Normalização de ontologias em lógica de descrições para o raciocínio com o provador de teoremas leanCoP

Adriano S. T. Melo[1], **Fred Freitas**[1]

[1]Centro de Informática - Universidade Federal de Pernambuco
Caixa Postal 50740-560 – Cidade Universitária – Recife – PE – Brasil

{astm,fred}@cin.ufpe.br

Abstract. *The Semantic Web promises to revolutionize the way people interact on the Web. Concepts like intelligent agents, semantic-based services, business-to-business communication between companies through rule-based agents should be plentiful in this new scenario. Several technologies are being studied and constructed to form a new infrastructure for the Web. These are the elements that form the layer of logic and proof, context that this project is inserted. This project proposes to implement part of the algorithm of the connection method for description logic, used to do activities of reasoning in a knowledge base. The module to be implemented is the conversion of the axioms of the knowledge base for the positive normal form.*

Resumo. *A Web Semântica promete revolucionar o jeito de como as pessoas interagem na Web. Os conceitos de agentes inteligentes, serviços baseados em semântica de documentos e empresas se comunicando com empresas através de agentes baseados em regras deverão ser abundantes nesse novo cenário. Várias tecnologias estão sendo estudadas e construídas para formar uma nova infra-estrutura para a Web. Entre elas estão os elementos que formarão a camada de lógica e prova, contexto que está inserido este trabalho. Este trabalho propôe implementar parte do algoritmo do método das conexões para lógica de descrição, usado para fazer atividades de raciocínio em uma base de conhecimento. O módulo a ser implementado é a conversão dos axiomas da base de conhecimento para a forma normal positiva.*

1. Introdução

A World Wide Web é uma das tecnologias mais revolucionárias que o homem já inventou. Ela mudou em escala global a forma com que pessoas e empresas trocam informações, contribuindo para que o conhecimento se tornasse mais universal e que limites físicos e lingüísticos fossem cada vez mais minimizados.

A web como conhecemos hoje nasceu de uma proposta feita por Tim Berners-Lee à empresa CERN em 1989 [Berners-Lee 1989]. O problema enfrentado pela empresa na época era a perca de informações internas por falta de documentação ou pela saída de algum funcionário. A solução proposta por Berners-Lee foi fazer uma rede de documentos interligados por hyperlinks em que cada setor da empresa poderia adicionar novos documentos.

A estrutura básica que Berners-Lee montou a 22 anos evoluiu a passos largos em relação à escalabilidade e padronização de protocolos e linguagens, tendo hoje cerca de 2 bilhões de usuários, mais de 30% da população do planeta.

Apesar do avanço das infra-estruturas e serviços para a Web, ainda há muito o que evoluir. Uma das propostas de mudanças é prover uma maior expressividade da linguagem que descreve os documentos na Web [Heflin 2004]. Hoje, esses documentos não possuem um significado que possa ser extraído de forma concisa, apresentam ambigüidade, misturam os dados com elementos visuais e muitas vezes não podem ser indexados por engenhos de busca.

2. Web Semântica

> *"I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web, the content, links, and transactions between people and computers. A* ***Semantic Web*** *which should make this possible has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The* ***intelligent agents*** *people have touted for ages will finally materialize." Tim Berners-Lee*
>
> Tradução literal: *"Eu tenho um sonho para a Web [em que os computadores] tornam-se capazes de analisar todos os dados na Web, o conteúdo, links, e as transações entre pessoas e computadores. A* ***Web Semântica*** *que deve tornar isso possível ainda está para surgir, mas quando isso acontecer, os mecanismos dia-a-dia da burocracia do comércio e nossas vidas diárias serão tratados por máquinas falando com máquinas. Os* ***agentes inteligentes*** *que as pessoas têm falado por anos vão finalmente se concretizar." Tim Berners-Lee*

A Web Semântica citada no texto de Berners-Lee acima é uma iniciativa de pesquisadores da área de inteligência artificial e lingüística computacional que estudam como adequar a Web de hoje a uma infra-estrutura que a tornará mais acessível às máquinas. Essa nova roupagem que os pesquisadores querem dar à Web permitirá que serviços mais sofisticados possam ser construídos, como os que serão descritos a seguir.

