = last()]
```

Consulta Prolog:

```
dblp(IDDBLP), www(IDDBLP, IDWWW),
findall(IDSEARCH2, www(IDDBLP, IDSEARCH2), LIST2), indexOf(LIST2, IDWWW, RESULT2),
findall(IDSEARCH3, www(IDDBLP, IDSEARCH3), LIST3 ),
listSize(LIST3, LENGTH3), indexOf(LIST3, IDWWW, LENGTH3 ), RESULT3 is LENGTH3, RESULT2 = RESULT3.
```

Figura 64: Consulta XPath com as funções *position*() *last*() e sua respectiva tradução Prolog

Operador de união

O operador de união *"|"* possibilita unir dois conjuntos de resultados retornados por consultas XPath (Figura 65). De maneira análoga o algoritmo de tradução para Prolog trata cada caminho separado pelo caractere *"|"* como uma consulta individual, aqui chamada de subconsulta. A tradução de cada subconsulta é concatenada às demais através do operador Prolog OU *";"* (Figura 65), garantindo assim a execução de todas as subconsultas.

Consulta XPath:
/dblp/article/title | /dblp/proceedings/title
Consulta Prolog:
(dblp(IDDBLP), article(IDDBLP, IDARTICLE), title(IDARTICLE, IDTITLE,TITLE)); (dblp(IDDBLP), proceedings(IDDBLP, IDPROCEEDINGS), title(IDPROCEEDINGS, IDTITLE, TITLE)).

Figura 65: XPath utilizando operador de união e sua respectiva tradução Prolog

Referências a *namespaces*

Em uma consulta XPath, é possível referenciar elementos através dos *namespaces* utilizados em um documento XML. A abordagem proposta nesta dissertação permite a utilização de algumas expressões e funções que tratam *namespaces*, como *name*(), *local-name*() e *namespace*(). Nativamente a função *name*() verifica a existência de elementos que possuam o exato nome desejado, diferindo da função *local-name*(), que desconsidera algum possível *namespace* atrelado ao elemento. Novamente, o auxílio do XML Schema é de suma importância para o algoritmo de tradução automática. O documento de definição de esquema possui informações acerca de nomes e presença de *namespaces* de um determinado elemento. Durante o processo de tradução, uma análise do XML Schema é realizada com o objetivo de verificar a possibilidade da existência do *namespace* ou nome do elemento desejado. Caso não sejam encontrados elementos válidos para a busca desejada, a tradução Prolog considera a consulta inválida *(false)*, poupando o esforço efetivo de busca no documento. Na hipótese da ocorrência de múltiplos elementos que atendam os requisitos de *namespace* da consulta, cada possível alternativa é separada pelo operador Prolog OU *";"*, semelhantemente ao que ocorre com outras operações. A Figura 66 exibe um exemplo da utilização da função *local-name*() e sua respectiva tradução Prolog. Neste exemplo como não existe outro elemento com o mesmo *local-name*() no esquema do documento, o algoritmo imprime todos os elementos *book*. O algoritmo de impressão será detalhado em subseção posterior.

Consulta XPath:
//*[local-name() = 'book']
Consulta Prolog:
(print_book(IDNOPARENT, IDNODEORDERSORTED)).

Figura 66: XPath com a função *local-name*() e sua respectiva tradução Prolog

Funções de processamento de texto

Funções de tratamento de *strings* também são traduzidas pela abordagem proposta neste trabalho. As funções *starts-with*(), *substring-before*(), *substring-after*(), *substring*(), *translate*(), *contains*() e *concat*() contam com rotinas equivalentes criadas para o processo de tradução. As funções de tratamento de *strings* podem ser utilizadas sozinhas dentro de filtros, ou ainda aninhadas servindo de argumento para outras funções. De maneira geral, as funções criadas para processamento de texto Prolog (*startsWith*(), *substringBefore*(), *substringAfter*(), *substring*(), *translate*(), *contains*() e *concat*()) recebem o conteúdo textual como parâmetro e em seguida o transforma em uma lista de caracteres. Desta forma, é possível realizar operações como substituição de parte dos caracteres, verificação e extração de conteúdo. Ao final do processamento, a lista caracteres é novamente convertida em texto e retornada. A Figura 67 apresenta uma consulta XPath contendo uma função de processamento de texto e sua respectiva tradução Prolog.

Consulta XPath:
/dblp/www[starts-with(title,'Leo')]
Consulta *Prolog*:
dblp(IDDBLP), www(IDDBLP, IDWWW), title(IDWWW, IDTITLE, TITLE), startsWith(TITLE, 'Leo').

Figura 67: Consulta XPath utilizando função de processamento de texto e sua tradução Prolog

6.3 FUNÇÕES NUMÉRICAS

Funções que tratam de conteúdo numérico são comumente utilizadas pelos usuários durante consultas XPath. A abordagem de tradução de consultas possibilita a utilização das funções *count*(), *sum*(), *ceiling*() e *floor*(). A função *count*() realiza a operação de contagem de elementos relativos a um determinado caminho. A Figura 68 contém um exemplo da sua utilização, juntamente com sua tradução Prolog. Os *ids* dos elementos (respeitando as devidas relações de parentesco) são atribuídos a uma lista com o auxílio da função *findall*. Em seguida a função Prolog desenvolvida *count*() (Figura 69) conta o tamanho da lista e a compara com o resultado esperado.

Consulta XPath:

```
/dblp/phdthesis[count(keyword) > 1]
```

Consulta Prolog:

```
dblp(IDDBLP), phdthesis(IDDBLP, IDPHDTHESIS),
findall(IDKEYWORD, ((keyword(IDPHDTHESIS, IDKEYWORD);keyword(IDPHDTHESIS,IDKEYWORD,_))), LIST ),
count(LIST, RESULT1), RESULT1 > 1.
```

Figura 68: Exemplo de utilização da função *count*() e sua respectiva tradução Prolog

```
count(List, Count) :- count(List, 0, Count).
count([], Accumulator, Accumulator).
count([Head|Tail], Accumulator, Result) :- NewAccumulator is Accumulator + 1,
    count(Tail, NewAccumulator, Result).
```

Figura 69: Função Prolog *count*()

A tradução da função *sum*() recebe uma lista de valores como parâmetro. Semelhantemente à função *count*(), a tradução da função XPath *sum*() atribui todos os valores dos elementos desejados (novamente respeitando as relações parentais) a uma lista que será o parâmetro de entrada da função *sum*() Prolog (Figura 70). O resultado do somatório também é atribuído a uma variável para uso posterior (Figura 71). Já a tradução das funções *ceiling*() e *floor()* utilizam as funções nativas do Prolog internamente, porém o seu resultado é exportado para uma variável (funções *myfloor*() e *myceiling*()), possibilitando sua manipulação posterior. A Figura 72 e Figura 73 exibem exemplos da sua tradução Prolog.

```
sum(List, Sum) :-sum(List, 0, Sum).
sum([], Accumulator, Accumulator).
sum([Head|Tail], Accumulator, Result):- mynumber(Head, NHEAD),
                NewAccumulator is Accumulator + NHEAD, sum(Tail, NewAccumulator, Result).
```

Figura 70: Tradução Prolog para a função *sum*()

Consulta XPath:

```
/dblp/article/authors[sum(author/score) >= 18.5]
```

Consulta *Prolog*:

```
dblp(IDDBLP), article(IDDBLP, IDARTICLE), authors(IDARTICLE, IDAUTHORS),
findall(SCORE, (author(IDAUTHORS, IDAUTHOR), score(IDAUTHOR, IDSCORE, SCORE)),LIST),
sum(LIST, RESULT1 ), RESULT1 >= 18.5.
```

Figura 71: XPath utilizando função *sum*() e sua respectiva tradução Prolog

Consulta XPath:
/dblp/article/authors[floor(sum(author/score)) >= 18]
Consulta *Prolog*:
dblp(IDDBLP), article(IDDBLP, IDARTICLE), authors(IDARTICLE, IDAUTHORS), findall(SCORE, (author(IDAUTHORS, IDAUTHOR), score(IDAUTHOR, IDSCORE, SCORE)),LIST), sum(LIST, RESULT1), myfloor(RESULT1, RESULT2), RESULT2 >= 18.

Figura 72: XPath utilizando função *floor*() e sua respectiva tradução Prolog

Consulta XPath:
/dblp/article/authors[ceiling(sum(author/score)) <= 19]
Consulta *Prolog*:
dblp(IDDBLP), article(IDDBLP, IDARTICLE), authors(IDARTICLE, IDAUTHORS), findall(SCORE, (author(IDAUTHORS, IDAUTHOR), score(IDAUTHOR, IDSCORE, SCORE)), LIST), sum(LIST, RESULT1), myceiling(RESULT1, RESULT2), RESULT2 =< 19.

Figura 73: XPath utilizando função *ceiling*() e sua respectiva tradução Prolog

6.4 ALGORITMO DE IMPRESSÃO

Os trabalhos de Lima *et al*. (2012) e Santos (2015) previam a impressão do resultado da consulta Prolog em formato XML. Durante os seus respectivos experimentos, cada consulta testada foi customizada manualmente para permitir a impressão do resultado no formato desejado. O trabalho de Santos *et al*. (2012) realiza a impressão dos resultados em formato XML através da interpretação do output da máquina TuProlog, o que pode impactar nos tempos de resposta.

Para cobrir as lacunas dos trabalhos anteriores, esta dissertação desenvolveu uma técnica automática de construção e obtenção dos resultados em formato XML. Durante a tarefa de construção da base de regras automáticas, um procedimento de construção de regras exclusivas de impressão é executado. Como resultado final, há a geração da chamada base de regras de impressão, aonde cada elemento do XML Schema possui uma regra própria que define como o mesmo será impresso. A impressão dos elementos contendo seus respectivos atributos, valores e elementos filhos é realizada com a função *WRITE/1 (ISO)* do Prolog, a qual escreve no *stream STDOUT* padrão do sistema operacional. Vale ressaltar que a funcionalidade visa utilizar funções nativas e meta-funções no padrão *ISO*, a fim de prover uma maior interoperabilidade entre diferentes máquinas Prolog. A construção das regras de impressão, assim como seu funcionamento é detalhada a seguir.

Conforme mencionado anteriormente, durante o parse do XML Schema, o processo de criação das regras de impressão é acionado. Em sua maioria, cada regra de impressão

construída tem seu nome iniciado pelo termo *print_*, seguido do nome do elemento em questão (Figura 74). Seguindo o padrão de nomenclatura dos fatos, as regras de impressão dos atributos são iniciadas pelo termo *print_attribute_*, seguidos do nome do atributo em questão (Figura 75). Por simplificação, todos os elementos mistos são impressos por uma única regra, de nome *print_XMLMixedElement* (Figura 76). De modo geral, cada regra de impressão possui dois argumentos, correspondentes ao identificador do elemento pai e o seu identificador, respectivamente. A exceção fica por conta das regras de impressão do nó raiz, que só possuem o seu próprio identificador como argumento (Figura 77). O conteúdo de todas as regras de impressão é iniciado com a representação do fato correspondente, objetivando desta forma o casamento dos identificadores passados como argumento.

```
print_mastersthesis(IDPARENT,IDTAG).
```

Figura 74: Assinatura da função Prolog de impressão de elementos simples e complexos

```
print_article_attribute_rating(IDPARENT,IDATTRIBUTE).
```

Figura 75: Assinatura da função Prolog de impressão de atributos

```
print_xmlMixedElement(IDTAG).
```

Figura 76: Assinatura da função Prolog de impressão de elementos mistos

```
print_dblp(IDTAG).
```

Figura 77: Assinatura da função Prolog de impressão do elemento raiz do documento

Durante a construção das regras de impressão para cada elemento presente no XML Schema, há verificação sobre sua complexidade. Caso o elemento seja do tipo simples *(xs:simpleType)* uma regra do tipo simples é criada, aonde há a impressão do início da *tag* do elemento, seguidos do seu possível valor e respectivo fechamento (Figura 78). Caso o elemento seja do tipo complexo (*xs:complexType*), a regra de impressão se torna um pouco mais complexa. Apesar do elemento complexo possuir em sua descrição quais filhos e atributos possui, a ordem em que os mesmos surgem no documento pode não estar fixada. A estrutura flexível de um documento XML permite que os elementos filhos surjam de maneira aleatória, ou mesmo nem apareçam no caso de um elemento complexo definido com a estrutura *xs:choice*, ou com *minOccurs=0*. Desta forma a técnica proposta utiliza os identificadores presentes em cada fato para obter a ordem em que os elementos filhos surgem no documento. Durante o processo de transformação do documento XML em base de fatos, os

identificadores de cada fato são definidos através de um contador numérico em ordem sequencial crescente. Assim sendo, um elemento presente no início do documento possui um identificador menor que outro elemento posterior.

```
print_title(IDPARENT, IDTAG) :- title(_, IDTAG, VALUE), write('<title>'), write(VALUE),
                write('</title>'), printnl('').
```

Figura 78: Exemplo de regra Prolog de impressão de elementos

Desta forma, o identificador de cada elemento filho é atribuído a uma lista, que posteriormente é ordenada. Após esse procedimento, a lista de identificadores ordenados é percorrida, passando o identificador ordenado como argumento para a função de impressão de cada elemento filho. O objetivo deste processo é realizar a tentativa de casamento do identificador da iteração corrente com algum identificador dos elementos filhos, através da utilização do operador OU *";"* (Figura 79). Amparado pela abordagem de tradução de fatos já mencionada, cada identificador da lista casará somente com uma regra de impressão por vez.

```
findall(IDCHILD, (authorname(IDTAG, IDCHILD, _); title(IDTAG, IDCHILD, _); year(IDTAG, IDCHILD, _) ), LIST),
                setof(X, member(X, LIST), LISTSORTED), member(IDCHILDSORTED, LISTSORTED),
                 (print_authorname(IDTAG, IDCHILDSORTED); print_title(IDTAG, IDCHILDSORTED);
                print_year(IDTAG, IDCHILDSORTED)).
```

Figura 79: Regra Prolog para impressão dos elementos filhos de *book*

Utilizando a mesma técnica, cada função de impressão dos elementos filhos é definida de forma recursiva aproveitando o grande potencial das máquinas de inferência. Para todos os nós que possuem atributos, a mesma estratégia é utilizada, pois pode não haver conhecimento prévio acerca da ordem de um atributo.

A regra de impressão dos atributos se assemelha a dos elementos simples. A mesma é iniciada pela declaração do predicado que representa o atributo desejado. Em seguida, há a impressão do nome do atributo acompanhado pelo símbolo de *"="* e seu respectivo valor obtido anteriormente (Figura 80). As regras de impressão de elementos complexos que possuam atributos em sua composição são definidas conforme a Figura 81 e a Figura 82. No início há a declaração do fato correspondente ao elemento em questão, e em seguida a impressão da sua *tag* de início. Imediatamente as funções de impressão dos atributos e filhos são executadas, através do processo de ordenação e tentativa de casamento. Por fim há a impressão da *tag* de finalização do nó.

```
print_article_attribute_rating(IDPARENT, IDATTRIBUTE):-
                article_attribute_rating(IDPARENT, IDATTRIBUTE, VALUE),
                write(' rating="'), write(VALUE), write('" ').
```

Figura 80: Exemplo de regra Prolog para impressão de atributos

```
print_book(IDPARENT, IDTAG) :- book(IDPARENT, IDTAG), write('<book'), print_book_childs(IDTAG).
```

Figura 81: Exemplo de regra Prolog para impressão de elemento complexo

```
print_book_childs(IDTAG):- write('>'); book(_, IDTAG),
        findall(IDCHILD, (authorname(IDTAG, IDCHILD, _); title(IDTAG, IDCHILD,_); year(IDTAG, IDCHILD, _)),LIST),
        setof(X, member(X, LIST), LISTSORTED), member(IDCHILDSORTED, LISTSORTED),
       (print_authorname(IDTAG, IDCHILDSORTED); print_title(IDTAG, IDCHILDSORTED);
       print_year(IDTAG, IDCHILDSORTED)); write('</book>'), printnl('').
```

Figura 82: Exemplo de regra Prolog para impressão dos filhos de elemento complexo

Outro fator importante é que esta técnica abstrai o problema de impressão de elementos com referências cíclicas diretas ou mesmo em alguns graus mais distantes. Na Figura 83 e Figura 84, por exemplo, o elemento *X* do tipo *Xtype* possui como um de seus possíveis filhos o elemento *Y* do tipo *YType*. De forma análoga, o elemento *Y* em sua definição possui o elemento *X* como possível filho. Este relacionamento cíclico direto também pode ocorrer de forma indireta conforme exemplo da Figura 85, Figura 86 e Figura 87. O elemento *X* do tipo *Xtype* possui o elemento *Y* como possível filho, e *Y*, do tipo *YType* por sua vez possui o elemento *Z* como filho. Já na composição do elemento *Z* do tipo *ZType*, *X* também aparece como filho, fechando o ciclo.

```
1 <xs:complexType name="XType" mixed="true">
2   <xs:choice minOccurs="0" maxOccurs="unbounded">
3     <xs:element name="Y" type="YType"/>
4    <xs:element name="Z" type="ZType"/>
5   </xs:choice>
6 </xs:complexType>
```

Figura 83: Composição do elemento *X* (Relacionamento cíclico direto)

```
1 <xs:complexType name="YType" mixed="true">
2    <xs:choice minOccurs="0" maxOccurs="unbounded">
3      <xs:element name="X" type="XType" />
4      <xs:element name="W" type="WType"/>
5   </xs:choice>
6 </xs:complexType>
```

Figura 84: Composição do elemento *Y* (Relacionamento cíclico direto)

```
1 <xs:element name="X" type="XType"/>
2
3 <xs:complexType name="XType" mixed="true">
4   <xs:choice minOccurs="0" maxOccurs="unbounded">
5     <xs:element name="Y" type="YType"/>
6     <xs:element name="W" type="WType"/>
7   </xs:choice>
8 </xs:complexType>
```

Figura 85: Composição do elemento *X* (Relacionamento cíclico indireto)

```
1 <xs:element name="Y" type="XType"/>
2
3 <xs:complexType name="YType" mixed="true">
4   <xs:choice minOccurs="0" maxOccurs="unbounded">
5     <xs:element name="Z" type="ZType" />
6     <xs:element name="K" type="KType"/>
7   </xs:choice>
8 </xs:complexType>
```

Figura 86: Composição do elemento *Y* (Relacionamento cíclico indireto)

Apesar da descrição de cada elemento complexo estar contida no XML Schema do documento, não há indicação prévia do número de recursões presentes em cada ciclo, pois isso depende do documento XML, e não do esquema. Essa característica evidenciou-se ao longo da análise dos diversos documentos XML no decorrer da pesquisa deste trabalho, e tornou-se um tema importante na fase de impressão do resultado das consultas. Após diversas abordagens desenvolvidas, a solução final subdivide o processo utilizando regras relativamente simples, que utilizam o potencial recursivo das máquinas de inferência. As técnicas aplicadas nesta funcionalidade tornaram-na eficaz e portável, podendo ser adaptada a diversas implementações que manipulem o resultado enviado à saída padrão.

```
1 <xs:element name="Z" type="XType"/>
2
3 <xs:complexType name="ZType" mixed="true">
4   <xs:choice minOccurs="0" maxOccurs="unbounded">
5     <xs:element name="X" type="XType" />
6     <xs:element name="A" type="AType"/>
7   </xs:choice>
8 </xs:complexType>
```

Figura 87: Composição do elemento *Z* (Relacionamento cíclico indireto)

6.5 CONSIDERAÇÕES FINAIS

Este capítulo descreve o algoritmo responsável por desenvolver a tradução automática de consultas XPath para Prolog, assim como a técnica de impressão de resultados. Diferentemente de outras abordagens, o algoritmo de tradução automática de consultas XPath para Prolog proposto nesse trabalho aumenta o poder de expressão das consultas, ao fornecer

suporte à uma gama de operadores, funções e possibilidades de filtros disponíveis na linguaguem XPath, conforme pode ser visualizado na Tabela 2.

Tabela 2: Funções e operadores disponíveis no algoritmo de tradução automática de consultas XPath

Funções e Operadores
/
//
=
>
<
>=
<=
*
parent::
following-sibling::
following::
preceding-sibling::
preceding::
namespace::
.
..
|
position()
last()
count()
local-name()
name()
lang()
starts-with()
substring-before()
substring-after()
substring()
translate()
concat()
floor()
ceiling()
sum()
round()

Após o processo de tradução, a execução da consulta XPath retorna seus respectivos resultados em formato nativo XML, graças à estratégia de impressão. O próximo capítulo descreve os experimentos e resultados obtidos com o trabalho desenvolvido.

Capítulo 7 – AVALIAÇÃO EXPERIMENTAL

7.1 INTRODUÇÃO

A avaliação experimental deste trabalho consistiu primariamente na análise acerca da expressividade do algoritmo de tradução automática de consultas XPath. O *benchmark* XPathMark (FRANCESCHET, 2005) foi utilizado durante os experimentos por prover uma série de consultas que abrangem inúmeros aspectos inerentes à XPath. As consultas do XPathMark foram desenvolvidas para o documento do *benchmark* XMark (SCHMIDT *et al.*, 2002) que trata de consultas XQuery (BOAG *et al.*, 2010).

A avaliação experimental foi realizada com 2 modelos de documentos fornecidos pelo XPathMark. O primeiro documento (D1), conforme mencionado anteriormente, é o mesmo utilizado pelo *benchmark* XMark, e trata da estrutura de um modelo de leilão online, conforme esquema exibido no Anexo A. O segundo documento (D2), consiste na estrutura de uma página HTML básica. A intenção do documento D2 é prover características que não estavam contidas no documento D1, como uso de *namespaces* e referências a contextos de linguagem (*lang*). O Anexo B exibe o esquema correspondente ao documento D2. Complementando a avaliação da abordagem, experimentos secundários foram realizados com o objetivo de verificar o desempenho da execução de consultas XPath nativas frente às automaticamente traduzidas para Prolog.

Durante os experimentos foram utilizadas os interpretadores XPath nativos Galax (SIMÉON *et al.*, 2000) (versão *1.0*) e Saxon (KAY, 2001) (versão *enterprise edition trial* 9.7), além do banco de dados XML Sedna (FOMICHEV; GRINEV; KUZNETSOV, 2006), versão 3.5- 64 *bits*. Para as consultas Prolog foi utilizada a máquina de execução SWI-Prolog (JAN WIELEMAKER *et al.*, 2012), versão 7.2.3- 64 *bits*). Todas as consultas foram executadas em uma máquina com as seguintes configurações: CPU Intel *Core* i7-4790K CPU 4.00GHZ, Memória 15,6 GB, Sistema Operacional Ubuntu 15.04

7.2 AVALIAÇÃO DE EXPRESSIVIDADE

7.2.1 OBTENÇÃO DAS CONSULTAS

O *benchmark* XPathMark oferece um total de 47 consultas (Anexo C) para o documento D1 e 12 consultas (Anexo D) para o documento D2. Deste total, algumas consultas foram desconsideradas.

Sobre o documento D1 as consultas 4, 6 e 7 foram desconsideradas por conterem operadores que fazem referência a elementos ancestrais ou descendentes. Estes operadores ainda não foram incorporados à abordagem de tradução atual. As consultas de 18 a 20 foram desconsideradas pelo motivo da técnica atual de tradução de documentos XML não considerar comentários e instruções de processamento. Há ainda o fator de que alguns elementos referenciados por estes operadores podem possuir referências cíclicas diretas e indiretas. A consulta 21 foi desconsiderada por conta da implementação atual ainda não conseguir lidar com a seleção de um elemento descendente, com possíveis referências cíclicas internas, sem especificação de nível. As consultas de 25 a 29 foram desconsideradas por conta da abordagem atual não considerar o uso de *ids* definidos no esquema do documento. As consultas 17 e 34 foram desconsideras pelo motivo das funções *namespace-uri* e *node* ainda não estarem contempladas na abordagem atual. As consultas 42 e 36 foram desconsideradas por considerar o conteúdo dos elementos descendentes do elemento *description*. A tradução atual da função *contains* considera somente o conteúdo atual do elemento corrente e não todo o seu conteúdo descendente. A função *string* ainda não é considerada na abordagem atual.

Sobre o documento D2, as consultas 1 e 2 foram desconsideradas pelo já citado motivo da não tradução de comentários e instruções de processamento. Da mesma forma a consulta 9 foi descartada por conta da função *namespace-uri* ainda não estar disponível nesta abordagem.

É válido mencionar que apesar do descarte de algumas consultas do *benchmark*, sua grande maioria foi utilizada durante os experimentos. Mais especificamente foram utilizadas 31 consultas para o documento D1 e 9 consultas para o documento D2. A Tabela 4 e a Tabela 3 exibem as consultas consideradas nos experimentos para cada documento.

Tabela 3: Consultas utilizadas durante os experimentos para o documento D1

Consultas XPath Documento D1	
Q1	/site/regions/*/item
Q2	/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/text/keyword
Q3	//keyword
Q5	/site/regions/*/item[parent::namerica or parent::samerica]
Q8	/site/open_auctions/open_auction[bidder[personref/@person = 'person0']/ following-sibling::bidder[personref/@person = 'person1']]
Q9	/site/open_auctions/open_auction[@id = 'open_auction0']/bidder/preceding-sibling::bidder
Q10	/site/regions/*/item[@id = 'item0']/following::item
Q11	/site/open_auctions/open_auction/bidder[personref/@person = 'person1']/ preceding::bidder[personref/@person = 'person0']
Q12	//item[@featured = 'yes']
Q13	//*[@id]
Q14	//person[namespace::xml]
Q15	//increase[. > 20]
Q16	//xml:*
Q22	/site/regions/namerica/item | /site/regions/samerica/item
Q23	/site/people/person[address and (phone or homepage)]
Q24	/site/people/person[not(homepage)]
Q30	/site/open_auctions/open_auction/bidder[position() = 1 and position() = last()]
Q31	/site/open_auctions/open_auction[count(bidder) > 5]
Q32	//*[local-name() = 'item']
Q33	//*[name() = 'svg:item']
Q35	//*[lang('it')]
Q37	/site/people/person[starts-with(name,'Ed')]
Q38	/site/regions/*/item/mailbox/mail[substring-before(date,'/') = '10']
Q39	/site/regions/*/item/mailbox/mail[substring-before(substring-after(date,'/'),'/') = '09']
Q40	/site/regions/*/item/mailbox/mail[substring-after(substring-after(date,'/'), '/') = '1998']
Q41	/site/regions/*/item/mailbox/mail[substring(date,7,2) = '20']
Q43	/site/people/person [string-length(translate(concat(address/street,address/city,address/country,address/zipcode)," ","")) > 30]
Q44	/site/open_auctions/open_auction[floor(sum(bidder/increase)) >= 70]
Q45	/site/open_auctions/open_auction[ceiling(sum(bidder/increase)) <= 70]
Q46	/site/open_auctions/open_auction[round((number(current) - number(initial)) div count(bidder)) > 8]
Q47	/site/people/person[boolean(emailaddress) = true() and not(boolean(homepage)) = false()]

Tabela 4: Consultas utilizadas durante os experimentos para o documento D2

Consultas XPath Documento D2	
Q3	//*[namespace::xlink]
Q4	//*[namespace::* = 'http://www.w3.org/1999/xlink']
Q5	//xlink:*
Q6	//*[namespace::svg]
Q7	//*[namespace::* = 'http://www.w3.org/2000/svg']
Q8	//svg:*
Q10	//*[local-name() = 'ellipse']
Q11	//*[name() = 'svg:ellipse']
Q12	//*[lang('it')]

7.2.2 TRADUÇÃO XPATH PROLOG

Todas as consultas foram automaticamente traduzidas para Prolog utilizando a abordagem proposta nessa dissertação. Uma funcionalidade especial foi desenvolvida na ferramenta XMLInference (LIMA *et al.*, 2012), de forma que dada uma lista de entrada contendo consultas XPath para um mesmo documento, o algoritmo de tradução automática é acionado múltiplas vezes devolvendo a tradução Prolog das respectivas consultas. Todas as consultas XPath utilizadas em ambos os documentos XML, juntamente com suas respectivas traduções Prolog podem ser analisadas nos Anexos D e E.

7.2.3 ANÁLISE DOS RESULTADOS

Documento D1. Os resultados das consultas selecionadas do documento XML D1 apresentaram as respostas esperadas, conforme indicado pela documentação do *benchmark* XPathMark. A Tabela 5 exibe as consultas consideradas no experimento e a capacidade de execução da técnica proposta e dos demais concorrentes. É possível perceber de forma clara o alto poder de expressão da abordagem Prolog deste trabalho, até mesmo frente a abordagens XPath nativas. Nenhuma abordagem XPath foi capaz de executar a consulta Q14. Além desta consulta, a abordagem Galax não foi capaz de executar as consultas Q23, Q44 e Q45.

Tabela 5: Capacidade de execução de consultas para o documento D1

Consultas	SAXON	GALAX	SEDNA	XP2PL
Q1	X	X	X	X
Q2	X	X	X	X
Q3	X	X	X	X
Q5	X	X	X	X
Q8	X	X	X	X
Q9	X	X	X	X
Q10	X	X	X	X
Q11	X	X	X	X
Q12	X	X	X	X
Q13	X	X	X	X
Q14				X
Q15	X	X	X	X
Q16	X	X	X	X
Q22	X	X	X	X
Q23	X		X	X
Q24	X	X	X	X
Q30	X	X	X	X
Q31	X	X	X	X
Q32	X	X	X	X
Q33	X	X	X	X
Q35	X	X	X	X
Q37	X	X	X	X
Q38	X	X	X	X
Q39	X	X	X	X
Q40	X	X	X	X
Q41	X	X	X	X
Q43	X	X	X	X
Q44	X		X	X
Q45	X		X	X
Q46	X	X	X	X
Q47	X	X	X	X

Documento D2. As consultas selecionadas do documento XML D2 também apresentaram o retorno esperado, conforme pode ser visualizado na Tabela 6, salvo uma pequena limitação acerca da impressão individual dos *namespaces* (Figura 88).

Tabela 6: Capacidade de execução das consultas para o documento D2

Consultas	SAXON	GALAX	SEDNA	XP2PL
Q3				X
Q4				X
Q5				X
Q6				X
Q7				X
Q8				X
Q10	X	X	X	X
Q11	X	X	X	X
Q12	X	X	X	X

Elemento svg:SVG:

```
1 <svg:SVG xmlns:svg="http://www.w3.org/2000/svg" svg:width="12cm" svg:height="10cm">
2   <svg:ellipse svg:rx="110" svg:ry="130"/>
3   <svg:rect svg:x="4cm" svg:y="1cm" svg:width="3cm" svg:height="6cm"/>
4 </svg:SVG>
```

Resposta esperada da consulta //*[namespace::svg]:

```
1 <svg:SVG xmlns:svg="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
svg:width="12cm" svg:height="10cm">
2       <svg:ellipse svg:rx="110" svg:ry="130"/>
3       <svg:rect svg:x="4cm" svg:y="1cm" svg:width="3cm" svg:height="6cm"/>
4 </svg:SVG>
5
6 <svg:ellipse xmlns:svg="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
svg:rx="110" svg:ry="130"/>
7 <svg:rect xmlns:svg="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
svg:x="4cm" svg:y="1cm" svg:width="3cm" svg:height="6cm"/>
```

Figura 88: Elemento *svg:SVG* e resultado esperado da XPath *//*[namespace::svg]*

Consultas do tipo *//*[namespace::* = 'XYZ']*, aonde o *namespace 'XYZ'* está atribuído a um elemento complexo, devem imprimir este elemento e todos os seus filhos, indicando este *namespace* de forma individual. A Figura 88 mostra um trecho do documento XML D2 fornecido pelo *benchmark*, juntamente com a resposta esperada para a consulta *//*[namespace::svg]*. Atualmente a abordagem imprime os elementos de forma correta, porém não atribui a declaração individual do *namespace* nos elementos filhos.

A utilização do XML Schema permitiu avaliar a pertinência de consultas antes mesmo de sua execução. Isso ocorre, por exemplo, na consulta Q5 – *//xlink:**. A consulta Q5 busca os elementos que possuem o *namespace xlink:* como prefixo. A tarefa de identificação dos possíveis elementos que se encaixam nesta consulta pode ser realizada com o XML Schema. No caso da consulta Q5 não há nenhum elemento que possua como prefixo o respectivo *namespace*.

Outra consulta que foi favorecida pelo XML Schema foi a Q12 – *//*[lang('it')]*. A análise preliminar do esquema permitiu verificar que somente os elementos *p* e *html* possuem o atributo *lang*. Desta forma somente esses elementos foram incluídos na consulta traduzida. A Figura 89 exibe a respectiva tradução.

```
findall(IDNODEORDER, ((IDNODEORDER = IDP, p(_, IDNODEORDER, _); IDNODEORDER = IDP,
p(_, IDNODEORDER)); html(IDNODEORDER)), LISTNODEORDER),
setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
member(IDNODEORDERSORTED,LISTNODEORDERSORTED), ((IDP = IDNODEORDERSORTED,
p_attribute_-xml-lang(IDP, IDP_ATTRIBUTE_XML_LANG, 'it'),
print_p(IDNOPARENT, IDNODEORDERSORTED)); (IDHTML = IDNODEORDERSORTED,
html_attribute_-xml-lang(IDHTML, IDHTML_ATTRIBUTE_XML_LANG, 'it'),
print_html(IDNODEORDERSORTED))).
```

Figura 89: Tradução Prolog para a consulta *//*[lang('it')]*

Discussão Geral. De modo geral, as consultas automaticamente traduzidas para Prolog funcionaram conforme indicado pelo benchmark utilizado nos experimentos. O poder de expressão proporcionado pela abordagem é alto quando comparado com outras abordagens XPath nativas. Quando comparado com outras abordagens Prolog, fica clara a superioridade deste trabalho acerca da gama de funções e operadores suportados. A

Tabela 7 exibe um comparativo com os principais concorrentes Prolog.

7.3 AVALIAÇÃO DE DESEMPENHO

7.3.1 GERAÇÃO DAS BASES

O *benchmark* XMark fornece um executável[1] que permite a geração de documentos de diversos tamanhos de acordo com um fator de escala informado como argumento. Este executável permitiu a geração de diversos documentos XML de tamanhos distintos, todos baseados na estrutura do documento D1. Como o documento D2 não é original do XMark, não foi possível utilizar o executável para gerar tais bases de tamanhos diferentes.

Para o documento D1 foram geradas 5 versões de documentos XML baseados em sua estrutura, com os tamanhos de 10 MB, 50 MB, 100 MB, 500 MB e 1 GB. Para cada

[1] XMark: http://www.xml-benchmark.org/downloads.html

documento XML gerado foi criada uma base de fatos correspondente utilizando a última versão do algoritmo de tradução de documentos do XMLInference.

Tabela 7: Abrangência de funções e operadores deste trabalho e das abordagens Prolog concorrentes

Funções e Operadores	Jiménez et al. (2008)	Santos et al.(2012)	Machado (2016)
/	X	X	X
//	X	X	X
=	X	X	X
>	X	X	X
<	X	X	X
>=	X	X	X
<=	X	X	X
*	X		X
parent::			X
following-sibling::			X
following::			X
preceding-sibling::			X
preceding::			X
namespace::			X
.			X
..	X		X
|	X		X
position()			X
last()			X
count()			X
local-name()			X
name()			X
lang()			X
starts-with()			X
substring-before()			X
substring-after()			X
substring()			X
translate()			X
concat()			X
floor()			X
ceiling()			X
sum()			X
round()			X

De posse das consultas do documento D1 obtidas no experimento anterior, documentos XML e bases de fatos, os cenários de execução foram montados. Cada uma das 31 consultas selecionadas para o documento D1 foi submetida às 5 versões de documentos XML ou base de fatos equivalentes, tomando as respectivas medições de tempo em nano segundos. Galax, Saxon e Sedna executaram as consultas originais XPath do *benchmark* sobre as versões dos documentos XML. Já no SWI-Prolog, foram utilizadas a base de fatos Prolog correspondente a cada uma das versões dos documentos XML utilizadas no experimento, além das consultas XPath do mesmo *benchmark* traduzidas para Prolog.

Para minimizar possíveis retardos de operações de interface, todas as execuções foram realizadas via terminal. *Scripts* foram desenvolvidos com o objetivo de automatizar os processos de execução e tomadas de tempo das rodadas de experimentos. A Figura 90 exibe um exemplo de *script* utilizado durante os experimentos para a execução de consultas na plataforma Saxon.

```
1 (cd Consultas/ && ls -tr -1 *.XPath) | while read file
2 do
3  baseName="XML1_12"
4  mkdir resultado_${baseName}
5  for i in 0 1 2 3 4 5 6 7 8 9 10
6   do
7    echo "****** Executing XPath inside file ${file}! ******"
8    output="${baseName}${file}_${i}"
9
10   # UNIX timestamp concatenated with nanoseconds
11   T="$(date +%s%N)"
12
13  java -cp saxon9ee.jar net.sf.saxon.Query -xsiloc:on -xsd:XML1.xsd -expand:on -sa -
s:Bases/${baseName}.xml -q:Consultas/${file} 2>> resultado_${baseName}/resultados.${output}.xml >
/dev/null
14
15   # Time interval in nanoseconds
16   T="$(($(date +%s%N)-T))"
17
18  echo -n "${T};" >> resultado_${baseName}/tempos_${baseName}.csv
19
20  echo "******** Fim da XPath ********\n\n"
21 done
22 echo "\n" >> resultado_${baseName}/tempos_${baseName}.csv
23 done
```

Figura 90: *Script* de execução e tomada de tempo de consultas para o interpretador Saxon

Para execução das consultas Prolog na máquina de execução SWI-Prolog, foi utilizada sua interface C++ (WIELEMAKER, 2002). A Figura 91 exibe trecho de código C++ referente à execução e tomada de tempo das consultas Prolog utilizando tal interface.