2.1. Aplicações

2.1.1. Gerenciamento de Conhecimento

Gerenciamento de conhecimento está relacionado à aquisição, acesso e manutenção de conhecimento dentro de uma empresa ou organização. Essa atividade se tornou e está se estabelecendo como uma necessidade básica em grandes empresas visto que o conhecimento que é gerado internamente agrega valor, pode se tornar um diferencial competitivo e também pode aumentar a produtividade de seus colaboradores. Com o uso de tecnologias criadas para a Web Semântica, soluções para G.C. podem melhorar em vários aspectos, entre eles:

- Organização do conhecimento existente a partir de seu significado;
- Geração de novas informações de forma automática;
- Checagem de inconsistências semânticas em documentos;
- Substituição de consultas baseadas em palavras-chave por perguntas em linguagem natural;

2.1.2. Comércio eletrônico *Business to Consumer* (B2C)

O comércio eletrônico entre vendedores e consumidores é um dos modelos de negócio na Internet que melhor se estabeleceu, sites como amazon [1], americanas [2] e mercado livre [3] possuem público fiel e que os visitam por vários objetivos. É muito comum para a geração que cresceu imersa na Web entrar em sites de compra como esses a procura do melhor preço antes de decidir fazer uma compra. Muitas vezes o produto não é adquirido em uma loja virtual, mas a pesquisa inicial de preços é que muitas vezes determina a escolha do produto. Observando esse comportamento, sites como o buscapé [4] fazem o trabalho de indicar qual é a loja que está com o melhor preço.

A Web Semântica pode ajudar nesse cenário provendo interfaces de consulta mais completas aos sites que fazem comparação de preços, porém, com muito mais detalhes técnicos sobre o produto. Supondo que cada produto tem, por exemplo, uma ontologia que o descreve em detalhes (provida pelo fabricante ou por sites de review de produtos), o consumidor poderá fazer comparações muito mais detalhadas, ajudando-o a encontrar o produto que vai suprir a sua necessidade.

2.1.3. Comércio eletrônico *Business to Business* (B2B) e agentes pessoais

A maioria das pessoas que compram serviços ou produtos na Web só conhecem o comércio eletrônico do tipo B2C, mas existem tecnologias para comércio do tipo B2B, Business to Business, agentes computacionais de empresas que se comunicam para fechar acordos e otimizar o ciclo de negócios que muitas vezes já podem ser previstos e modelados.

Com a popularização da Web Semântica e a introdução de agentes pessoais e agentes que representam negócios, eles poderão se comunicar de forma mais natural e aplicações para otimizar tarefas do dia-a-dia que poderão ser produzidos. Por exemplo, um médico que possua um agente pessoal que negocie a sua agenda com os agentes pessoais de seus clientes pode ser utilizado para remarcar seus atendimentos em caso de uma viagem ou imprevisto, agindo como uma secretária virtual.

2.2. Tecnologias

Aplicações como as citadas acima já existem, mas o trabalho de engenharia para conseguir bons resultados é alto devido às tecnologias que são adotadas hoje. Vamos usar o case do *site* de comparação de preços BuscaPé na próxima seção da monografia.

2.2.1. Metadados Explícitos

A primeira tarefa de engenharia de um agente coletor de preço, como os do buscaPé, é fazê-lo visitar vários sites de compras todo dia à procura de modificações nas listas de produtos para saber quais estão disponíveis naquela loja. É feito então o *parsing* do

[1] site: amazon.com
[2] site: americanas.com
[3] site: mercadolivre.com.br
[4] site: buscape.com.br

```
<div class="chosenProds infoP">
    <strong>Notebook Itautec W7440 c/ Intel® Core i3 370M 2.5GHz 4GB 500GB DVD-RW LED 14&q
uot; Windows 7 Premium - Itautec</strong>
</div>
<div class="infoProd">
    <p><strong>Design e tecnologia na ponta dos seus dedos</strong><br><br>
        ....
</div>
<dl>
    <dt>Marca</dt>
    <dd class="">Itautec</dd>
    <dt>Processador</dt>
    <dd class="">Core i3 370M 2.4GHz.</dd>
    <dt>Barramento</dt>
    <dd class="">2.5 GT/s</dd>
    <dt>Cache</dt>
    <dd class="">3 MB</dd>
    ...
</dl>
```

Figure 1. Código HTML de uma página da americanas.com

HTML de cada site de compras à procura das informações de preço, descrição, avaliação e detalhes de cada produto. A limitação dessa abordagem é que sempre que um dos sites de compra mudar o *layout* (estrutura do HTML), um novo script de *parsing* deverá ser escrito. A grande demanda técnica de uma aplicação como essa é a escrita de agentes muito especializados para atingir bons resultados.

Na figura 1.1 está parte do código em HTML de uma página de produtos da americanas.com. As informações do produto estão cercados apenas de código para a renderização dessa página pelo navegador. Ou seja, a única preocupação dos engenheiros da americanas.com foi a leitura por humanos da informação do produto. Uma aplicação que deseje usar as informações dos produtos da loja vai ter que fazer um agente especializado no *parsing* desse código.

Motores de busca que também se baseiam em *parsing* de páginas para extrair informações da Web dificilmente saberão, por exemplo, qual é o preço de um produto nesse site da americanas.com, já que há várias informações de preço na página e o *parsing* que é feito não é otimizado para sites específicos.

A conseqüência para o usuário final é que ele terá que usar um site específico como o buscapé ou terá que fazer buscas a um engenho de busca por palavras-chave para achar os sites de compra que possuam um produto e em uma segunda etapa, fazer a análise de preços manualmente.

A abordagem da Web Semântica para resolver problemas como esse não é fazer agentes especializados (como os do buscapé), e sim, anotar metadados semânticos dos documentos disponíveis na Web. O exemplo dado anteriormente seria escrito na figura 1.2.

2.2.2. Ontologias

O termo ontologia vem da filosofia, nesse contexto, é um ramo da filosofia que se dedica a estudar a natureza da existência, concentra-se em identificar e descrever o qu existe no universo. Em computação, uma ontologia é um artefado para descrever um domínio. Consiste em uma lista finita de termos e relações entre eles. Os termos denotam conceitos

```
<product>
    <class>notebook</class>
    <vendor>Itautec</vendor>
    <model>W7440</model>
    <attributes>
        <attr name="processor">Intel Core i3 370M 2.4GHz</attr>
        <attr name="barramento">2.5 GT/s</attr>
        <attr name="cache">3 MB</attr>
    </attributes>
</product>
```

Figure 2. Exemplo de código com semântica para um produto

importantes de um domínio [Antoniou and Harmelen 2008].

Grande parte dos trabalhos referentes à Web Semantica estão ligados a ontologias, inclusive este. As linguagem de descrição de ontologias mais importântes para a Web são:

- XML: usado para dirigir a sintaxe de documentos estruturados. Não impõe restrições semânticas no conteúdo do documento;
- XML Schema: linguagem para impor restrições na estrutura dos documentos XML;
- RDF: modelo de dados para recursos (objetos) e relações entre eles. As restrições semânticas são fixas e podem ser representados a partir da sintaxe do XML;
- RDF Schema: descreve as propriedades e classes dos objetos RDF;
- OWL: linguagem rica para modelagem de classes, propriedades, relações entre classes (e.g. disjunção), restrições de cardinalidade, características de propriedades (e.g. simetria). Mais detalhes sobre a OWL serão dados no capítulo 2 dessa monografia.

2.2.3. Lógica

Lógica é a disciplina que estuda os princípios do raciocínio. Ela provê linguagens formais para expressar conhecimento, a semântica formal para a interpretação de sentenças sem precisar realizar operações sobre a base de conhecimento e a transformação de conhecimento implícito em conhecimento explícito, através de deduções a partir da base de conhecimento [Antoniou and Harmelen 2008].

Lógica é mais geral que ontologias, ela pode ser usada por agentes inteligentes para tomada de decisões e escolha de ações. Por exemplo, um agente de B2C pode dar um desconto a um cliente baseado na seguinte regra:

$$\forall x \forall y, cliente(x) \wedge produto(y) \wedge clienteFiel(x) \rightarrow desconto(x, y, 5\%)$$

Onde *cliente(x)* indica que x é um cliente/consumidor, *produto(y)* indica que y é um produto de uma loja, *clienteFiel(x)* indica que x é um cliente fiel da loja e *desconto(x, y, 5%)* indica que o cliente x terá um desconto de 5% no produto y.

2.2.4. Agentes

Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por meio de atuadores [Russell and Norvig 2002]. Agentes lógicos são aqueles que executam ações através de uma base de conhecimento e possuem um requisito fundamental, quando ele formula uma pergunta para a base de conhecimento, a resposta deve seguir o que já foi informado anteriormente.