```
1  auto aStartTime = chrono::high_resolution_clock::now();
2  PlQuery q1("call", PlTermv(PlCompound(query.c_str())));
3  bool bDone = false;
4  while (q1.next_solution())
5  {
6   fflush(stdout);
7  }
8  auto aEndTime = chrono::high_resolution_clock::now();
9  std::ostringstream s;
10 s<< chrono::duration_cast<chrono::nanoseconds>(aEndTime - aStartTime).count();
```

Figura 91: Trecho de código C++ referente à execução e tomada de tempo de consultas Prolog

Cada consulta Prolog e XPath foi executada 11 vezes, descartando a primeira execução para evitar problemas relacionados à “partida a frio”. O tempo da efetiva impressão dos resultados de cada consulta na tela (uma operação sabidamente lenta) foi desconsiderado, através do redirecionamento do *output* para */dev/null*. É importante notar que apenas o tempo de print do sistema foi desconsiderado, e não o tempo da geração do resultado XML em si.

7.4 ANÁLISE DOS RESULTADOS

A análise acerca do desempenho da solução proposta frente a outras abordagens nativas XML se deu através da comparação dos tempos de execução coletados em cada cenário já descrito. Os valores dos tempos de execução, convertidos em milisegundos, obtidos durante os experimentos podem ser visualizados a seguir na Tabela 8, Tabela 9, Tabela 10, Tabela 11 e Tabela 12. De modo geral, é possível notar o bom desempenho da abordagem desenvolvida neste trabalho. Uma característica importante constatada durante estes experimentos está relacionada à escalibidade. Os interpretadores XML Saxon e Galax não conseguiram lidar com todos os tamanhos de documentos XML gerados. Quando testado com documentos XML de aproximadamente 500 MB, o interpretador Galax acusou problemas relacionados à pilha de execução. Já o interpretador Saxon apresentou problemas de estouro de memória quando executado com documentos de 1 GB. Por outro lado, a solução Prolog desenvolvida por este trabalho foi capaz de executar todas as consultas em todos os cenários propostos, demonstrando competitividade com o banco de dados XML Sedna.

Como pode ser observado nas tabelas a seguir, algumas consultas apresentam “N/A” como valor. Conforme já mencionado, apesar destas consultas terem sido executadas

conforme sua respectiva descrição no *benchmark*, alguns interpretadores e/ou banco de dados XML não conseguiram processá-las com sucesso.

Ainda sobre a disposição dos resultados, gráficos de barras agrupadas foram construídos com o objetivo de prover melhor visualização e análise. Cada cenário correspondente ao tamanho de documento XML utilizado, juntamente com sua respectiva base de fatos traduzida**.**

Tabela 8: Tempos de execução das consultas em milissegundos utilizando documento XML de 10 MB e sua respectiva tradução Prolog

Consulta	SAXON	GALAX	SEDNA	XP2PL
Q1	840,09	1064,63	118,9	325,97
Q2	864,89	988,76	69,95	2,66
Q3	804,16	899,34	19,87	24,78
Q5	813,6	2116,37	116,89	169,79
Q8	822,99	931,31	20,46	2,19
Q9	832,35	902,29	16,82	0,22
Q10	886,32	1130,32	164,69	341,15
Q11	860,68	975,6	67,97	2,74
Q12	853,39	2303,77	778,42	36,4
Q13	928,75	2911,91	1022,74	778,78
Q14	N/A	N/A	N/A	147,59
Q15	833,54	2311,8	795,42	5,48
Q16	795,29	2078,48	14,75	0
Q22	864,88	982,05	58,6	174,43
Q23	827,27	N/A	40,07	69,29
Q24	853,82	942,93	46,99	1461
Q30	834,88	912,6	20,89	28,26
Q31	855,44	979,38	45,58	151,05
Q32	936,29	2766,12	979,05	322,28
Q33	881,97	2588,03	976,29	0
Q35	870,89	2771,54	1253,42	0
Q37	799,28	904,18	19,95	1,19
Q38	842,84	920,96	30,32	8,85
Q39	857,9	925,32	27,16	10,56
Q40	865,1	935,53	43,47	28,7
Q41	878,04	940,53	50,53	42,12
Q43	858,48	968,85	48,85	67,79
Q44	910,07	N/A	58,81	166,96
Q45	877,64	N/A	57,35	135,27
Q46	873,68	1043,18	81,44	238,81
Q47	844,17	957,38	58,37	77,25

Tabela 9: Tempos de execução das consultas em milissegundos utilizando documento XML de 50 MB e sua respectiva tradução Prolog

Consulta	SAXON	GALAX	SEDNA	XP2PL
Q1	1280,17	4427,53	329,66	1424,16
Q2	1260,57	4084,46	171,76	13,09
Q3	1145,31	3746,79	36,2	116,75
Q5	1199,54	9444,26	435,82	720,82
Q8	1169,95	3844,29	35,9	15,32
Q9	1158,41	3738,64	18,1	0,63
Q10	1358,8	4673,12	649,55	1440,99
Q11	1254,36	4250,3	270,14	34,54
Q12	1239,03	10223,76	3290,27	158,37
Q13	1499,06	12682,18	4282,96	3390,04
Q14	N/A	N/A	N/A	679,27
Q15	1208,63	10253,67	3304,11	22,84
Q16	1154,05	9319,43	14,96	0
Q22	1258,8	4080,19	169,17	825,97
Q23	1210,43	N/A	116,28	322,59
Q24	1220,94	3915,96	127,88	25479,8
Q30	1159,35	3792,09	38,5	111,41
Q31	1255,42	4074,2	123,98	639,73
Q32	1446,12	12063,43	4020,45	1418,45
Q33	1303,37	11336,42	3944,9	0
Q35	1269,13	12035,14	4749,99	0
Q37	1129,46	3779,2	25,64	6,48
Q38	1212,83	3846,9	48,14	44,18
Q39	1206,77	3859,64	42,14	49,49
Q40	1265,77	3878,95	78,44	132,26
Q41	1257,98	3913,36	112,94	197,55
Q43	1250,97	4031,67	116,24	322,34
Q44	1323,09	N/A	150,3	718,84
Q45	1264,67	N/A	147,68	575,6
Q46	1294,78	4316,5	197,75	1025,07
Q47	1246,12	3976,38	143,07	378,72

Tabela 10: Tempos de execução das consultas em milissegundos utilizando documento XML de 100 MB e sua respectiva tradução Prolog

Consulta	SAXON	GALAX	SEDNA	XP2PL
Q1	1952,48	9490,23	638,41	2888,98
Q2	1865,76	8815,26	329,3	28,75
Q3	1710,7	8157,77	58,34	247,31
Q5	1793,51	20610,59	853,44	1464,58
Q8	1747,97	8336,04	57,33	37,51
Q9	1726,88	8135,49	22,04	1,3
Q10	1986,73	10013,84	1112,39	2921,82
Q11	1843,05	9391,69	740,76	89,44
Q12	1786,51	22143,12	6561,1	307,45
Q13	2232,83	27249,74	8452,96	6959,59
Q14	N/A	N/A	N/A	1383,73
Q15	1773,92	22275,83	6627,87	47,85
Q16	1712,63	20290,42	15,63	0
Q22	1864,91	8823,52	315,49	1882,59
Q23	1825,13	N/A	213,06	658,11
Q24	1822,78	8538,02	244,27	101715,98
Q30	1742,9	8233,13	61,87	216,85
Q31	1839,74	8783,88	233,21	1226,86
Q32	2196,53	25874,38	7995,37	2733,23
Q33	1931,02	24481,54	7929,54	0
Q35	1929,7	25852,56	9266,46	0
Q37	1739,6	8241,33	38	14,88
Q38	1776,91	8402,47	78,82	89,15
Q39	1798,65	8399,29	67,55	98,58
Q40	1820,37	8448,17	151,58	262,68
Q41	1858,6	8515,96	207,59	386,56
Q43	1854,22	8715,22	215,78	643,47
Q44	1936,84	N/A	284,36	1391,99
Q45	1899,17	N/A	287,24	1121
Q46	1934,11	9301,61	393,05	2010,93
Q47	1817,8	8631,11	270,43	727,22

Tabela 11: Tempos de execução das consultas em milissegundos utilizando documento XML de 500 MB e sua respectiva tradução Prolog

Consulta	SAXON	GALAX	SEDNA	XP2PL
Q1	6856,25	N/A	2897,51	14657,8
Q2	6563,16	N/A	1503,36	168,06
Q3	5849,23	N/A	207,54	1291,14
Q5	5738,52	N/A	4032,05	7452,49
Q8	5737,12	N/A	221,21	216,73
Q9	6024,92	N/A	49,22	7,87
Q10	7197,23	N/A	5130,38	14736,87
Q11	6325,88	N/A	207,62	213,76
Q12	6306,68	N/A	37570,56	1537,18
Q13	8191,89	N/A	46570,78	34488,87
Q14	N/A	N/A	N/A	7016,22
Q15	6091,93	N/A	37975,96	241,53
Q16	5800,55	N/A	14,97	0
Q22	6621,32	N/A	1469,48	16994,07
Q23	6409,62	N/A	972,1	3365
Q24	6228,96	N/A	1087,1	3645272,86
Q30	5855,04	N/A	261,16	1069,59
Q31	6466,07	N/A	1109,32	5999,83
Q32	8666,48	N/A	44832,9	13861,13
Q33	7760,78	N/A	44147,27	0
Q35	7309,25	N/A	51337,87	0
Q37	5901,81	N/A	149,45	78,66
Q38	6378,67	N/A	328,6	462,78
Q39	6356,86	N/A	282,95	499,58
Q40	6479,74	N/A	611,54	1323,52
Q41	6466,34	N/A	945,21	1968,61
Q43	6121,78	N/A	954,45	3725,92
Q44	6349,09	N/A	1330,08	6892,66
Q45	6245,26	N/A	1311,9	5630,26
Q46	6576,25	N/A	1848,67	9920,34
Q47	6178,08	N/A	1243,38	3735,76

Tabela 12: Tempos de execução das consultas em milissegundos utilizando documento XML de 1 GB e sua respectiva tradução Prolog

Consultas	SAXON	GALAX	SEDNA	XP2PL
Q1	N/A	N/A	6528,87	33736,41
Q2	N/A	N/A	3332,26	388,07
Q3	N/A	N/A	468,58	3037,66
Q5	N/A	N/A	9226,53	17128,61
Q8	N/A	N/A	488,12	520,32
Q9	N/A	N/A	91,38	23,71
Q10	N/A	N/A	11761,6	33931,17
Q11	N/A	N/A	15171,18	1775,56
Q12	N/A	N/A	105094,25	3532,29
Q13	N/A	N/A	126008,54	80210,96
Q14	N/A	N/A	N/A	16467,77
Q15	N/A	N/A	106364,57	557,45
Q16	N/A	N/A	15,1	0
Q22	N/A	N/A	3362,56	74682,2
Q23	N/A	N/A	2133,59	7831,04
Q24	N/A	N/A	2426,47	24610744,05
Q30	N/A	N/A	582,1	2475,35
Q31	N/A	N/A	2475,09	14052,37
Q32	N/A	N/A	121731,83	32453,66
Q33	N/A	N/A	121060,14	0
Q35	N/A	N/A	136522,24	0
Q37	N/A	N/A	311,73	197,92
Q38	N/A	N/A	737,87	1065,72
Q39	N/A	N/A	631,04	1174,62
Q40	N/A	N/A	1345,27	3112,92
Q41	N/A	N/A	2077,45	4567,57
Q43	N/A	N/A	2176,75	9594,66
Q44	N/A	N/A	2949,05	16558,9
Q45	N/A	N/A	2943,08	14012,85
Q46	N/A	N/A	4172,98	24280,71
Q47	N/A	N/A	2766,53	8908,8

As consultas que apresentaram algum erro por parte dos interpretadores XPath ou banco de dados XML (Q14, Q23, Q44 e Q45) foram desconsideradas nessa análise gráfica e serão analisadas em separado.

Ainda dentro deste contexto, a consulta Q24 apresenta um caso particular. Seu desempenho Prolog não foi satisfatório durante a realização dos experimentos. Desta forma, de modo a não comprometer a visualização dos demais resultados, seus respectivos tempos de execução serão analisados à parte e tembém não foram incluídos nos gráficos gerados.

Documento D1 de 10MB. Analisando somente a Figura 92 é possível observar que a abordagem XP2PL possui desempenho superior na maioria das consultas deste cenário, seguido pelo banco de dados XML Sedna. Dos interpretadores XPath, o Galax foi o que obteve o pior desempenho.

Das 10 consultas onde o Sedna obteve o melhor desempenho (Q1, Q3, Q5, Q10, Q22, Q30, Q31, Q43, Q46, Q47), quatro se destacam pelo contraste em relação ao tempo de execução do XP2PL, que apresentou tempos mais elevados (Q1, Q22, Q31, Q46 - mais do que o dobro do tempo obtido pelo Sedna). Analisando de forma detalhada, a consulta Prolog Q1 utiliza o operador de seleção múltipla ("*") sem nenhum filtro associado, havendo desta forma diversas tentativas de casamento com o identificador pai do elemento *item*. A consulta Q22 realiza a união de duas consultas sobre o elemento *item*. Desta forma a sua respectiva tradução Prolog utiliza o operador OU ";" para executar ambas as consultas. Tal característica possivelmente elevou seu tempo de resposta. Por fim, as consultas Q31 e Q46 além de envolverem operações matemáticas, fazem uso da função *count* sobre o elemento *bidder* que possui granularidade muito alta.

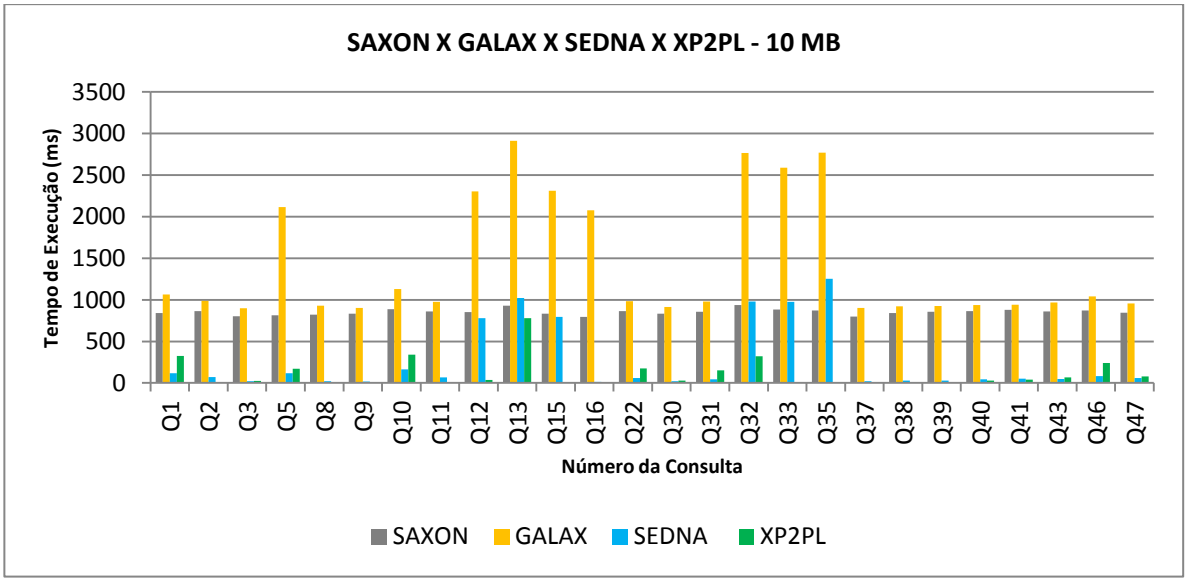


Figura 92: Comparação de tempos de execução entre Saxon, Galax, Sedna e XP2PL, para base de fatos e documento XML D1 de 10 MB

Ao contrário dessas quatro, algumas consultas Prolog traduzidas automaticamente obtiveram um ótimo desempenho quando comparadas a todas as outras abordagens. Neste contexto, a utilização do XML Schema permitiu ao algoritmo de tradução automática antecipar se uma consulta XPath tem a possibilidade de ser executada com sucesso ou não.

As consultas Q16, Q33, Q35 se beneficiaram desta técnica, por envolver elementos que não estão descritos no XML Schema ou ainda não possuirem o atributo ou *namespace*

informado. Dessa forma, a consulta não necessita ser executada, e retorna *false* (o equivalente a uma resposta vazia).

Outro ponto observado é que algumas consultas que envolvem processamento de texto obtiveram bons resultados neste cenário. As consultas Q37, Q38, Q39 e Q40 utilizam funções Prolog criadas especificamente para lidar com *strings*. Ainda que algumas destas consultas façam referências a elementos que possuem granularidade alta, a utilização de filtros envolvendo conteúdo textual forneceu tempos de resposta superiores às demais abordagens. Já a consulta Q2 delimita o caminho completo até o elemento *keyword*, que pode possuir referências cíclicas. Desta forma a sua respectiva tradução Prolog se beneficiou desta característica, poupando a máquina de execução Prolog da utilização de funções extras e tentativas de casamento desnecessários com outros elementos.

A consulta Prolog Q12 utiliza um filtro de atributo com um valor específico, diminuindo o número de ocorrências do elemento *item* recuperados pela função *findall*, conforme ilustrado na Figura 93.

Consulta XPath:

```
//item[@featured='yes']
```

Consulta Prolog:

```
findall(IDNODEORDER, (item(IDNOPARENT, IDITEM), FEATURED = 'yes',
item_attribute_featured(IDITEM, IDFEATURED, FEATURED), nonvar(IDITEM), IDNODEORDER = IDITEM,
item(IDNOPARENT, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER),
LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED),
 IDITEM = IDNODEORDERSORTED, ( print_item(IDNOPARENT, IDITEM)).
```

Figura 93: Consulta Q12 e sua respectiva tradução Prolog

Por fim as consultas Q9, Q11 e Q15 também se destacaram positivamente por seus resultados. A consulta Q15 busca o elemento simples "*increase*". O filtro de comparação numérica, que objetiva selecionar um elemento simples com pouco conteúdo textual, possivelmente manteve seu tempo de execução baixo. Já as consultas Q9 e Q11 foram beneficiadas pelos filtros sobre o atributo *id*, que descartou os identificadores indesejados, poupando assim a função Prolog de verificação dos elementos irmãos *preceding*. Em resumo, para o cenário de 10MB, das 26 consultas analisadas, o XP2PL apresentou melhor resultado em 16 (61.5%).

Documento D1 de 50MB. A Figura 94 exibe tempos de execução para o documento XML e base de fatos de 50 MB.

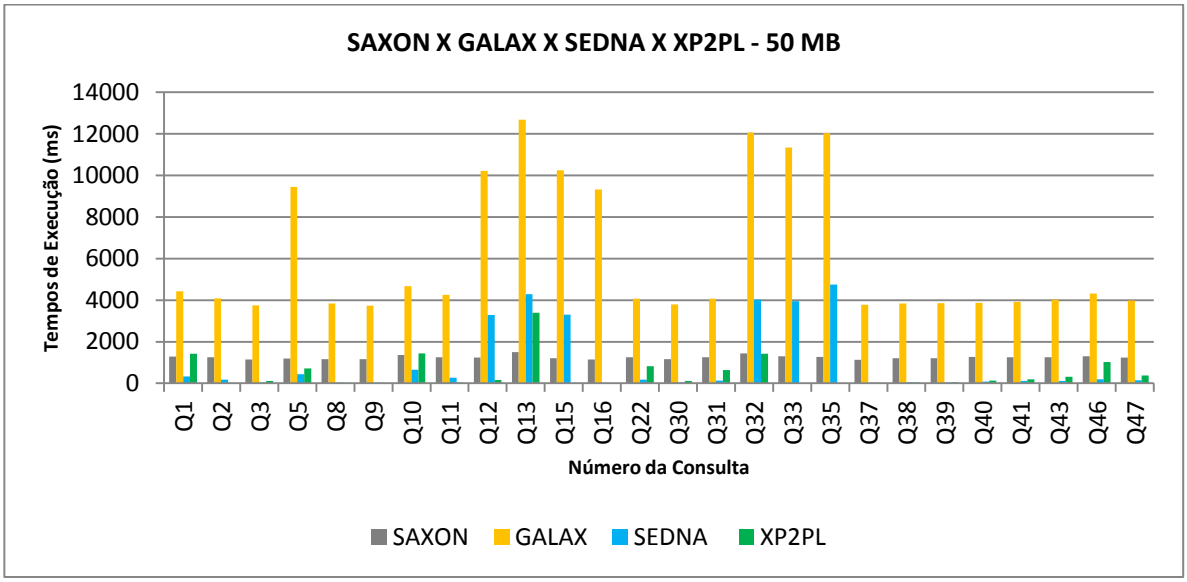


Figura 94: Comparação de tempos de execução entre Saxon, Galax, Sedna e XP2PL, para base de fatos e documento XML D1 de 50 MB

Podemos observar no resultado do gráfico referente à utilização da base de fatos e documento XML de 50 MB, que quantitativamente o Sedna possui o maior número de consultas com o melhor tempo de execução, seguido de perto pelo XP2PL.

Neste cenário, as consultas Q39, Q40 e Q41 passaram a possuir seu melhor tempo de execução associado ao Sedna (anteriormente o XP2PL apresentava o melhor tempo). Vale mencionar que a diferença dos tempos de execução entre ambos já era pouco expressiva, e manteve-se de forma semelhante. Os resultados das consultas Prolog Q3, Q30 e Q31 distanciaram-se dos obtidos pelo banco de dados Sedna. Isso foi causado pelo aumento do número de ocorrências dos elementos *bidder* e *keyword*. De forma semelhante, a consulta Q10, que emprega o operador de seleção múltipla ("*"), teve seu tempo aumentado. A utilização do filtro sobre o atributo *id* não trouxe ganho aparente por conta da utilização do operador *following*. Conforme mencionado no capítulo anterior, a tradução Prolog deste operador fez uso da função *obtainDescendents_item* que verifica os identificadores dos elementos descendentes do elemento *item* em questão. O fato do elemento *item* possuir uma quantidade razoável de elementos filhos, muitos deles complexos, possivelmente elevou os tempos de execução desta consulta.

Ainda neste cenário, podemos observar que o interpretador Saxon possui bom desempenho em algumas consultas, chegando a superar Sedna e XP2PL (Q13). Comparando

somente com a abordagem Prolog, o Saxon foi superior nas consultas Q1, Q10 e Q13. Em resumo, para o cenário de 50MB, das 26 consultas analisadas, o XP2PL apresentou melhor resultado em 12 (46%).

Documento D1 de 100MB. A Figura 95 exibe os tempos de execução para o documento XML e base de fatos de 100 MB.

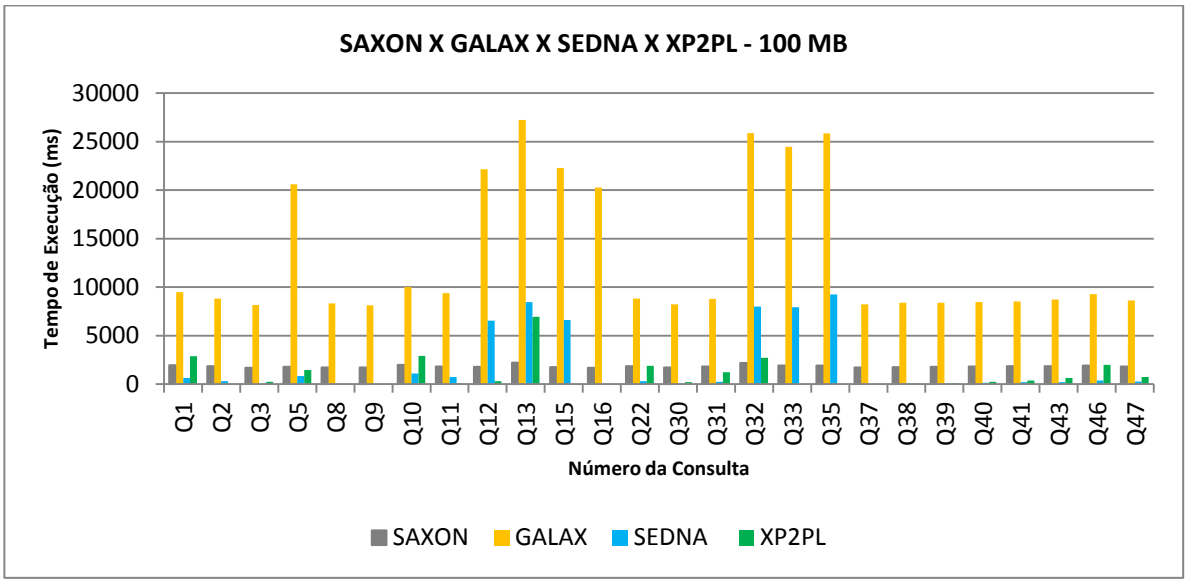


Figura 95: Comparação de tempos de execução entre Saxon, Galax, Sedna e XP2PL, para base de fatos e documento XML D1 de 100 MB

Considerando as consultas dos gráficos do cenário de 100 MB, o banco de dados Sedna continuou superior quantitativamente, contando com 14 consultas com o melhor tempo de execução. Os tempos de execução dos 2 primeiros colocados (Sedna e XP2PL) são semelhantes em sua maioria. A consulta Q38 passou a ter seu melhor tempo de execução associado ao Sedna, por uma pequena diferença (10.3 ms). De forma semelhante, a consulta Q32 obteve seu melhor resultado utilizando o interpretador Saxon, sendo superior ao XP2PL. O interpretador XPath Saxon ainda foi superior à abordagem Prolog nas consultas Q1, Q10, Q13, Q22 e Q46. Em resumo, para o cenário de 100MB, das 26 consultas analisadas, o XP2PL apresentou melhor resultado em 10 (38,5%).

Documento D1 de 500MB. Podemos observar na Figura 96, que o interpretador Galax não possui tempos de execução para o cenário de documento XML e base de fatos de 500 MB. Conforme mencionado anteriormente, o motivo da ausência destas medições é que o Galax não suportou execuções com documentos de 500 MB, apresentando erros de *stack overflow*.

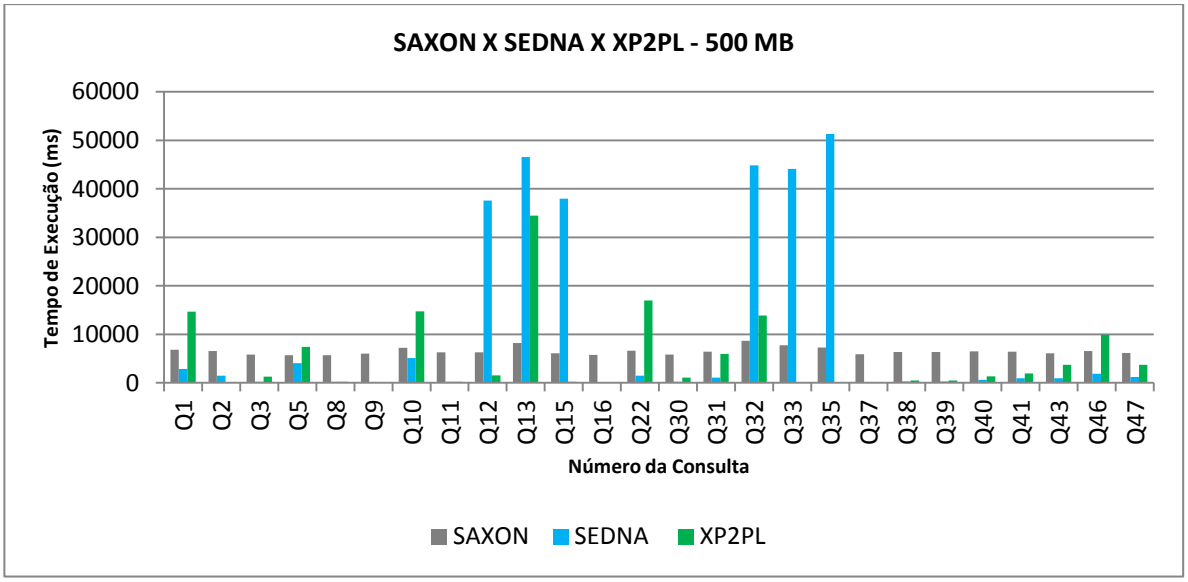


Figura 96: Comparação de tempos de execução entre Saxon, Sedna e XP2PL, para base de fatos e documento XML D1 de 500 MB

Neste cenário, inesperadamente neste cenário a consulta Q11, quando executada pelo banco de dados Sedna, apresentou tempo de resposta inferior à sua respectiva execução no cenário anterior. Analisando de forma mais detalhada foi possível verificar que o neste cenário não são retornados elementos como resultado, ao contrário do cenário anterior de 100 MB. Este fato deve-se à aleatoriedade na geração dos documentos XML construídos pelo executável presente no *benchmark* XMark. Possivelmente a estruturação da informação, beneficia o banco de dados XML Sedna em relação aos demais executores, que por sua vez lidam diretamente com o arquivo XML ou Prolog de grande tamanho. A consulta Q1 apresentou aumento no tempo de resposta, possivelmente associado à elevação da cardinalidade dos elementos *item*. Ainda neste cenário, o interpretador XPath Saxon foi superior à abordagem Prolog nas consultas Q1, Q5, Q10, Q13, Q22, Q32 e Q46. Em resumo, para o cenário de 500MB, das 26 consultas analisadas, o XP2PL apresentou melhor resultado em 9 (34,6%).

Documento D1 de 1GB. O cenário que utiliza documento XML e base de fatos de 1 GB não possui medições do interpretador XPath Saxon pois o mesmo apresentou erros de estouro de memória quando submetido ao documento XML de 1 GB. A Figura 97 exibe os tempos de execução deste cenário.

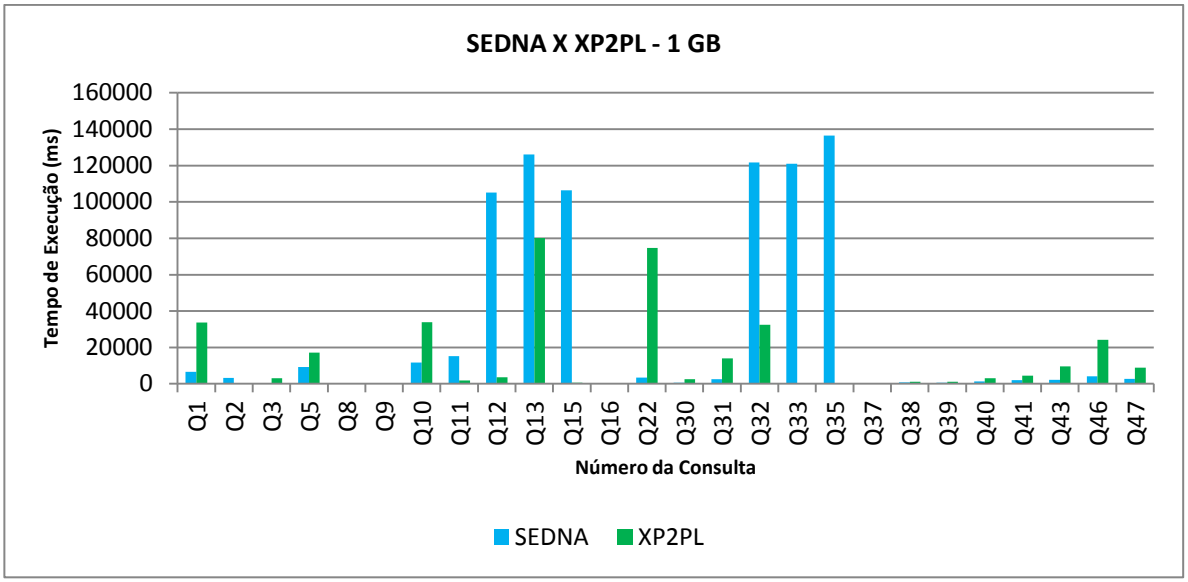


Figura 97: Comparação de tempos de execução entre Sedna e XP2PL, para base de fatos e documento XML D1 de 1 GB

Quantitativamente neste cenário o banco de dados Sedna permanece com o maior número de consultas com melhor desempenho. Algumas consultas Prolog, além das já mencionadas anteriormente, distanciaram seus tempos de execução em relação ao Sedna. As consultas de manipulação de *strings* Q40, Q41, Q43 e Q47 se encaixam neste grupo, assim como a consulta Q5. Esta última também utiliza o operador de seleção múltipla "*". Ainda que alguns identificadores pai possuam o valor igual a 0 (Figura 98, em negrito), há diversas tentativas de casamento. Ainda neste cenário a ausência do interpretador XPath Saxon fez com que o XP2PL retomasse a liderança dos tempos de execução nas consultas Q13 e Q32. Em resumo, para o cenário de 1GB, das 26 consultas leves, o XP2PL apresentou melhor resultado em 11 (42,3%).

Consulta XPath:

/site/regions/*/item[parent::namerica or parent::samerica]

Consulta Prolog:

```
findall(IDNODEORDER, (IDAFRICA is 0 , IDASIA is 0 , IDAUSTRALIA is 0 , IDEUROPE is 0 , site(IDSITE),
regions(IDSITE, IDREGIONS),
findall(IDWILD,(IDWILD = IDAFRICA, africa(IDREGIONS, IDAFRICA));
(IDWILD = IDASIA, asia(IDREGIONS, IDASIA));
(IDWILD = IDAUSTRALIA, australia(IDREGIONS, IDAUSTRALIA));
 (IDWILD = IDEUROPE, europe(IDREGIONS, IDEUROPE));
(IDWILD = IDNAMERICA, namerica(IDREGIONS,IDNAMERICA));
(IDWILD = IDSAMERICA, samerica(IDREGIONS, IDSAMERICA)), LISTWILD),
member(IDWILD, LISTWILD), item(IDWILD, IDITEM), nonvar(IDITEM),
IDNODEORDER = IDITEM, item(_, IDNODEORDER)), LISTNODEORDER), setof(X,member(X, LISTNODEORDER),
LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED),
IDITEM = IDNODEORDERSORTED, (print_item(IDNOGRANDPARENT, IDITEM)).
```

Figura 98: Consulta Q5 e sua respectiva tradução Prolog.

Casos Especiais. Conforme já mencionando, algumas consultas não foram executadas com sucesso pelos interpretadores XPath e/ou banco de dados XML. A consulta Q14 do *benchmark* (*//person[namespace::xml]*), não conseguiu ser executada por nenhuma das soluções XML nativas. A Tabela 13 exibe os respectivos tempos de execução para o XP2PL.

Tabela 13: Tempos de execução das consultas em milissegundos para a consulta Q14 – XP2PL

Tamanho da Base	Tempo de execução
10 MB	147,587
50 MB	679,27
100 MB	1383,734
500 MB	7016,219
1 GB	16467,78

As consultas Q23, Q44 e Q45 também não foram executadas com sucesso pelo interpretador XPath Galax. A Tabela 14 exibe os resultados para a consulta Q23. É possível observar que a partir do cenário de 100 MB, há um maior distanciamento entre os tempos do Sedna e XP2PL. Semelhantemente a diversas outras consultas, a abordagem Prolog possui tempo inferior quando comparada com o interpretador XPath Saxon.

Tabela 14: Tempos de execução das consultas em milissegundos para a consulta Q23

	SAXON	SEDNA	XP2PL
10 MB	827,273	40,073	69,293
50 MB	1210,432	116,282	322,589
100 MB	1825,131	213,064	658,11
500 MB	6409,625	972,098	3364,999
1 GB	N/A	2133,59	7831,039

Os tempos de reposta para as consultas Q44 e Q45 podem ser visualizados na Tabela 15 e Tabela 16. A tradução Prolog deste tipo de consulta utiliza a função *findall* para recuperar todos os valores do elemento em questão, e em seguida percorre a lista de valores realizando o somatório. Esta característica aliada a razoável granularidade do elemento *increase* colaborou na elevação destes tempos de resposta.

Tabela 15: Tempos de execução das consultas em milissegundos para a consulta Q44

	SAXON	SEDNA	XP2PL
10 MB	910,07	58,808	166,964
50 MB	1323,094	150,299	718,835
100 MB	1936,845	284,363	1391,994
500 MB	6349,091	1330,08	6892,662
1 GB	N/A	2949,051	16558,9

Tabela 16: Tempos de execução das consultas em milissegundos para a consulta Q45

	SAXON	SEDNA	XP2PL
10 MB	877,643	57,35	135,269
50 MB	1264,666	147,682	575,598
100 MB	1899,173	287,245	1120,996
500 MB	6245,256	1311,902	5630,258
1 GB	N/A	2943,084	14012,85

A consulta Q24 apresentou um desempenho aquém do esperado. A consulta original juntamente com sua respectiva tradução Prolog pode ser visualizada na Figura 99. O objetivo desta consulta é trazer como resultado os elementos *person* que não possuem o elemento *homepage* como filho.

Consulta XPath:
/site/people/person[not(homepage)]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), people(IDSITE, IDPEOPLE), person(IDPEOPLE, IDPERSON), findall(IDPERSON, person(IDPEOPLE, IDPERSON), LISTFALSE2), findall(IDPERSON3, (person(IDPEOPLE, IDPERSON3), homepage(IDPERSON3, _, HOMEPAGE)), LISTFALSE3), minus(LISTFALSE2, LISTFALSE3, LISTFALSE4), member(IDPERSON, LISTFALSE4), nonvar(IDPERSON), IDNODEORDER = IDPERSON, person(IDPEOPLE, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDPERSON = IDNODEORDERSORTED, (print_person(IDPEOPLE, IDPERSON)).

Figura 99: Consulta XPath Q24 e sua tradução Prolog.

Seguindo o paradigma da linguagem Prolog, a solução genérica adotada para este tipo de consulta foi a obtenção da diferença entre conjuntos que possuem os identificadores do elemento requerido. Mais especificamente, todos os identificadores do elemento *person* são obtidos através da função *findall* e atribuídos a uma lista. Em seguida todos os identificadores dos elementos *person* que possuem um elemento *homepage* como filho são colocados em uma segunda lista. Assim, subtraindo a segunda lista da primeira, obtemos como resultado os identificadores dos elementos *person* que não possuem *homepage*.

Para a base de fatos de 10 MB a consulta apresenta tempos de execução razoáveis quando comparados ao interpretador Galax, penúltimo colocado. Porém conforme a base de fatos Prolog cresce, o desempenho se distancia de maneira acentuada dos demais concorrentes. Estes resultados devem-se, possivelmente, à granularidade do elemento *person*. Este tipo de elemento é bastante referenciado no documento XML D1, evoluindo, por exemplo, de 2805 ocorrências no documento de 10 MB para 11475 ocorrências no documento de 50 MB. Esta característica, aliada à necessidade da obtenção de todos os identificadores do elemento *person* para realizar a operação de subtração, degradaram o desempenho desta consulta. Os seus tempos de execução são exibidos na Tabela 17.

Tabela 17: Tempos de execução em milissegundos para a consulta Q24

	SAXON	GALAX	SEDNA	XP2PL
10 MB	853,819	942,931	46,995	1461,005
50 MB	1220,94	3915,963	127,884	25479,8
100 MB	1822,781	8538,022	244,267	101716
500 MB	6228,957	N/A	1087,1	3645273
1 GB	N/A	N/A	2426,473	24610744

Discussão Geral. De modo geral as consultas automaticamente traduzidas para Prolog obtiveram bons resultados durante os experimentos realizados com o documento XML D1. A Tabela 18 destaca, com um "*X*"em cada cenário, as consultas aonde a abordagem Prolog foi superior a todas às demais. Para documentos XML de aproximadamente 10 MB, o que cobre grande parte do universo habitual de documentos disponíveis na *Web*, o desempenho da abordagem Prolog foi superior a todas as abordagens analisadas, na maioria das consultas (17/31, o que significa dizer que o XP2PL obteve o melhor resultado em 54,8% das consultas).

As consultas Q16, Q33 e Q35 foram beneficiadas pela técnica de verificação do XML Schema do documento, sendo superior em todos os cenários. Mesmo consultas que não foram influenciadas pelo conhecimento prévio sobre a estrutura do documento através do XML Schema, produziram resultados melhores do que o banco de dados nativo XML (Q2, Q9, Q12, Q15, Q33, Q35 e Q37).

A consulta Q2 possivelmente foi beneficiada por explicitar o caminho completo até o elemento *text*, que pode estar presente em vários elementos complexos do documento. A consulta Q37 também obteve bom desempenho, aplicando um filtro textual simples sobre o elemento não complexo *name* (*starts-with(name,'Ed')*). Uma função Prolog equivalente à função XPath *starts-with* foi desenvolida, e possui a assinatura *startsWith(STRING1, STRING2)*. A função Prolog *startsWith* transforma as strings de entrada em listas de caracteres, comparando-as em seguida. Caso um dos caracteres seja diferente, e ainda haja elementos a comparar na variável *STRING2*, a execução é abortada retorando *false*.

As consultas com filtros entre elementos irmãos (Q8 e Q9) apresentaram resultados favoráveis, com a consulta Q9 ganhando dos demais concorrentes em todos os resultados. Na consulta Q9 o filtro *@id='open_auction0'* restringe o resultado a uma única ocorrência do elemento *open_auction*, delimitando assim o número total de tentativas de casamento na consulta. De forma semelhante, a utlilização de um filtro com o atributo *featured* na consulta

Q12 (*@featured='yes'*) reduziu o conjunto total de casamentos possíveis para o elemento *item*, que possui granularidade considerável. Já a consulta Q14 foi executada com sucesso somente na máquina de execução Prolog, não possuindo tempos nas abordagens avaliadas. Na consulta Q15, a comparação do conteúdo numérico do elemento simples *increase* possivelmente beneficiou seu tempo de resposta.

De forma resumida, podemos citar algumas características das consultas aonde a abordagem Prolog foi superior. Consultas que utilizam o XML Schema para verificar a presença de atributos ou elementos sob um determinado *namespace* foram beneficiadas pela simplificação da consulta Prolog realizada pelo algoritmo de tradução. De maneira semelhante, consultas que utilizam filtros e/ou possuem baixa frequência da aplicação da função Prolog *findall*, possuem tempos de execução menores. De maneira contrária, é possível verificar as características das consultas aonde a abordagem Prolog foi inferior. A alta cardinalidade dos elementos envolvidos nas consultas, e/ou a alta frequência da aplicação da função Prolog *findall*, aumenta o número de inferências realizadas pela máquina de execução Prolog, impactando diretamente nos tempos de resposta.

Na medida em que a base cresce, a maioria das abordagens avaliadas deixa de conseguir executar as consultas por problemas de estouro de memória. De fato, apenas o Sedna e a abordagem proposta nessa dissertação conseguem executar com sucesso todas as consultas na base de 1GB. No entanto, nessa base, as consultas executadas no Sedna tiveram desempenho superior, em sua maioria (20/31 consultas, o que significa dizer que o Sedna obteve desempenho superior em 64,5% das consultas). O XP2PL obteve melhor resultado em 35,5% das consultas nessa base. Das 5 consultas que consideramos casos especiais, o XP2PL obtém o melhor desempenho em apenas uma (Q14). De fato, nossa abordagem é a única que consegue executar essa consulta com sucesso.

É válido mencionar que mesmo não imprimindo os resultados, o tempo da função de impressão está incluso em todas as consultas Prolog. A função de impressão é responsável por ordenar os identificadores obtidos na consulta principal e imprimir recursivamente o elemento alvo, que por sua vez executa a função de impressão para seus elementos descendentes. Esta característica, aliada à estrutura centrada em elementos textuais do documento D1, pode ter contribuído de forma significativa nos tempos de execução das consultas Prolog.

Tabela 18: Consultas por cenário aonde a abordagem XP2PL foi superior

	10 MB	50 MB	100 MB	500 MB	1 GB
Q1					
Q2	X	X	X	X	X
Q3					
Q5					
Q8	X	X	X	X	
Q9	X	X	X	X	X
Q10					
Q11	X	X	X		X
Q12	X	X	X	X	X
Q13	X				X
Q14	X	X	X	X	X
Q15	X	X	X	X	X
Q16	X	X	X	X	X
Q22					
Q23					
Q24					
Q30					
Q31					
Q32	X	X			
Q33	X	X	X	X	X
Q35	X	X	X	X	X
Q37	X	X	X	X	X
Q38	X	X			
Q39	X				
Q40	X				
Q41	X				
Q43					
Q44					
Q45					
Q46					
Q47					

7.5 AMEAÇAS À VALIDADE

Apesar de toda precaução e critérios adotados acerca dos procedimentos realizados durante a execução dos experimentos, é importante mencionar aspectos que podem influenciar os resultados obtidos.

As medições de tempo nas plataformas nativas XML foram tomadas por funções distintas das funções de medição de tempo utilizadas nos experimentos Prolog. Para as plataformas nativas XML foi utilizada a função *date*, nativa do SO *Unix*. Já para as consultas Prolog, por conta da utilização da *API* C++ do SWI-Prolog, foi utilizada a função *chrono::high_resolution_clock::now()* oriunda da biblioteca *chrono* do C++ 11. Caso a função de medição nativa do SO *Unix* fosse utilizada, o tempo de carregamento da base de

fatos seria incluso no tempo total das consultas Prolog, prejudicando assim a análise do desempenho. É importante ressaltar que as medições dos tempos de execução foram realizadas em *nano* segundos em ambas as abordagens. Esta decisão deve minimizar quaisquer eventuais nano diferenças entre as plataformas.

A abordagem XMLInference utiliza o XML Schema do documento durante a avaliação e tradução da consulta XPath (*schema awareness*). Para obter o mesmo comportamento em todas as demais abordagens, iniciou-se a pesquisa acerca da execução das consultas com o conhecimento do esquema nas plataformas XML nativas selecionadas. Dentre as abordagens XML nativas escolhidas, o interpretador Saxon foi o único aonde foi possível verificar esta característica. A versão *Enterprise*, utilizada nos experimentos, permite informar o esquema do documento durante a execução de uma consulta. É válido mencionar que esta opção não pareceu ser decisiva na maioria dos tempos de resposta observados para este interpretador XPath.

Por fim, a nomenclatura dos predicados de elementos complexos e sem filhos foi alterada posteriormente aos experimentos realizados. A modificação somente tornou-se necessária durante testes de completude, realizados com um novo documento que não pertencia ao *becnhmark* XPathMark. Esta adequação do algortimo de tradução não impacta nos tempos de resposta das consultas Prolog. As conclusões e contribuições relevantes deste trabalho são descritas no próximo capítulo.

Capítulo 8 – CONCLUSÃO

8.1 CONTRIBUIÇÕES

Este trabalho apresentou uma abordagem de tradução automática de consultas XPath para Prolog. Dentro deste contexto, diversas funções e operadores XPath foram traduzidos, permitindo ao usuário final uma grande expressividade no momento de criação de suas consultas. Os respectivos resultados Prolog, obtidos na execução destas consultas sobre bases de fatos, foram retraduzidos para o formato original XML. A solução, nomeada XP2PL, foi implementada na ferramenta XMLInference (LIMA *et al.*, 2012), que por sua vez foi totalmente remodelada.

Experimentos foram realizados com o *benchmark* XPathMark, criado especificamente para consultas XPath. As consultas deste *benchmark* foram utilizadas para análise da expressividade e desempenho da solução proposta, comparando seus resultados e respectivos tempos de execução com um banco de dados XML nativo e dois interpretadores XPath bastante conhecidos na área. Diversos tamanhos de documentos XML foram utilizados para testar a robustez da solução proposta.

Os resultados obtidos comprovaram a eficácia do processo de tradução, aonde os tempos de execução desta abordagem se mostraram competitivos, sendo superiores em muitos casos. Os resultados relativos à robustez também foram positivos, pois todas as consultas Prolog foram executadas com sucesso em bases de fatos de diferentes tamanhos. De forma contrária, os interpretadores XML utilizados suportaram tamanhos de documentos XML de no máximo 500 MB.

Embora não tenha sido possível comparar detalhadamente o desempenho do algoritmo proposto por este trabalho com seu semelhante mais próximo (o autor não respondeu à tentativa de contato realizada por e-mail), é possível realizar uma análise pelos resultados obtidos. Considerando um documento XML inspirado no trabalho de Almendros-Jiménez *et al*. (2008), e uma máquina contemporânea à época em que o artigo foi publicado (o autor não menciona os detalhes sobre o ambiente de execução dos seus experimentos), foi possível realizar um breve comparativo. A Tabela 19 exibe os tempos de resposta de cada abordagem considerando o tempo total das etapas de análise e exibição dos resultados para a consulta */books/book/title* em documento XML de 512 KB.

Tabela 19: Comparativo de tempo de resposta com a abordagem de Almendros-Jiménez *et al*. (2008)

Abordagem	Tempo total
Jiménez et al. (2008)	9,75 sg
XP2PL (2016)	0,89 sg

Comparando os tempos de resposta presentes na tabela é possível perceber a superioridade do desemplenho desta abordagem frente ao que foi publicado por Almendros-Jiménez *et al*. (2008). Além disso, os demais resultados apresentados por Almendros-Jiménez *et al*. (2008) utilizam consultas simplificadas em documentos XML de no máximo 1 MB, enquanto este trabalho realizou experimentos com consultas complexas de um *benchmark* em documentos que variam de 10 MB a até 1 GB. Além disso, os tempos de execução descritos por Almendros-Jiménez *et al*. (2008) chegam na casa dos minutos para documentos de 1 MB, bastante superiores aos demonstrados por este trabalho. Desta forma esta abordagem reafirma sua eficiência mesmo sem prover índices de apoio, ou reescrita da consulta em função de análise prévia de seu impacto. Do ponto de vista da expressividade do algoritmo de tradução, a abordagem proposta fornece uma gama de funções, operadores e possibilidades de filtros que não são possíveis de serem realizados pelo algoritmo de Almendros-Jiménez *et al*. (2008).

Após todas as análises, é possível destacar algumas contribuições deste trabalho:

- Realização de consultas XPath em documentos XML traduzidos para Prolog de forma transparente. A partir de uma consulta XPath inserida pelo usuário no protótipo desenvolvido, o processo de tradução é iniciado de forma transparente, culminando na execução da respectiva consulta traduzida para Prolog e a posterior exibição do resultado em formato XML.
- Solução portável e robusta. A solução proposta por este trabalho não necessita de configurações complexas como em um banco de dados XML nativo, podendo ainda ser executada em diversos sistemas operacionais. Além disso, esta solução consegue lidar com documentos XML de tamanhos consideráveis, que podem não ser aceitos por outras abordagens nativas.
- Otimização do processo de tradução de documentos XML para bases de fatos Prolog. O algoritmo de tradução de documentos XML foi refeito, permitindo o processamento de documentos de grande porte, que até então não eram passíveis de serem traduzidos em tempo hábil.
- Construção de arcabouço para possíveis trabalhos futuros. As técnicas de tradução e impressão de resultados foram construídas de forma que

possibilitem sua reutilização em possíveis trabalhos futuros na área, os quais são detalhados em subseção posterior.

8.2 LIMITAÇÕES

Apesar de todo o esforço para alcançar os objetivos propostos neste trabalho, é importante mencionar suas limitações. Ainda que esta abordagem cubra grande parte dos operadores e funções XPath mais comumente utilizados em consultas, ainda restam traduções a serem realizadas. Funções como *text()* e *node()*, que ainda não foram traduzidas, podem limitar o conjunto possível de consultas a serem executadas com o auxílio da abordagem proposta. De forma semelhante, a momentânea ausência da impressão dos *namespaces* para elementos filhos retornados em consultas do tipo *//*[namespace::* = 'XYZ']* pode gerar possíveis dúvidas no usuário.

Outro ponto limitador é a realização de consultas com a utilização de elementos envolvidos em referências cíclicas. Atualmente a abordagem não lida bem com elementos cíclicos que não tenham todo seu caminho definido na consulta. A consulta */site/regions/*/item[@id='item0']/description//keyword* presente no XPathMark, por exemplo, requisita o elemento com referência cíclica *keyword*, que tem como ancestral o elemento *description*. Conforme já mencionado o elemento *keyword* possui como possíveis filhos os elementos *bold* ou *emph*. Os elementos *bold* ou *emph* por sua vez podem possuir o elemento *keyword* como filho, formando ciclos. A não definição do caminho e nível de parentesco (número de ciclos) dificulta a construção de uma consulta Prolog que una os fatos pelos identificadores do fato pai. Uma possível solução poderia passar pela criação de funções Prolog que utilizem o poder recursivo das máquinas de inferência para mapear todos os possíveis caminhos até o elemento desejado.

Por fim a abordagem atual utiliza somente arquivos XML Schema durante o processo de criação da base fatos, tradução da consulta e devolução dos resultados. Apesar de não impeditivo, documentos XML que possuam sua estrutura definida em DTDs dependem da respectiva tradução deste documento para o formato XML Schema adotado. Este tema pode ser estudado posteriormente em trabalhos futuros.

8.3 TRABALHOS FUTUROS

Apesar deste trabalho ter alcançado os objetivos propostos, algumas melhorias e possíveis trabalhos futuros podem ser desenvolvidos. Novas funções e operadores podem ser adicionados à abordagem, permitindo desta forma que o usuário tenha um poder de expressão

ainda maior na construção de consultas XPath. Ainda tratando do poder de expressão do usuário, um importante trabalho futuro seria a adição de suporte a consultas na linguagem XQuery. A linguagem XQuery engloba a XPath e permite realizar consultas mais complexas e transformações dos resultados obtidos.

Sobre questões relacionadas ao desempenho, melhorias podem ser desenvolvidas na ferramenta. Técnicas envolvendo índices podem ser implementadas para elementos que possuem alta cardinalidade. Além disso, durante o processo de tradução dos documentos XML, estatísticas podem ser coletadas. Tais informações estatísticas podem ser utilizadas para particionamento de informações, confecção de novos índices ou ainda auxiliar no momento da construção de uma consulta Prolog.

Outras melhorias estão relacionadas à construção e utilização das regras automáticas e manuais propostas por Lima *et al.* (2012). A nomenclatura das regras automáticas pode ser alterada para que não haja conflitos com os fatos presentes na base de fatos Prolog. Da mesma maneira, a criação de regras manuais deverá respeitar as condições de nomenclatura. Estas ações possibilitarão a utilização das regras automáticas e manuais em conjunto com os fatos criados no processo de tradução de documentos XML. A utilização destas regras pode ser de grande ajuda para o usuário quando se trata de inferência e descoberta de conhecimento implícito em documentos XML. A utilização da linguagem XPath pode auxiliar este tema, de forma amigável.

Como trabalho futuro, é possível também imaginar neste cenário o aproveitamento de novas relações entre elementos, descobertas através de regras criados previamente por especialistas dos documentos XML traduzidos para Prolog. Tais relações poderiam ser mapeadas como pseudo elementos, e posteriormente consultadas via XPath. Como exemplo pode-se utilizar a regra manual *sameYearPublication* descrita anteriormente. A Figura 100 exibe a consulta ao pseudo elemento *sameyearpublication* que retorna os pares de pesquisadores que publicaram algum artigo no mesmo ano.

Consulta XPath:

//sameyearpublication

Resposta:

```
1<sameyearpublication>
2    <year>2016</year>
3    <author>Leo Cesar </author>
4    <author>Cesar Mac </author>
5</sameyearpublication>
```

Figura 100: Consulta ao pseudo elemento *sameyearpublication* utilizando XPath e sua respectiva resposta XML

REFERÊNCIAS

ALMENDROS-JIMÉNEZ, J.; BECERRA-TERÓN, A.; ENCISO-BANOS, F. J. Querying XML documents in Logic Programming. **Journal of Theory and Practice of Logic Programming**, v. 8, n. 3, p. 323–361, 2008.

ALMENDROS-JIMENEZ, J. M. A Rule-based Implementation of XQuery. **IX Jornadas sobre Programacion y Lenguajes ITaller de Programacion Funcional**, 9 ago. 2009.

ALMENDROS-JIMÉNEZ, J. M.; CABALLERO, R.; GARCÍA-RUIZ, Y.; SÁENZ-PÉREZ, F. XPath Query Processing in a Functional-Logic Language. **Electronic Notes in Theoretical Computer Science**, v. 282, p. 19–34, maio 2012.

ALMENDROS-JIMÉNEZ, J. M.; LUNA, A.; MORENO, G. A Flexible XPath-Based Query Language Implemented with Fuzzy Logic Programming. In: BASSILIADES, N.; GOVERNATORI, G.; PASCHKE, A. (Ed.). **Rule-Based Reasoning, Programming, and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. 6826p. 186–193.

ALMENDROS-JIMÉNEZ, J. M.; LUNA TEDESQUI, A.; MORENO, G. Dynamic Filtering of Ranked Answers When Evaluating Fuzzy XPath Queries. In: 9th International Conference, RSCTC, Granada and Madrid, Spain: 2014.

ANDRADE, A.; RUBERG, G.; BAIÃO, F.; BRAGANHOLO, V.; MATTOSO, M. Efficiently Processing XML Queries over Fragmented Repositories with PartiX. In: INTERNATIONAL WORKSHOP ON DATABASE TECHNOLOGIES FOR HANDLING XML INFORMATION ON THE WEB (DATAX). Munich, Germany: 2006.

BOAG, S.; CHAMBERLIN, D.; FERNANDEZ, M. F.; FLORESCU, D.; ROBIE, J.; SIMÉON, J. **XQuery 1.0: An XML Query Language. W3C Recommendation.** Disponível em: <www.w3.org/TR/xquery>. Acesso em: 11 jun. 2016.

BRATKO, I. **Prolog programming for artificial intelligence**. Harlow, England; New York: Addison Wesley, 2001.

CLARK, J.; DEROSE, S. **XML Path Language (XPath) Version 1.0. W3C Recommendation.** Disponível em: <www.w3.org/tr/xpath>. Acesso em: 11 jun. 2016.

COELHO, J.; FLORIDO, M. XCentric: Logic Programming for XML Processing. In: Lisbon, Portugal. In: CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT. Lisbon, Portugal: ACM Press, 2007.

DENTI, E.; OMICINI, A.; RICCI, A. tuProlog: A Light-weight Prolog for Internet Applications and Infrastructures. **Practical Aspects of Declarative Languages**, p. 184–198, 2001.

FALLSIDE, D. C.; WALMSLEY, P. **XML Schema Part 0: Primer Second Edition. W3C Recommendation.** Disponível em: <http://www.w3.org/TR/xmlschema-0/>.

FIGUEIREDO, G.; BRAGANHOLO, V.; MATTOSO, M. Processing Queries over Distributed XML Databases. **Journal of Information and Data Management (JIDM)**, v. 1, n. 3, p. 455–470, 2010.

FOMICHEV, A.; GRINEV, M.; KUZNETSOV, S. Sedna: A Native XML DBMS. In: INTERNATIONAL CONFERENCE ON CURRENT TRENDS IN THEORY AND PRACTICE OF COMPUTER SCIENCE (SOFSEM). Merin, Czech Republic: 2006.

FRANCESCHET, M. XPathMark: An XPath Benchmark for the XMark Generated Data. (S. Bressan, S. Ceri, E. Hunt, Z. G. Ives, Z. Bellahsène, M. Rys, R. Unland, Eds.) In: DATABASE AND XML TECHNOLOGIES, THIRD INTERNATIONAL XML DATABASE SYMPOSIUM. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

JAN WIELEMAKER; TORBJORN LAGER; TOM SCHRIJVERS; MARKUS TRISKA. SWI-Prolog. **Theory and Practice of Logic Programming**, v. 12, n. 1–2, p. 67–96, 2012.

KAY, M. **Saxon XSLT and XQuery Processor**. Disponível em: <http://sourceforge.net/projects/saxon>. Acesso em: 11 jun. 2016.

LEY, M. **DBLP Computer Science bibliography**. Disponível em: <http://dblp.uni-trier.de/>. Acesso em: 11 jun. 2016.

LIMA, D. **Xmlinference: agregando inferência às consultas a dados xml**. 2012. Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil, 2012.

LIMA, D.; DELGADO, C.; MURTA, L.; BRAGANHOLO, V. Towards Querying Implicit Knowledge in XML Documents. **Journal of Information and Data Management (JIDM)**, v. 3, n. 1, p. 51–60, 2012.

MAY, W. XPathLog: A Declarative, Native XML Data Manipulation Language. In: INTERNATIONAL SYMPOSIUM ON DATABASE ENGINEERING AND APPLICATIONS. Grenoble, France: IEEE Computer Society, 2001.

MICHAEL KAY. **XPath and XQuery Functions and Operators 3.0**. Disponível em: <https://www.w3.org/TR/xpath-functions-30/>. Acesso em: 24 jul. 2016.

RODRIGUES, C.; BRAGANHOLO, V.; MATTOSO, M. Virtual Partitioning ad-hoc Queries over Distributed XML Databases. **Journal of Information and Data Management (JIDM)**, v. 2, n. 3, p. 495–510, 2011.

RONEN, R.; SHMUELI, O. Evaluation of Datalog Extended with an XPath Predicate. In: CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMEN (CIKM). ACM Press, 2007.

SANTOS, F.; PINHEIRO, R.; BRAGANHOLO, V. Processamento de Consultas XML usando Máquinas de Inferência. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS (SBBD). São Paulo, Brasil: 2012.

SANTOS, F. G. dos. **Prolog versus xquery processors: a performance evaluation of xml queries processing methods**. 2015. Universidade Federal Fluminense, Niterói, RJ - Brasil, 2015.

SCHATTEN, M.; IVKOVIC, K. Regular Path Expression for Querying Semistructured Data - Implementation in Prolog. In: CENTRAL EUROPEAN CONFERENCE ON INFORMATION AND INTELLIGENT SYSTEMS. Varazdin, Croatia: 19 set. 2012.

SCHMIDT, A.; WAAS, F.; KERSTEN, M.; CAREY, M. J.; MANOLESCU, I.; BUSSE, R. XMark: A Benchmark for XML Data Management. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB). Hong Kong, China: 2002.

SIMÉON, J.; FERNÁNDEZ, M.; CHOI, B.; RADHAKRISHNAN, M.; RESNICK, L.; SUR, G.; WADLER, P. **Galax: An implementation of XQuery**. Disponível em: <http://galax.sourceforge.net/>. Acesso em: 11 jun. 2016.

WIELEMAKER, J. **A C++ Interface to SWI Prolog**. Disponível em: <http://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/pl2cpp.html%27)>. Acesso em: 11 jun. 2016.

WIELEMAKER, J. **SWI-Prolog SGML/XML parser**. Disponível em: <http://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/sgml.html%27)>. Acesso em: 11 jun. 2016.

XML-DEV COMMUNITY. **SAX 2.0.1**. Disponível em: <http://www.saxproject.org/>. Acesso em: 11 jun. 2016.

ANEXO A – XML SCHEMA *BENCHMARK* XPATHMARK PARA O DOCUMENTO D1

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
3
4  <xs:element name="site" type="siteType"/>
5
6  <xs:complexType name="siteType">
7    <xs:sequence>
8     <xs:element name="regions" type="regionsType"/>
9     <xs:element name="categories" type="categoriesType"/>
10     <xs:element name="catgraph" type="catgraphType"/>
11     <xs:element name="people" type="peopleType"/>
12     <xs:element name="open_auctions" type="open_auctionsType"/>
13     <xs:element name="closed_auctions" type="closed_auctionsType"/>
14    </xs:sequence>
15  </xs:complexType>
16
17  <xs:complexType name="regionsType">
18    <xs:sequence>
19     <xs:element name="africa" type="africaType"/>
20     <xs:element name="asia" type="asiaType"/>
21     <xs:element name="australia" type="australiaType"/>
22     <xs:element name="europe" type="europeType"/>
23     <xs:element name="namerica" type="namericaType"/>
24     <xs:element name="samerica" type="samericaType"/>
25    </xs:sequence>
26  </xs:complexType>
27
28  <xs:complexType name="africaType">
29    <xs:sequence>
30     <xs:element name="item" type="itemType"/>
31    </xs:sequence>
32  </xs:complexType>
33
34  <xs:complexType name="asiaType">
35    <xs:sequence>
36     <xs:element name="item" type="itemType"/>
37    </xs:sequence>
38  </xs:complexType>
39
40  <xs:complexType name="australiaType">
41    <xs:sequence>
42     <xs:element name="item" type="itemType"/>
43    </xs:sequence>
44  </xs:complexType>
45
46  <xs:complexType name="europeType">
```

```
47    <sequence>
48      <element maxOccurs="unbounded" name="item" type="itemType"/>
49    </sequence>
50  </complexType>
51
52  <complexType name="namericaType">
53    <sequence>
54      <element maxOccurs="unbounded" name="item" type="itemType"/>
55    </sequence>
56  </complexType>
57
58  <complexType name="samericaType">
59    <sequence>
60      <element name="item" type="itemType"/>
61    </sequence>
62  </complexType>
63
64  <complexType name="categoriesType">
65    <sequence>
66      <element name="category" type="categoryType"/>
67    </sequence>
68  </complexType>
69
70  <complexType name="categoryType">
71    <sequence>
72      <element name="name" type="nameType"/>
73      <element name="description" type="descriptionType"/>
74    </sequence>
75    <attribute name="id" use="required" type="xs:NCName"/>
76  </complexType>
77
78  <complexType name="catgraphType">
79    <sequence>
80      <element name="edge" type="edgeType"/>
81    </sequence>
82  </complexType>
83
84  <complexType name="edgeType">
85    <attribute name="from" use="required" type="xs:NCName"/>
86    <attribute name="to" use="required" type="xs:NCName"/>
87  </complexType>
88
89  <complexType name="peopleType">
90    <sequence>
91      <element maxOccurs="unbounded" name="person" type="personType"/>
92    </sequence>
93  </complexType>
94
95  <complexType name="personType">
96    <sequence>
97      <element name="name" type="nameType" />
```

```
98      <xs:element name="emailaddress" type="emailaddressType" />
99      <xs:element minOccurs="0" name="phone" type="phoneType"/>
100      <xs:element minOccurs="0" name="address" type="addressType"/>
101      <xs:element minOccurs="0" name="homepage" type="homepageType"/>
102      <xs:element minOccurs="0" name="creditcard" type="creditcardType"/>
103      <xs:choice minOccurs="0">
104        <xs:element name="profile" type="profileType"/>
105        <xs:element name="watches" type="watchesType"/>
106      </xs:choice>
107    </xs:sequence>
108    <xs:attribute name="id" use="required" type="xs:NCName"/>
109  </xs:complexType>
110
111  <xs:simpleType name="emailaddressType">
112   <xs:restriction base="xs:anyURI"/>
113  </xs:simpleType>
114
115  <xs:simpleType name="phoneType">
116   <xs:restriction base="xs:string"/>
117  </xs:simpleType>
118
119  <xs:complexType name="addressType">
120    <xs:sequence>
121      <xs:element name="street" type="streetType"/>
122      <xs:element name="city" type="cityType"/>
123      <xs:element name="country" type="countryType"/>
124      <xs:element minOccurs="0" name="province" type="provinceType"/>
125      <xs:element name="zipcode" type="zipcodeType"/>
126    </xs:sequence>
127  </xs:complexType>
128
129
130  <xs:simpleType name="streetType">
131   <xs:restriction base="xs:string"/>
132  </xs:simpleType>
133
134  <xs:simpleType name="cityType">
135   <xs:restriction base="xs:NCName"/>
136  </xs:simpleType>
137
138  <xs:simpleType name="countryType">
139   <xs:restriction base="xs:string"/>
140  </xs:simpleType>
141
142  <xs:simpleType name="provinceType">
143   <xs:restriction base="xs:string"/>
144  </xs:simpleType>
145
146  <xs:simpleType name="zipcodeType">
147   <xs:restriction base="xs:integer"/>
148  </xs:simpleType>
```

```
149
150 <xs:simpleType name="homepageType">
151  <xs:restriction base="xs:anyURI"/>
152 </xs:simpleType>
153
154 <xs:simpleType name="creditcardType">
155  <xs:restriction base="xs:string"/>
156 </xs:simpleType>
157
158
159 <xs:complexType name="profileType">
160   <xs:sequence>
161     <xs:element maxOccurs="unbounded" name="interest" type="interestType"/>
162     <xs:element minOccurs="0" name="education"  type="educationType"/>
163     <xs:element minOccurs="0" name="gender" type="genderType"/>
164     <xs:element name="business" type="businessType"/>
165     <xs:element minOccurs="0" name="age" type="ageType"/>
166   </xs:sequence>
167   <xs:attribute name="income" use="required" type="xs:decimal"/>
168 </xs:complexType>
169
170 <xs:complexType name="interestType">
171   <xs:attribute name="category" use="required" type="xs:NCName"/>
172 </xs:complexType>
173
174 <xs:simpleType name="educationType">
175  <xs:restriction base="xs:string"/>
176 </xs:simpleType>
177
178 <xs:simpleType name="genderType">
179  <xs:restriction base="xs:NCName"/>
180 </xs:simpleType>
181
182 <xs:simpleType name="businessType">
183  <xs:restriction base="xs:NCName"/>
184 </xs:simpleType>
185
186 <xs:simpleType name="ageType">
187  <xs:restriction base="xs:integer"/>
188 </xs:simpleType>
189
190 <xs:complexType name="watchesType">
191   <xs:sequence>
192     <xs:element maxOccurs="unbounded" name="watch" type="watchType"/>
193   </xs:sequence>
194 </xs:complexType>
195
196 <xs:complexType name="watchType">
197   <xs:attribute name="open_auction" use="required" type="xs:NCName"/>
198 </xs:complexType>
199
```

```
200 <xs:complexType name="open_auctionsType">
201   <xs:sequence>
202     <xs:element maxOccurs="unbounded" name="open_auction" type="open_auctionType"/>
203   </xs:sequence>
204 </xs:complexType>
205
206 <xs:complexType name="open_auctionType">
207   <xs:sequence>
208     <xs:element name="initial" type="initialType"/>
209     <xs:element minOccurs="0" name="reserve" type="reserveType"/>
210     <xs:element minOccurs="0" maxOccurs="unbounded" name="bidder" type="bidderType"/>
211     <xs:element name="current" type="currentType"/>
212     <xs:element minOccurs="0" name="privacy" type="privacyType"/>
213     <xs:element name="itemref" type="itemrefType"/>
214     <xs:element name="seller" type="sellerType"/>
215     <xs:element name="annotation" type="annotationType"/>
216     <xs:element name="quantity" type="quantityType"/>
217     <xs:element name="type" type="typeType"/>
218     <xs:element name="interval" type="intervalType"/>
219   </xs:sequence>
220   <xs:attribute name="id" use="required" type="xs:NCName"/>
221 </xs:complexType>
222
223 <xs:simpleType name="initialType">
224  <xs:restriction base="xs:decimal"/>
225 </xs:simpleType>
226
227 <xs:simpleType name="reserveType">
228  <xs:restriction base="xs:decimal"/>
229 </xs:simpleType>
230
231 <xs:complexType name="bidderType">
232   <xs:sequence>
233     <xs:element name="date" type="dateType"/>
234     <xs:element name="time" type="timeType"/>
235     <xs:element name="personref" type="personrefType"/>
236     <xs:element name="increase" type="increaseType"/>
237   </xs:sequence>
238 </xs:complexType>
239
240 <xs:simpleType name="timeType">
241  <xs:restriction base="xs:NMTOKEN"/>
242 </xs:simpleType>
243
244 <xs:complexType name="personrefType">
245   <xs:attribute name="person" use="required" type="xs:NCName"/>
246 </xs:complexType>
247
248 <xs:simpleType name="increaseType">
249  <xs:restriction base="xs:decimal"/>
250 </xs:simpleType>
```

```
251
252 <xs:simpleType name="currentType">
253  <xs:restriction base="xs:decimal"/>
254 </xs:simpleType>
255
256 <xs:simpleType name="privacyType">
257  <xs:restriction base="xs:NCName"/>
258 </xs:simpleType>
259
260 <xs:complexType name="intervalType">
261   <xs:sequence>
262     <xs:element name="start" type="startType"/>
263     <xs:element name="end" type="endType"/>
264   </xs:sequence>
265 </xs:complexType>
266
267 <xs:simpleType name="startType">
268  <xs:restriction base="xs:string"/>
269 </xs:simpleType>
270
271 <xs:simpleType name="endType">
272  <xs:restriction base="xs:string"/>
273 </xs:simpleType>
274
275 <xs:complexType name="closed_auctionsType">
276   <xs:sequence>
277     <xs:element maxOccurs="unbounded" name="closed_auction" type="closed_auctionType"/>
278   </xs:sequence>
279 </xs:complexType>
280
281 <xs:complexType name="closed_auctionType">
282   <xs:sequence>
283     <xs:element name="seller" type="sellerType"/>
284     <xs:element name="buyer" type="buyerType"/>
285  <xs:element name="itemref" type="itemrefType"/>
286     <xs:element name="price" type="priceType"/>
287     <xs:element name="date" type="dateType"/>
288     <xs:element name="quantity" type="quantityType"/>
289     <xs:element name="type" type="typeType"/>
290     <xs:element name="annotation" type="annotationType"/>
291   </xs:sequence>
292 </xs:complexType>
293
294 <xs:complexType name="buyerType">
295   <xs:attribute name="person" use="required" type="xs:NCName"/>
296 </xs:complexType>
297
298 <xs:simpleType name="priceType">
299  <xs:restriction base="xs:decimal"/>
300 </xs:simpleType>
301
```

```
302 <xs:complexType name="itemType">
303   <xs:sequence>
304     <xs:element  name ="location" type="locationType"/>
305     <xs:element name ="quantity" type="quantityType"/>
306     <xs:element name ="name" type="nameType"/>
307     <xs:element name ="payment" type="paymentType"/>
308     <xs:element name ="description" type="descriptionType"/>
309     <xs:element name ="shipping" type="shippingType"/>
310     <xs:element maxOccurs="unbounded" name="incategory" type="incategoryType"/>
311     <xs:element  name="mailbox" type="mailboxType"/>
312   </xs:sequence>
313   <xs:attribute name="featured" type="xs:NCName"/>
314   <xs:attribute name="id" use="required" type="xs:NCName"/>
315 </xs:complexType>
316
317 <xs:simpleType name="locationType">
318  <xs:restriction base="xs:string"/>
319 </xs:simpleType>
320
321 <xs:simpleType name="paymentType">
322  <xs:restriction base="xs:string"/>
323 </xs:simpleType>
324
325 <xs:simpleType name="shippingType">
326  <xs:restriction base="xs:string"/>
327 </xs:simpleType>
328
329 <xs:complexType name="incategoryType">
330   <xs:attribute name="category" use="required" type="xs:NCName"/>
331 </xs:complexType>
332
333 <xs:complexType name="mailboxType">
334   <xs:sequence>
335     <xs:element minOccurs="0" maxOccurs="unbounded" name="mail" type="mailType"/>
336   </xs:sequence>
337 </xs:complexType>
338
339 <xs:complexType name="mailType">
340   <xs:sequence>
341     <xs:element name="from" type="fromType"/>
342     <xs:element name="to" type="toType"/>
343     <xs:element name="date" type="dateType"/>
344     <xs:element name="text" type="textType"/>
345   </xs:sequence>
346 </xs:complexType>
347
348 <xs:simpleType name="fromType">
349  <xs:restriction base="xs:string"/>
350 </xs:simpleType>
351
352 <xs:simpleType name="toType">
```

```
353   <xs:restriction base="xs:string"/>
354 </xs:simpleType>
355
356 <xs:simpleType name="nameType">
357   <xs:restriction base="xs:string"/>
358 </xs:simpleType>
359
360 <xs:complexType name="descriptionType">
361     <xs:choice>
362       <xs:element name="parlist" type="parlistType"/>
363       <xs:element name="text"  type="textType"/>
364     </xs:choice>
365 </xs:complexType>
366
367 <xs:simpleType name="dateType">
368   <xs:restriction base="xs:string"/>
369 </xs:simpleType>
370
371 <xs:complexType name="itemrefType">
372     <xs:attribute name="item" use="required" type="xs:NCName"/>
373 </xs:complexType>
374
375 <xs:complexType name="sellerType">
376     <xs:attribute name="person" use="required" type="xs:NCName"/>
377 </xs:complexType>
378
379 <xs:complexType name="annotationType">
380     <xs:sequence>
381       <xs:element name="author" type="authorType"/>
382       <xs:element name="description" type="descriptionType"/>
383       <xs:element name="happiness" type="happinessType"/>
384     </xs:sequence>
385 </xs:complexType>
386
387 <xs:complexType name="authorType">
388     <xs:attribute name="person" use="required" type="xs:NCName"/>
389 </xs:complexType>
390
391 <xs:simpleType name="happinessType">
392   <xs:restriction base="xs:integer"/>
393 </xs:simpleType>
394
395 <xs:simpleType name="quantityType">
396   <xs:restriction base="xs:integer"/>
397 </xs:simpleType>
398
399 <xs:simpleType name="typeType">
400   <xs:restriction base="xs:string"/>
401 </xs:simpleType>
402
403 <xs:complexType name="textType" mixed="true">
```

```
404     <xs:choice minOccurs="0" maxOccurs="unbounded">
405       <xs:element name="bold" type="boldType"/>
406       <xs:element name="emph" type="emphType"/>
407       <xs:element name="keyword" type="keywordType"/>
408     </xs:choice>
409   </xs:complexType>
410
411   <xs:complexType name="parlistType">
412     <xs:sequence>
413       <xs:element minOccurs="0" maxOccurs="unbounded" name="listitem" type="listitemType"/>
414     </xs:sequence>
415   </xs:complexType>
416
417   <xs:complexType name="listitemType">
418     <xs:sequence>
419       <xs:element minOccurs="0" name="parlist" type="parlistType"/>
420       <xs:element minOccurs="0" name="text" type="textType"/>
421     </xs:sequence>
422   </xs:complexType>
423
424   <xs:complexType name="boldType" mixed="true">
425     <xs:choice minOccurs="0" maxOccurs="unbounded">
426       <xs:element name="emph" type="emphType"/>
427       <xs:element name="keyword" type="keywordType"/>
428     </xs:choice>
429   </xs:complexType>
430
431   <xs:complexType name="keywordType" mixed="true">
432     <xs:choice minOccurs="0" maxOccurs="unbounded">
433       <xs:element name="bold" type="boldType"/>
434       <xs:element name="emph" type="emphType"/>
435     </xs:choice>
436   </xs:complexType>
437
438   <xs:complexType name="emphType" mixed="true">
439     <xs:choice minOccurs="0" maxOccurs="unbounded">
440       <xs:element name="bold" type="boldType" />
441       <xs:element name="keyword" type="keywordType"/>
442     </xs:choice>
443   </xs:complexType>
444
445 </xs:schema>
```