Agentes para a Web Semântica utilizam as três tecnologias que já foram descritas:

- Metadados serão usados para identificar e extrair informações da Web;
- Ontologias serão usadas para dar assistência às consultas realizadas à Web, interpretar informações recuperadas e para comunicação com outros agentes;
- Lógica será usada para processar informações recuperadas, chegar a conclusões e tomar decisões;

3. Organização da Monografia

Esta monografia está dividida em cinco capítulos. No Capítulo 1, é apresentada uma visão geral sobre a Web Semântica, exemplificando com aplicações e citando as tecnologias que estão sendo usadas. No Capítulo 2, são apresentados conceitos referentes a Lógica de Descrição e a sua ligação com a linguagem de descrição de ontologias OWL. No Capítulo 3 são descritos os algoritmos para normalização de ontologias para a Forma Normal Positiva. No Capítulo 4 o LeanCop é apresentado e é mostrada a validação do trabalho realizado. O Capítulo 5 apresenta as considerações finais sobre o trabalho, bem como propostas de trabalhos futuros.

4. Lógica de Descrição ALC

Lógica de Descrição é uma família de linguagens de representação de conhecimento que pode ser usada para representar o conhecimento de domínio de uma aplicação de forma estruturada e formal [Baader et al. 2003].

A motivação para estudar lógica de descrição neste trabalho vem da Web Semântica. Para que as máquinas possam fazer inferências sobre os documentos da Web, é preciso que a linguagem de descrição dos documentos vá além da semântica básica definida pelo RDF Schema e consiga definir e descrever classes e propriedades sobre os objetos encontrados na Web.

5. Sintaxe da Lógica de Descrição

Nessa seção será mostrada a sintaxe básica da lógica de descrição. A tabela 2.1 mostra o alfabeto de símbolos usado pela linguagem.

Os elementos mais básicos são os conceitos atômicos e propriedades atômicas. Descrições de conceitos podem ser construídas indutivamente a partir dos construtores com conceitos e propriedades.

alfabeto	
a, b	indivíduos
A, B	conceitos atômicos
C, D	descrição de conceitos
R, S	papeis (propriedades)
f, g	símbolos de funções
conectivos	
$\sqcap$	interseção
$\sqcup$	união
$\neg$	negação
relações	
$\sqsubseteq$	inclusão
$\equiv$	equivalência

Table 1. Notação da lógica de descrição

$$
\begin{array}{lll}
C, D \rightarrow & A — & \text{(conceito atômico)} \\
& \top — & \text{(conceito universal)} \\
& \bot — & \text{(conceito vazio)} \\
& \neg A — & \text{(negação de conceito atômico)} \\
& C \sqcap D — & \text{(interseção de conceitos)} \\
& \forall R.C — & \text{(restrição de valor)} \\
& \exists R.\top & \text{(restrição existencial)}
\end{array}
$$

Uma interpretação ι consiste em um conjunto não vazio Δ^ι (domínio da interpretação) e uma função de interpretação, que para conceito atômico A é o conjunto $A^\iota \subseteq \Delta^\iota$ e para cada propriedade atômica R é a relação binária $R^\iota \subseteq \Delta^\iota \times \Delta^\iota$. As funções de interpretação se estendem a descrição de conceitos a partir das definições indutivas [Baader et al. 2003] como as que estão abaixo:

$$
\begin{aligned}
\top^\iota &= \Delta^\iota \\
\bot^\iota &= \emptyset \\
\neg A^\iota &= \Delta^\iota \backslash A^\iota \\
(C \sqcap D)^\iota &= C^\iota \cap D^\iota \\
(\forall R.C)^\iota &= \{a \in \Delta^\iota | \forall b.(a,b) \in R^\iota \rightarrow b \in C^\iota\} \\
(\exists R.\top)^\iota &= \{a \in \Delta^\iota | \exists b.(a,b) \in R^\iota\}
\end{aligned}
$$

Esse trabalho é restrito à família ALC, que compreende os conceitos e propriedades atômicas, negação de conceitos, interseção, união, restrições de valor e existencial, *top* (verdade) e *bottom* (absurdo). A tabela 2.2 mostra além de ALC, outras famílias de DL existentes [Baader et al. 2003].

Uma ontologia ou base de conhecimento em ALC é composta pela tripla (N_C, N_R, N_O), onde N_C é o conjunto de conceitos, N_R é o conjunto de predicados, e N_O é o conjunto de indivíduos, que são as instâncias de N_C e N_R. A base de conhecimento ou ontologia também pode ser descrita como o par (τ, α), onde τ é a terminologia do domínio (TBox), equivalente a $N_C \cup N_C$ e α é a instanciação da base, que corresponde a N_O, também conhecida como *assertional box* (ABox).