ANEXO B – XML SCHEMA *BENCHMARK* XPATHMARK PARA O DOCUMENTO D2

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="html" type="htmlType"/>
4
5   <xs:complexType name="htmlType" >
6     <xs:sequence>
7       <xs:element name="head" type="headType"/>
8       <xs:element name="body" type="bodyType"/>
9     </xs:sequence>
10    <xs:attribute name="xmlns:xlink" use="required" type="stringtype"/>
11    <xs:attribute name="xml:lang" use="required"/>
12  </xs:complexType>
13
14  <xs:complexType name="headType">
15    <xs:sequence>
16      <xs:element name="title" type="stringtype"/>
17    </xs:sequence>
18  </xs:complexType>
19
20
21  <xs:complexType name="bodyType">
22    <xs:sequence>
23      <xs:element name="h1" type="h1Type"/>
24      <xs:element name="svg:SVG" type="svg:SVGType"/>
25      <xs:choice maxOccurs="unbounded">
26        <xs:element name="hr" type="stringtype"/>
27        <xs:element name="p" type="pType"/>
28      </xs:choice>
29    </xs:sequence>
30  </xs:complexType>
31
32  <xs:complexType name="h1Type" mixed="true">
33    <xs:attribute name="align" use="required" type="xs:NCName"/>
34  </xs:complexType>
35
36  <xs:complexType name="svg:SVGType" >
37    <xs:attribute name="xmlns:svg" use="required" type="stringtype"/>
38    <xs:attribute name="svg:width" use="required" type="inttype"/>
39    <xs:attribute name="svg:height" use="required" type="inttype"/>
40    <xs:element name="svg:ellipse" type="svg:ellipseType"/>
41    <xs:element name="svg:rect" type="svg:rectType"/>
42  </xs:complexType>
43
44  <xs:complexType name="svg:ellipseType" >
45    <xs:attribute name="svg:rx" use="required" type="inttype"/>
46    <xs:attribute name="svg:ry" use="required" type="inttype"/>
47  </xs:complexType>
48
49  <xs:complexType name="svg:rectType" >
50    <xs:attribute name="svg:x" use="required" type="inttype"/>
51    <xs:attribute name="svg:y" use="required" type="inttype"/>
```

```
52    <xs:attribute name="svg:width" use="required" type="stringtype"/>
53    <xs:attribute name="svg:height" use="required" type="stringtype"/>
54   </xs:complexType>
55
56   <xs:complexType name="pType" mixed="true">
57    <xs:attribute name="xlink:href" type="stringtype"/>
58    <xs:attribute name="xlink:type" type="stringtype"/>
59    <xs:attribute name="xml:lang" type="stringtype"/>
60   </xs:complexType>
61
62   <xs:simpleType name="stringtype">
63    <xs:restriction base="xs:string"/>
64   </xs:simpleType>
65
66   <xs:simpleType name="inttype">
67    <xs:restriction base="xs:positiveInteger"/>
68   </xs:simpleType>
69
70   <xs:simpleType name="dectype">
71    <xs:restriction base="xs:decimal"/>
72   </xs:simpleType>
73
74</xs:schema>
```

ANEXO C – CONSULTAS DO *BENCHMARK* XPATHMARK PARA O DOCUMENTO XML D1

Consultas XPathMark -Documento 1	
Q1	/site/regions/*/item
Q2	/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/text/keyword
Q3	//keyword
Q4	/descendant-or-self::listitem/descendant-or-self::keyword
Q5	/site/regions/*/item[parent::namerica or parent::samerica]
Q6	//keyword/ancestor::listitem
Q7	//keyword/ancestor-or-self::mail
Q8	/site/open_auctions/open_auction[bidder[personref/@person = 'person0']/ following-sibling::bidder[personref/@person = 'person1']]
Q9	/site/open_auctions/open_auction[@id = 'open_auction0']/bidder/preceding-sibling::bidder
Q10	/site/regions/*/item[@id = 'item0']/following::item
Q11	/site/open_auctions/open_auction/bidder[personref/@person = 'person1']/ preceding::bidder[personref/@person = 'person0']
Q12	//item[@featured = 'yes']
Q13	//*[@id]
Q14	//person[namespace::xml]
Q15	//increase[. > 20]
Q16	//xml:*
Q17	/node()
Q18	/comment()
Q19	/processing-instruction()
Q20	/processing-instruction('robots')
Q21	/site/regions/*/item[@id='item0']/description//keyword/text()
Q22	/site/regions/namerica/item | /site/regions/samerica/item
Q23	/site/people/person[address and (phone or homepage)]
Q24	/site/people/person[not(homepage)]
Q25	id('person0')/name
Q26	id(/site/people/person[@id='person1']/watches/watch/@open_auction)
Q27	id(id(/site/people/person[@id='person1']/watches/watch/@open_auction)/seller/@person)
Q28	id(/site/closed_auctions/closed_auction[buyer/@person='person4']/itemref/@item) [parent::namerica or parent::samerica]
Q29	id(/site/closed_auctions/closed_auction[id(seller/@person)/name='Alassane Hogan']/itemref/@item)
Q30	/site/open_auctions/open_auction/bidder[position() = 1 and position() = last()]
Q31	/site/open_auctions/open_auction[count(bidder) > 5]
Q32	//*[local-name() = 'item']
Q33	//*[name() = 'svg:item']
Q34	//*[boolean(namespace-uri())]
Q35	//*[lang('it')]
Q36	/site/regions/*/item[contains(description,'gold')]
Q37	/site/people/person[starts-with(name,'Ed')]

Q38	/site/regions/*/item/mailbox/mail[substring-before(date,'/') = '10']
Q39	/site/regions/*/item/mailbox/mail[substring-before(substring-after(date,'/'),'/') = '09']
Q40	/site/regions/*/item/mailbox/mail[substring-after(substring-after(date,'/'), '/') = '1998']
Q41	/site/regions/*/item/mailbox/mail[substring(date,7,2) = '20']
Q42	/site/regions/*/item[string-length(normalize-space(string(description))) > 1000]
Q43	/site/people/person [string-length(translate(concat(address/street,address/city,address/country,address/zipcode)," ","")) > 30]
Q44	/site/open_auctions/open_auction[floor(sum(bidder/increase)) >= 70]
Q45	/site/open_auctions/open_auction[ceiling(sum(bidder/increase)) <= 70]
Q46	/site/open_auctions/open_auction[round((number(current) - number(initial)) div count(bidder)) > 8]
Q47	/site/people/person[boolean(emailaddress) = true() and not(boolean(homepage)) = false()]

ANEXO C – CONSULTAS DO *BENCHMARK* XPATHMARK PARA O DOCUMENTO XML D2

Consultas XPathMark -Documento 2	
Q1	/comment()
Q2	/processing-instruction('robots')
Q3	//*[namespace::xlink]
Q4	//*[namespace::* = 'http://www.w3.org/1999/xlink']
Q5	//xlink:*
Q6	//*[namespace::svg]
Q7	//*[namespace::* = 'http://www.w3.org/2000/svg']
Q8	//svg:*
Q9	//*[boolean(namespace-uri())]
Q10	//*[local-name() = 'ellipse']
Q11	//*[name() = 'svg:ellipse']
Q12	//*[lang('it')]

ANEXO D – CONSULTAS DO *BENCHMARK* XPATHMARK E SUA RESPECTIVA TRADUÇÃO PROLOG PARA O DOCUMENTO D1

Consulta Q1:

Consulta XPath:
/site/regions/*/item
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), regions(IDSITE, IDREGIONS), findall(IDWILD,(IDWILD = IDAFRICA,africa(IDREGIONS, IDAFRICA)); (IDWILD = IDASIA, asia(IDREGIONS, IDASIA)); (IDWILD = IDAUSTRALIA,australia(IDREGIONS, IDAUSTRALIA)); (IDWILD= IDEUROPE, europe(IDREGIONS, IDEUROPE)); (IDWILD = IDNAMERICA, namerica(IDREGIONS, IDNAMERICA)); (IDWILD = IDSAMERICA,samerica(IDREGIONS, IDSAMERICA)) ,LISTWILD) , member(IDWILD,LISTWILD), item(IDWILD, IDITEM), nonvar(IDITEM), IDNODEORDER = IDITEM, item(_, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDITEM = IDNODEORDERSORTED, (print_item(IDNOGRANDPARENT,IDITEM)).

Consulta Q2:

Consulta XPath:
/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/text/keyword
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), closed_auctions(IDSITE, IDCLOSED_AUCTIONS), closed_auction(IDCLOSED_AUCTIONS, IDCLOSED_AUCTION), annotation(IDCLOSED_AUCTION, IDANNOTATION), description(IDANNOTATION, IDDESCRIPTION), parlist(IDDESCRIPTION, IDPARLIST), listitem(IDPARLIST, IDLISTITEM), (text(IDLISTITEM, IDTEXT, TEXT); text(IDLISTITEM, IDTEXT)), (keyword(IDTEXT, IDKEYWORD, KEYWORD); keyword(IDTEXT, IDKEYWORD)), nonvar(IDKEYWORD), IDNODEORDER = IDKEYWORD, (keyword(IDTEXT, IDNODEORDER, _); keyword(IDTEXT, IDNODEORDER))), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDKEYWORD = IDNODEORDERSORTED , (print_keyword(IDTEXT,IDKEYWORD)).

Consulta Q3:

Consulta XPath:
//keyword
Consulta Prolog:
findall(IDNODEORDER, ((keyword(IDNOPARENT,IDKEYWORD, KEYWORD); keyword(IDNOPARENT,IDKEYWORD)), nonvar(IDKEYWORD), IDNODEORDER = IDKEYWORD, (keyword(IDNOPARENT, IDNODEORDER, _); keyword(IDNOPARENT, IDNODEORDER))), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDKEYWORD = IDNODEORDERSORTED , (print_keyword(IDNOPARENT,IDKEYWORD)).

Consulta Q5:

Consulta XPath:
/site/regions/*/item[parent::namerica or parent::samerica]
Consulta Prolog:
findall(IDNODEORDER, (IDAFRICA is 0, IDASIA is 0, IDAUSTRALIA is 0, IDEUROPE is 0, site(IDSITE), regions(IDSITE, IDREGIONS), findall(IDWILD, (IDWILD = IDAFRICA,a frica(IDREGIONS, IDAFRICA)); (IDWILD = IDASIA, asia(IDREGIONS, IDASIA)); (IDWILD = IDAUSTRALIA, australia(IDREGIONS, IDAUSTRALIA)); (IDWILD = IDEUROPE, europe(IDREGIONS, IDEUROPE)); (IDWILD = IDNAMERICA, namerica(IDREGIONS, IDNAMERICA)); (IDWILD = IDSAMERICA, samerica(IDREGIONS, IDSAMERICA)), LISTWILD), member(IDWILD,LISTWILD), item(IDWILD, IDITEM), nonvar(IDITEM), IDNODEORDER = IDITEM, item(_, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDITEM = IDNODEORDERSORTED, (print_item(IDNOGRANDPARENT,IDITEM)).

Consulta Q8:

Consulta XPath:
/site/open_auctions/open_auction[bidder[personref/@person = 'person0']/following-sibling::bidder[personref/@person = 'person1']]
Consulta Prolog:
findall(IDNODEORDER, (findall(IDBIDDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS), open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION), bidder(IDOPEN_AUCTION, IDBIDDER), personref(IDBIDDER, IDPERSONREF), PERSON = 'person0', personref_attribute_person(IDPERSONREF, IDPERSON, PERSON)), FOLLOWINGLIST), following(FOLLOWINGLIST, FOLLOWINGFINALLIST), member(IDBIDDER1, FOLLOWINGFINALLIST), bidder(IDOPEN_AUCTION, IDBIDDER1), bidder(IDOPEN_AUCTION, IDBIDDER2), IDBIDDER1 < IDBIDDER2, personref(IDBIDDER2, IDPERSONREF2), PERSON2 = 'person1', personref_attribute_person(IDPERSONREF2, IDPERSON2, PERSON2), nonvar(IDOPEN_AUCTION), IDNODEORDER = IDOPEN_AUCTION, open_auction(IDOPEN_AUCTIONS, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDOPEN_AUCTION = IDNODEORDERSORTED, (print_open_auction(IDOPEN_AUCTIONS,IDOPEN_AUCTION)).

Consulta Q9:

Consulta XPath:
/site/open_auctions/open_auction[@id = 'open_auction0']/bidder/preceding-sibling::bidder
Consulta Prolog:
findall(IDNODEORDER, (findall(IDBIDDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS), open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION), ID = 'open_auction0', open_auction_attribute_id(IDOPEN_AUCTION, IDID ,ID), bidder(IDOPEN_AUCTION, IDBIDDER)), PRECEDINGLIST), preceding(PRECEDINGLIST, PRECEDINGFINALLIST), member(IDBIDDER1, PRECEDINGFINALLIST), bidder(IDOPEN_AUCTION, IDBIDDER1), bidder(IDOPEN_AUCTION, IDBIDDER2), IDBIDDER1 > IDBIDDER2, nonvar(IDBIDDER2), IDNODEORDER = IDBIDDER2, bidder(IDOPEN_AUCTION, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDBIDDER = IDNODEORDERSORTED, (print_bidder(IDOPEN_AUCTION,IDBIDDER)).

Consulta Q10:

Consulta XPath:

/site/regions/*/item[@id = 'item0']/following::item

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), regions(IDSITE, IDREGIONS),
findall(IDWILD, (IDWILD = IDAFRICA, africa(IDREGIONS, IDAFRICA));
(IDWILD = IDASIA, asia(IDREGIONS, IDASIA)); (IDWILD = IDAUSTRALIA, australia(IDREGIONS, IDAUSTRALIA));
(IDWILD = IDEUROPE, europe(IDREGIONS, IDEUROPE));
(IDWILD = IDNAMERICA, namerica(IDREGIONS, IDNAMERICA));
(IDWILD = IDSAMERICA, samerica(IDREGIONS, IDSAMERICA)), LISTWILD), member(IDWILD,LISTWILD),
item(IDWILD, IDITEM), ID = 'item0', item_attribute_id(IDITEM, IDID, ID),
obtainDescendents_item(IDITEM, LISTPF), item(IDWILD1, IDITEM1),
IDITEM < IDITEM1, notMember(IDITEM1, LISTPF), nonvar(IDITEM1), IDNODEORDER = IDITEM1,
item(_, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDITEM = IDNODEORDERSORTED,
(print_item(IDNOGRANDPARENT,IDITEM)).
```

Consulta Q11:

Consulta XPath:

/site/open_auctions/open_auction/bidder[personref/@person = 'person1']/preceding::bidder[personref/@person = 'person0']

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS),
open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION), bidder(IDOPEN_AUCTION, IDBIDDER),
personref(IDBIDDER, IDPERSONREF), PERSON = 'person1',
personref_attribute_person(IDPERSONREF, IDPERSON, PERSON),
obtainAncestrals_bidder(IDBIDDER, LISTPF), bidder(IDOPEN_AUCTION1, IDBIDDER1),
IDBIDDER > IDBIDDER1, notMember(IDBIDDER1, LISTPF), personref(IDBIDDER1, IDPERSONREF1),
PERSON1 = 'person0', personref_attribute_person(IDPERSONREF1, IDPERSON1, PERSON1),
nonvar(IDBIDDER1), IDNODEORDER = IDBIDDER1, bidder(_, IDNODEORDER)), LISTNODEORDER), setof(X,
member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED,
LISTNODEORDERSORTED), IDBIDDER = IDNODEORDERSORTED,
(print_bidder(IDOPEN_AUCTION,IDBIDDER)).
```

Consulta Q12:

Consulta XPath:
//item[@featured = 'yes']
Consulta Prolog:
findall(IDNODEORDER, (item(IDNOPARENT, IDITEM), FEATURED = 'yes', item_attribute_featured(IDITEM, IDFEATURED, FEATURED), nonvar(IDITEM), IDNODEORDER = IDITEM, item(IDNOPARENT, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDITEM = IDNODEORDERSORTED, (print_item(IDNOPARENT,IDITEM)).

Consulta Q13:

Consulta XPath:
//*[@id]
Consulta Prolog:
findall(IDNODEORDER, (IDNODEORDER = IDITEM, item(_, IDNODEORDER);IDNODEORDER = IDPERSON, person(_, IDNODEORDER);IDNODEORDER = IDOPEN_AUCTION, open_auction(_, IDNODEORDER); IDNODEORDER = IDCATEGORY, category(_, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), ((IDITEM = IDNODEORDERSORTED, print_item(IDNOPARENT, IDNODEORDERSORTED)) ; (IDPERSON = IDNODEORDERSORTED, print_person(IDNOPARENT,IDNODEORDERSORTED)); (IDOPEN_AUCTION = IDNODEORDERSORTED, print_open_auction(IDNOPARENT,IDNODEORDERSORTED)); (IDCATEGORY = IDNODEORDERSORTED, print_category(IDNOPARENT, IDNODEORDERSORTED))).

Consulta Q14:

Consulta XPath:
//person[namespace::xml]
Consulta Prolog:
findall(DNODEORDER, (person(IDNOPARENT, IDPERSON), nonvar(IDPERSON), IDNODEORDER = IDPERSON, person(IDNOPARENT, IDNODEORDER)), LISTNODEORDER), setof(X, member(X,LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDPERSON = IDNODEORDERSORTED , (print_person(IDNOPARENT,IDPERSON)).

Consulta Q15:

Consulta XPath:
//increase[. > 20]
Consulta Prolog:
findall(IDNODEORDER, (increase(IDNOPARENT, IDINCREASE, INCREASE), mynumber(INCREASE, INCREASE1), INCREASE1 @> 20, nonvar(IDINCREASE), IDNODEORDER = IDINCREASE, increase(IDNOPARENT, IDNODEORDER, _)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDINCREASE = IDNODEORDERSORTED, (print_increase(IDNOPARENT,IDINCREASE)).

Consulta Q16:

Consulta XPath:
//xml:*
Consulta Prolog:
false.

Consulta Q22:

Consulta XPath:
/site/regions/namerica/item | /site/regions/samerica/item
Consulta Prolog:
Capítulo 9 findall(IDNODEORDER, (setof(_, ((site(IDSITE), regions(IDSITE, IDREGIONS), namerica(IDREGIONS, IDNAMERICA), item(IDNAMERICA, IDITEM)); (site(IDSITE), regions(IDSITE, IDREGIONS), samerica(IDREGIONS, IDSAMERICA), item(IDSAMERICA, IDITEM))), _), nonvar(IDITEM), IDNODEORDER = IDITEM, item(IDSAMERICA, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDITEM = IDNODEORDERSORTED, (print_item(IDSAMERICA,IDITEM)).

Consulta Q23:

Consulta XPath:
/site/people/person[address and (phone or homepage)]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), people(IDSITE, IDPEOPLE), person(IDPEOPLE, IDPERSON), address(IDPERSON, IDADDRESS), setof(_, (phone(IDPERSON, IDPHONE, PHONE); homepage(IDPERSON, IDHOMEPAGE, HOMEPAGE)), _), nonvar(IDPERSON), IDNODEORDER = IDPERSON, person(IDPEOPLE, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDPERSON = IDNODEORDERSORTED, (print_person(IDPEOPLE,IDPERSON)).

Consulta Q24:

Consulta XPath:
/site/people/person[not(homepage)]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), people(IDSITE, IDPEOPLE), person(IDPEOPLE, IDPERSON), findall(IDPERSON, person(IDPEOPLE, IDPERSON), LISTFALSE2), findall(IDPERSON3, (person(IDPEOPLE, IDPERSON3), homepage(IDPERSON3, _, HOMEPAGE)), LISTFALSE3), minus(LISTFALSE2, LISTFALSE3, LISTFALSE4), member(IDPERSON, LISTFALSE4), nonvar(IDPERSON), IDNODEORDER = IDPERSON, person(IDPEOPLE,IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDPERSON = IDNODEORDERSORTED, (print_person(IDPEOPLE,IDPERSON)).

Consulta Q30:

Consulta XPath:
/site/open_auctions/open_auction/bidder[position() = 1 and position() = last()]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS), open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION), bidder(IDOPEN_AUCTION, IDBIDDER), findall(IDSEARCH1, bidder(IDOPEN_AUCTION, IDSEARCH1), LIST1), indexOf(LIST1, IDBIDDER, RESULT1), RESULT1 = 1 , findall(IDSEARCH3, bidder(IDOPEN_AUCTION,IDSEARCH3), LIST3), indexOf(LIST3, IDBIDDER, RESULT3), findall(IDSEARCH4, bidder(IDOPEN_AUCTION, IDSEARCH4), LIST4), listSize(LIST4, LENGTH4) ,indexOf(LIST4, IDBIDDER, LENGTH4), RESULT4 is LENGTH4, RESULT3 = RESULT4, nonvar(IDBIDDER), IDNODEORDER = IDBIDDER, bidder(IDOPEN_AUCTION, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDBIDDER = IDNODEORDERSORTED, (print_bidder(IDOPEN_AUCTION,IDBIDDER)).

Consulta Q31:

Consulta XPath:
/site/open_auctions/open_auction[count(bidder) > 5]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS), open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION), findall(IDBIDDER, (bidder(IDOPEN_AUCTION, IDBIDDER)), LIST), count(LIST, RESULT1), RESULT1 > 5, nonvar(IDOPEN_AUCTION), IDNODEORDER = IDOPEN_AUCTION, open_auction(IDOPEN_AUCTIONS, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDOPEN_AUCTION = IDNODEORDERSORTED, (print_open_auction(IDOPEN_AUCTIONS,IDOPEN_AUCTION)).

Consulta Q32:

Consulta XPath:
//*[local-name() = 'item']
Consulta Prolog:
(print_item(IDNOPARENT,IDNODEORDERSORTED)).

Consulta Q33:

Consulta XPath: //*[name() = 'svg:item']
Consulta Prolog: false.

Consulta Q35:

Consulta XPath: //*[lang('it')]
Consulta Prolog: false.

Consulta Q37:

Consulta XPath: /site/people/person[starts-with(name,'Ed')]
Consulta Prolog: findall(IDNODEORDER, (site(IDSITE), people(IDSITE, IDPEOPLE), person(IDPEOPLE, IDPERSON), name(IDPERSON, IDNAME, NAME), startsWith(NAME, 'Ed'), nonvar(IDPERSON), IDNODEORDER = IDPERSON, person(IDPEOPLE, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDPERSON = IDNODEORDERSORTED, (print_person(IDPEOPLE,IDPERSON)).

Consulta Q38:

Consulta XPath:

```
/site/regions/*/item/mailbox/mail[substring-before(date,'/') = '10']
```

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), regions(IDSITE, IDREGIONS),
findall(IDWILD, (IDWILD = IDAFRICA, africa(IDREGIONS, IDAFRICA));
(IDWILD = IDASIA, asia(IDREGIONS, IDASIA));
(IDWILD = IDAUSTRALIA, australia(IDREGIONS, IDAUSTRALIA));
(IDWILD = IDEUROPE, europe(IDREGIONS, IDEUROPE));
 (IDWILD = IDNAMERICA, namerica(IDREGIONS, IDNAMERICA));
(IDWILD = IDSAMERICA, samerica(IDREGIONS, IDSAMERICA)),LISTWILD), member(IDWILD, LISTWILD),
item(IDWILD, IDITEM), mailbox(IDITEM, IDMAILBOX), mail(IDMAILBOX, IDMAIL), date(IDMAIL, IDDATE,DATE),
substringBefore(DATE, '/', RESULT1) , RESULT1 = '10', nonvar(IDMAIL), IDNODEORDER = IDMAIL,
mail(_, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDMAIL = IDNODEORDERSORTED,
(print_mail(IDNOGRANDPARENT,IDMAIL)).
```

Consulta Q39:

Consulta XPath:

```
/site/regions/*/item/mailbox/mail[substring-before(substring-after(date,'/'),'/') = '09']
```

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), regions(IDSITE, IDREGIONS),
findall(IDWILD,(IDWILD = IDAFRICA, africa(IDREGIONS, IDAFRICA));
 (IDWILD = IDASIA, asia(IDREGIONS, IDASIA));
 (IDWILD = IDAUSTRALIA, australia(IDREGIONS, IDAUSTRALIA));
(IDWILD = IDEUROPE, europe(IDREGIONS, IDEUROPE));
(IDWILD = IDNAMERICA, namerica(IDREGIONS, IDNAMERICA));
(IDWILD = IDSAMERICA, samerica(IDREGIONS, IDSAMERICA)) ,LISTWILD),
 member(IDWILD, LISTWILD), item(IDWILD, IDITEM), mailbox(IDITEM, IDMAILBOX), mail(IDMAILBOX, IDMAIL),
date(IDMAIL, IDDATE, DATE), substringAfter(DATE, '/', RESULT1), substringBefore(RESULT1, '/', RESULT2),
RESULT2 = '09', nonvar(IDMAIL), IDNODEORDER = IDMAIL, mail(_, IDNODEORDER)), LISTNODEORDER),
setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDMAIL = IDNODEORDERSORTED,
(print_mail(IDNOGRANDPARENT,IDMAIL)).
```

Consulta Q40:

Consulta XPath:

```
/site/regions/*/item/mailbox/mail[substring-after(substring-after(date,'/'), '/') = '1998']
```

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), regions(IDSITE, IDREGIONS),
findall(IDWILD,(IDWILD = IDAFRICA, africa(IDREGIONS, IDAFRICA));
(IDWILD = IDASIA, asia(IDREGIONS, IDASIA)); (IDWILD = IDAUSTRALIA, australia(IDREGIONS, IDAUSTRALIA));
(IDWILD = IDEUROPE, europe(IDREGIONS, IDEUROPE));
(IDWILD = IDNAMERICA, namerica(IDREGIONS, IDNAMERICA));
(IDWILD = IDSAMERICA, samerica(IDREGIONS, IDSAMERICA)),LISTWILD),
member(IDWILD, LISTWILD), item(IDWILD, IDITEM), mailbox(IDITEM, IDMAILBOX), mail(IDMAILBOX, IDMAIL),
date(IDMAIL, IDDATE, DATE), substringAfter(DATE, '/', RESULT1), substringAfter(RESULT1, '/', RESULT2),
RESULT2 = '1998', nonvar(IDMAIL), IDNODEORDER = IDMAIL, mail(_, IDNODEORDER)), LISTNODEORDER),
setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
 member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDMAIL = IDNODEORDERSORTED,
(print_mail(IDNOGRANDPARENT,IDMAIL)).
```

Consulta Q41:

Consulta XPath:

```
/site/regions/*/item/mailbox/mail[substring(date,7,2) = '20']
```

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), regions(IDSITE, IDREGIONS),
findall(IDWILD,(IDWILD = IDAFRICA, africa(IDREGIONS, IDAFRICA));
 (IDWILD = IDASIA, asia(IDREGIONS, IDASIA));
(IDWILD = IDAUSTRALIA, australia(IDREGIONS, IDAUSTRALIA));
(IDWILD = IDEUROPE, europe(IDREGIONS, IDEUROPE));
(IDWILD= IDNAMERICA, namerica(IDREGIONS, IDNAMERICA));
(IDWILD = IDSAMERICA, samerica(IDREGIONS, IDSAMERICA)), LISTWILD), member(IDWILD, LISTWILD),
item(IDWILD, IDITEM), mailbox(IDITEM, IDMAILBOX), mail(IDMAILBOX, IDMAIL), date(IDMAIL, IDDATE, DATE),
substring(DATE, 7, 2, RESULT1), RESULT1 = '20', nonvar(IDMAIL), IDNODEORDER = IDMAIL,
mail(_, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDMAIL = IDNODEORDERSORTED,
(print_mail(IDNOGRANDPARENT,IDMAIL)).
```

Consulta Q43:

Consulta XPath:

```
/site/people/person[string-
length(translate(concat(address/street,address/city,address/country,address/zipcode)," ",""")) > 30]
```

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), people(IDSITE, IDPEOPLE), person(IDPEOPLE, IDPERSON),
address(IDPERSON, IDADDRESS), street(IDADDRESS, IDSTREET, STREET), address(IDPERSON, IDADDRESS),
city(IDADDRESS, IDCITY, CITY), string_concat(STREET, CITY, RESULT2) , address(IDPERSON, IDADDRESS),
country(IDADDRESS, IDCOUNTRY, COUNTRY), string_concat(RESULT2, COUNTRY, RESULT3),
address(IDPERSON, IDADDRESS), zipcode(IDADDRESS, IDZIPCODE, ZIPCODE),
string_concat(RESULT3, ZIPCODE, RESULT4) , translate(RESULT4, ' ', '', RESULT5),
stringLength(RESULT5, RESULT6), RESULT6 > 30, nonvar(IDPERSON), IDNODEORDER = IDPERSON,
person(IDPEOPLE, IDNODEORDER)), LISTNODEORDER),
setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDPERSON = IDNODEORDERSORTED,
(print_person(IDPEOPLE,IDPERSON)).
```

Consulta Q44:

Consulta XPath:

```
/site/open_auctions/open_auction[floor(sum(bidder/increase)) >= 70]
```

Consulta Prolog:

```
findall(IDNODEORDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS),
open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION),
findall(INCREASE, (bidder(IDOPEN_AUCTION, IDBIDDER), increase(IDBIDDER, IDINCREASE, INCREASE)), LIST),
sum(LIST, RESULT1), myfloor(RESULT1, RESULT2), RESULT2 >= 70, nonvar(IDOPEN_AUCTION),
IDNODEORDER = IDOPEN_AUCTION, open_auction(IDOPEN_AUCTIONS, IDNODEORDER)), LISTNODEORDER),
setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED),
member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDOPEN_AUCTION = IDNODEORDERSORTED,
(print_open_auction(IDOPEN_AUCTIONS,IDOPEN_AUCTION)).
```

Consulta Q45:

Consulta XPath:
/site/open_auctions/open_auction[ceiling(sum(bidder/increase)) <= 70]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS), open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION), findall(INCREASE, (bidder(IDOPEN_AUCTION, IDBIDDER), increase(IDBIDDER, IDINCREASE, INCREASE)), LIST), sum(LIST, RESULT1), myceiling(RESULT1, RESULT2), RESULT2 =< 70, nonvar(IDOPEN_AUCTION), IDNODEORDER = IDOPEN_AUCTION, open_auction(IDOPEN_AUCTIONS, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDOPEN_AUCTION = IDNODEORDERSORTED, (print_open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION)).

Consulta Q46:

Consulta XPath:
/site/open_auctions/open_auction[round((number(current) - number(initial)) div count(bidder)) > 8]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), open_auctions(IDSITE, IDOPEN_AUCTIONS), open_auction(IDOPEN_AUCTIONS, IDOPEN_AUCTION), findall(IDBIDDER, (bidder(IDOPEN_AUCTION, IDBIDDER)), LIST), count(LIST, RESULT3), initial(IDOPEN_AUCTION, IDINITIAL, INITIAL), mynumber(INITIAL, RESULT2), current(IDOPEN_AUCTION, IDCURRENT, CURRENT), mynumber(CURRENT, RESULT1), RESULT4 is RESULT1 - RESULT2, div(RESULT4, RESULT3, RESULT5), myround(RESULT5, RESULT6), RESULT6 > 8, nonvar(IDOPEN_AUCTION), IDNODEORDER = IDOPEN_AUCTION, open_auction(IDOPEN_AUCTIONS, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDOPEN_AUCTION = IDNODEORDERSORTED, (print_open_auction(IDOPEN_AUCTIONS,IDOPEN_AUCTION)).

Consulta Q47:

Consulta XPath:
/site/people/person[boolean(emailaddress) = true() and not(boolean(homepage)) = false()]
Consulta Prolog:
findall(IDNODEORDER, (site(IDSITE), people(IDSITE, IDPEOPLE), person(IDPEOPLE, IDPERSON), emailaddress(IDPERSON, IDEMAILADDRESS, EMAILADDRESS), verifyBooleanContent(EMAILADDRESS), homepage(IDPERSON, IDHOMEPAGE, HOMEPAGE), verifyBooleanContent(HOMEPAGE), nonvar(IDPERSON), IDNODEORDER = IDPERSON, person(IDPEOPLE, IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED, LISTNODEORDERSORTED), IDPERSON = IDNODEORDERSORTED, (print_person(IDPEOPLE,IDPERSON)).

ANEXO D – CONSULTAS DO *BENCHMARK* XPATHMARK E SUA RESPECTIVA TRADUÇÃO PROLOG PARA O DOCUMENTO D2

Consulta Q3:

Consulta XPath: //*[namespace::xlink]
Consulta Prolog: (print_wild_card_html(IDNODEORDERSORTED)).

Consulta Q4:

Consulta XPath: //*[namespace::* = 'http://www.w3.org/1999/xlink']
Consulta Prolog: (print_wild_card_html(IDNODEORDERSORTED)).

Consulta Q5:

Consulta XPath: //xlink:*
Consulta Prolog: false.

Consulta Q6:

Consulta XPath: //*[namespace::svg]
Consulta Prolog: (print_wild_card_-svg-svg(NOGRANDPARENT,IDNODEORDERSORTED)).

Consulta Q7:

Consulta XPath: //*[namespace::* = 'http://www.w3.org/2000/svg']
Consulta Prolog: (`print_wild_card_-svg-svg`(NOGRANDPARENT,IDNODEORDERSORTED)).

Consulta Q8:

Consulta XPath: //svg:*
Consulta Prolog: (`print_wild_card_-svg-svg`(NOPARENT,IDNODEORDERSORTED)).

Consulta Q10:

Consulta XPath: //*[local-name() = 'ellipse']
Consulta Prolog: (`print_-svg-ellipse`(IDNOPARENT,IDNODEORDERSORTED)).

Consulta Q11:

Consulta XPath: //*[name() = 'svg:ellipse']
Consulta Prolog: (`print_-svg-ellipse`(IDNOPARENT,IDNODEORDERSORTED)).

Consulta Q12:

Consulta XPath:
//*[lang('it')]
Consulta Prolog:
findall(IDNODEORDER, ((IDNODEORDER = IDP, p(_, IDNODEORDER, _); IDNODEORDER = IDP, p(_, IDNODEORDER));html(IDNODEORDER)), LISTNODEORDER), setof(X, member(X, LISTNODEORDER), LISTNODEORDERSORTED), member(IDNODEORDERSORTED,LISTNODEORDERSORTED), ((IDP = IDNODEORDERSORTED, p_attribute_-xml-lang(IDP, IDP_ATTRIBUTE_XML_LANG, 'it'), print_p(IDNOPARENT, IDNODEORDERSORTED)); (IDHTML = IDNODEORDERSORTED, html_attribute_-xml-lang(IDHTML, IDHTML_ATTRIBUTE_XML_LANG, 'it'), print_html(IDNODEORDERSORTED))).