Nome	Sintaxe	Semântica	Expressividade
Verdade	$\top$	Δ^ι	AL
Absurdo	$\bot$	$\emptyset$	AL
Conceito	C	$C^\iota \subseteq \Delta^\iota$	AL
Relação	R	$R^\iota \subseteq \Delta^\iota x \Delta^\iota$	AL
Interseção	$C \sqcap D$	$C^\iota \cap D^\iota$	AL
União	$C \sqcup D$	$C^\iota \cup D^\iota$	U
Negação	$\neg C$	$\Delta^\iota \backslash A^\iota$	C
Restrição de valor	$\forall R.C$	$\{a \in \Delta^\iota \vert \forall b.(a,b) \in R^\iota \rightarrow b \in C^\iota\}$	AL
Restrição existencial	$\exists R.C$	$\{a \in \Delta^\iota \vert \exists b.(a,b) \in R^\iota \wedge b \in C^\iota\}$	ϵ
Restrição	$\geq nR$	$\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota\}\vert \geq n\}$	
numérica	$\leq nR$	$\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota\}\vert \leq n\}$	N
não qualificada	$= nR$	$\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota\}\vert = n\}$	
Restrição	$\geq nR.C$	$\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota \wedge b \in C^\iota\}\vert \geq n\}$	
numérica	$\leq nR.C$	$\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota \wedge b \in C^\iota\}\vert \leq n\}$	Q
qualificada	$= nR.C$	$\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota \wedge b \in C^\iota\}\vert = n\}$	

Table 2. Sintaxe e semântica de alguns contructos de lógica de descrição com anotação de expressividade

Os axiomas são compostos por elementos de N_O e um conjunto finito de GCIs (*general concept inclusions*). Podem assumir a forma $C \sqsubseteq D$ ou $C \equiv D$ (uma equivalência ($\equiv$) é o mesmo que $(C \sqsubseteq D) \wedge (D \sqsubseteq C)$), onde C, D são conceitos e $\sqsubseteq$ é uma inclusão.

6. OWL: Web Ontology Language

A *Web Ontology Language*, OWL, foi escolhida pela w3c [5], grupo que regula os padrões na Web, como a linguagem de descrição de ontologias para a Web Semântica [Heflin 2004]. Alguns dos requisitos que ela atendeu foram [Antoniou and Harmelen 2008]: i) sintaxe bem definida; Como o objetivo da Web Semântica é tornar os documentos da Web mais fáceis de serem processados por máquinas, este é um requisito básico. ii) semântica formal; Descrever a base de conhecimento de forma logicamente precisa é fundamental para fazer inferências como dedução de conceitos, checagem de consistência na base de conhecimento e instanciação de indivíduos a uma classe. iii) suporte a raciocínio; Uma vez que a linguagem possui uma semântica formal, atividades de raciocínio podem ser realizadas. iv) expressividade; alguns domínios precisam de construtos mais elaborados para que possam ser descritos. Quanto maior a expressividade da linguagem, naturalmente fica mais fácil de descrever um domínio, apesar de aumentar a complexidade e tempo de processamento.

Entre os requisitos citados no parágrafo anterior estão expressividade e suporte a raciocínio. Apesar de ambos poderem estar na linguagem, são antagônicos, quanto maior for a expressividade da linguagem, mais complexas e demoradas serão as atividades de raciocínio sobre a linguagem. Para criar fronteiras nesse conflito entre expressividade e complexidade de raciocínio, a w3c criou três versões de OWL: OWL Full, OWL DL e

[5] sitio oficial: http://w3.org

OWL Lite.

6.1. OWL Full

A Web Ontology Language em sua versão mais expressiva, usando todas as primitivas da linguagem, é chamada de OWL Full. Essa combinação inclui, por exemplo, aplicar uma restrição de cardinalidade na classe que contém todas as outras classes, limitando a quantidade de classes que a ontologia pode ter.

OWL Full é completamente compatível com RDF, tanto sintaticamente, quanto em sua semântica. A desvantagem de OWL Full é que ela é tão poderosa que é indecidível em relação às atividades de raciocínio.

6.2. OWL DL

OWL DL (DL é a sigla para *Description Logic*, Lógica de Descrição em português) é a família de OWL que corresponde à lógica de Descrição. A sua grande vantagem é que ela é decidível, dando a possibilidade de realização de atividades de raciocínio de forma mais eficiente. A desvantagem de OWL DL é que ela perde a compatibilidade com RDF, qualquer documento em OWL DL pode ser descrito como um documento em RDF, mas o contrário não é verdade.

6.3. OWL Lite

OWL Lite é uma família de OWL que é mais limitada do que OWL Full e OWL DL. Ela não dá suporte a, por exemplo, disjunção entre classes, união e complemento. A grande vantagem dessa linguagem é uma maior facilidade para o desenvolvimento de ferramentas, e a sua desvantagem é a perda de expressividade.

7. Normalização para o método das conexões

O método das conexões proposto por W. Bibel [**?**] é um método para prova automática de teoremas descritos em lógica de primeira ordem [**?**]. Um dos trabalhos recentes de Freitas et al [**?**] foi a extenção desse método para lógica de descrição ALC.

O artigo intitulado *A Connection Method for Reasoning with the Description Logic ALC* [**?**] propõe algoritmos tanto para o método, quanto para a normalização que precisa ser feita na base de conhecimento para que seja possível a representação necessária para o método das conexões usando apenas uma matriz.

O objetivo deste trabalho é implementar algum algoritmo de normalização para o método das conexões como os citados no texto de Freitas et al. Dois algoritmos foram propostos com esse objetivo [**?**]; o primeiro utiliza-se de uma tabela com nove regras que devem ser aplicadas à base de conhecimento a fim de obter a forma normal positiva. O segundo, intitulado "*A more complex and efficient normalization*" não cria novos símbolos durante a sua execução, fazendo-o mais eficiente que o primeiro em relação ao uso de memória.

No cronograma deste trabalho estava prevista a implementação desse segundo algoritmo, porém, ao decorrer do desenvolvimento, propomos um terceiro algoritmo que é ainda mais eficiente em relação ao uso de memória, ele é linear em relação à quantidade de impurezas na base, enquanto que o "*A more complex and efficient normalization*" é quadrático. O restante desse capítulo se dedicará a dar definições para o entendimento dos últimos dois algoritmos comentados acima e também descrevá as suas implementações.

8. Tradução de ontologias ALC para forma normal disjuntiva

Para que o leitor consiga entender melhor os algoritmos de tradução, alguns conceitos precisam ser fixados.

Métodos diretos como o método das conexões são formulados para provar que uma fórmula ou um conjunto de fórmulas é um teorema, sse cada interpretação gerada é uma tautologia. Tautologias normalmente tomam a forma $L \vee \neg L$, nesse caso, a fórmula precisa estar na Forma Normal Disjuntiva (ou, do inglês, Disjunctive Normal Form - DNF).

Definição 1 (Forma Normal Disjuntiva, cláusula)**.** *Uma fórmula em DNF é uma conjunção de disjunções. Ou seja, tomam a forma:*

$$\bigcup_{i=1}^{n} C_i \text{ , ou, } C_1 \vee ... \vee C_n.$$

onde cada C_i é uma cláusula. Uma cláusula é uma conjunção de literais. Ou seja, tomam a forma:

$$\bigcap_{j=1}^{m} L_{i,j} \text{ , ou, } L_{i,1} \wedge ... \wedge L_{i,m} \text{ , também representado por } \{L_{i,1}, ..., L_{i,m}\}$$

onde cada $L_{i,j}$ é um literal, resultando na fórmula:

$$\bigcap_{i=1}^{n} \bigcup_{j=1}^{m} L_{i,j} \text{ , ou, } (L_{1,1} \wedge ... \wedge L_{1,m}) \vee (L_{n,1} \wedge ... \wedge L_{n,m})$$

podendo ser chamada também de forma causal disjuntiva, representada por:

$$\{\{(L_{1,1}, ..., L_{1,m}\}, ..., \{L_{n,1}, ..., L_{n,m}\}\}$$

A definição acima é a definição herdada da lógica de primeira ordem, para ser válida também para a lógica de descrição o conceito de conjunções e disjunções deve ser estendido.

Definição 2 (Conjunção ALC)**.** *Uma conjunção ALC é um literal L, uma conjunção $(E_0\wedge, ..., \wedge E_n)$, ou uma restrição existencial $\exists x.E$, onde E é uma expressão qualquer em lógica de descrição.*

Definição 3 (Disjunção ALC)**.** *Uma disjunção ALC é um literal L, uma disjunção $(E_0\vee, ..., \vee E_n)$, ou uma restrição de valor $\forall x.E$, onde E é uma expressão qualquer em lógica de descrição.*

Definição 4 (Conjunção ALC pura, Conjunção ALC não pura)**.** *Uma conjunção ALC pura é uma conjunção ALC que na forma normal negada não contém restrições de valor ($\forall x.E$) e também não contém disjunções $(E\vee, ..., \vee E)$. O conjunto de conjunções ALC puras é representado por $\hat{C}$. Uma conjunção ALC não pura é uma conjunção ALC que não é pura.*

Definição 5 (Disjunção ALC pura, Disjunção ALC não pura)**.** *Uma disjunção ALC pura é uma disjunção ALC que na forma normal negada não contém restrições existenciais ($\exists x.E$) e também não contém conjunções ($E\wedge, ..., \wedge E$). O conjunto de disjunções ALC puras é representado por $\check{D}$. Uma disjunção ALC não pura é uma disjunção ALC que não é pura.*

Definição 6 (Impureza em uma expressão não pura)**.** *Impureza em expressões ALC não puras são conjunções em disjunções não puras ou disjunções em conjunções não puras. O conjunto de impurezas é chamado de conjunto de impurezas ALC e é representado por I.*

Definição 7 (Forma Normal Positiva)**.** *Um axioma ALC está na Forma Normal Positiva sse ele está em uma das seguintes formas:*

$$\textit{i)}\ A \sqsubseteq \hat{C}$$
$$\textit{ii)}\ \check{D} \sqsubseteq A$$
$$\textit{iii)}\ \hat{C} \sqsubseteq \check{D}$$

onde A é um conceito atômico, $\hat{C}$ é uma conjunção ALC pura, $\check{D}$ é uma disjunção ALC pura.

O método das conexões utiliza matrizes para realizar provas de teoremas. No início deste trabalho, ainda não era possível fazer as provas com matrizes aninhadas, ou seja, havia sempre a necessidade de normalizar a base de conhecimento na forma normal positiva (definição 7). No entanto, Jens Otten em um trabalho entitulado *A Non-clausal Connection Calculus* [**?**] mostrou como aplicar o método das conexões sem o passo da normalização. Apesar da recente evolucão do método, para o objetivo deste trabalho, ainda é necessário fazer a normalização, já que para lógica de descrição, o método ainda não foi modificado para usar matrizes aninhadas.

A próxima seção desta monografia descreve o algoritmo que propomos para a normalização para a forma normal positiva.

8.1. Algoritmo Proposto

Esta seção da monografia descreve a implementação realizada neste trabalho.

O método das conexões é um método direto, ou seja, uma consulta à base de conhecimento verifica se uma fórmla é uma tautologia e toma a forma $KB \rightarrow \alpha$, onde α é um axioma e KB (*Knowledge base*) é da forma $\bigcap_{i=1}^{n} A_i$, onde A_i também é um axioma. Expandindo a fórmula $\neg KB \vee \alpha$, temos:

$\neg \bigcap_{i=1}^{n} A_i \vee \alpha$ [ou, $\neg(A_1 \wedge ... \wedge A_n) \vee \alpha$], que pode ser transformada para:

$\bigcup_{i=1}^{n} \neg A_i \vee \alpha$ [ou, $\neg A_1 \vee ... \vee \neg A_n \vee \alpha$]

Cada A_i ou α precisam estar na forma normal positiva, ou seja, precisam estar em uma das formas: i) $A \sqsubseteq \hat{C}$, ii) $\check{D} \sqsubseteq A$, ou, iii) $\hat{C} \sqsubseteq \check{D}$, onde A é um conceito, $\hat{C}$ é uma conjunção pura e $\check{D}$ é uma disjunção pura.

O primeiro passo do algoritmo é a separação de axiomas de equivalência de expressões e inclusões. Os axiomas de equivalência serão substituídos por dois axiomas de inclusão, e.g., $(A \equiv B \rightarrow A \sqsubseteq B \wedge B \sqsubseteq A)$.

Algorithm 1 Normalize-Ontology(O)

Require: O: ontologia em lógica de descrições ALC

1. **for all** $A \sqsubseteq B \in S_I$ **do** {A, B são expressões, S_I é o conjunto dos axiomas de inclusão pertencentes a O}
2. Normalize-Axiom ($A \sqsubseteq B$)
3. **end for**
4. **for all** $A \equiv B \in S_{EQ}$ **do** {A, B são expressões, S_{EQ} é o conjunto dos axiomas de equivalência pertencentes a O}
5. Normalize-Axiom ($A \sqsubseteq B$)
6. Normalize-Axiom ($B \sqsubseteq A$)
7. **end for**

Algorithm 2: O método *Normalize-Axiom($A \sqsubseteq B$)* remove o axioma da ontologia (linha 1), remove as impurezas do dado esquedo (linha 2) e do lado direto (linha 3) do axioma. Caso ele já esteja na forma normal positiva (linha 3) apenas pela remoção das impurezas, ele será adicionado à base novamente (linha 5), caso contrário, dois axiomas serão criados, um com a expressão do lado esquerdo (linha 7) e outro com a expressão do lado direito (linha 8). Caso esses novos axiomas sejam criados, eles já estarão na forma normal positiva, já que uma expressão pura em uma relação de inclusão com um conceito está na forma normal positiva em qualquer combinação.

Algorithm 2 Normalize-Axiom ($A \sqsubseteq B$)

Require: $A \sqsubseteq B$: inclusão de A em B

1. $O = \{O - (A \sqsubseteq B)\}$
2. $pure\_left =$ purify $(A, left)$
3. $pure\_right =$ purify $(B, right)$
4. **if** $(pure\_left \sqsubseteq pure\_right) \in S_{PNF}$ **then** {S_{PNF} é o conjunto das fórmulas na forma normal positiva}
5. $O = \{O \cup (pure\_left \sqsubseteq pure\_right)\}$
6. **else**
7. $O = \{O \cup (pure\_left \sqsubseteq N)\}$
8. $O = \{O \cup (N \sqsubseteq pure\_right)\}$
9. **end if**

Algorithm 3: O método *purify (e, side)* recebe uma expressão em DL ALC e a retorna sem impurezas. O parâmetro *side* é necessário para remoção de uma impureza, caso a impureza esteja no lado esquerdo, e.g., $C \sqcap Impurity \sqcap C \sqcap C \sqsubseteq A$, um axioma será criado com a impureza do lado esquerdo, e.g., $(C \sqcap NewConcept \sqcap C \sqcap C \sqsubseteq A) \wedge (Impurity \sqsubseteq NewConcept)$, caso a impureza esteja no lado direito, o comportamento será o mesmo mas de forma simétrica, o seja, um axima do tipo $NewConcept \sqsubseteq Impurity$ será criado e a impureza será substituída por um novo conceito ($NewConcept$).

As impurezas são identificadas através dos métodos *visit*. A primeira chamada para esse conjunto de métodos é na linha 3 do método *purity*, caso a expressão de entrada seja uma conjunção não pura, ou na linha 5 do método *purify* caso a expressão seja uma disjunção não pura. O método *visit* em sua implementação em Java [6] utiliza o padrão de

[6] código disponível em: http://github.com/adrianomelo/tg

projetos visitor, que permite criar novas operações sem mudar a definição das estruturas de dados [**?**].

Uma pilha chamada *expressions_stack* é usada no método pois em uma implementação em Java os métodos que visitam cada nó da ontologia não podem retornar objetos por uma limitação da biblioteca OWLAPI 3.0 [7] [8].

Algorithm 3 purify (e, side)

Require: e: uma expressão em DL ALC
Require: side: é $left$ ou $right$
1. expressions_stack = STACK()
2. **if** $e \in S_{npc}$ **then** $\{S_{npc}$ é o conjunto das conjunções não puras$\}$
3. visit (expressions_stack, e, side, $conjunction$)
4. **else if** $e \in S_{npd}$ **then** $\{S_{npd}$ é o conjunto das disjunções não puras$\}$
5. visit (expressions_stack, e, side, $disjunction$)
6. **end if**
7. **return** POP (expressions_stack)

Algorithm 4: O método *Extract-Impurity()* é usado para criar um novo axioma na ontologia com a impureza achada por um dos métodos *visit*. O parâmetro *side* é utilizado para saber em qual lado do axioma a impureza deverá ficar. Após o a criação do axioma, o método *Normalize-Axiom()* é chamado com o novo axioma para garantir que ele também esteja na forma normal positiva.

Algorithm 4 Extract-Impurity (e, side)

Require: e: uma expressão em DL ALC
Require: side: é $left$ ou $right$
1. N = NEW-CONCEPT()
2. **if** side = $left$ **then**
3. $O = \{O \cup (e \sqsubseteq N)\}$
4. Normalize-Axiom($e \sqsubseteq N$)
5. **else**
6. $O = \{O \cup (N \sqsubseteq e)\}$
7. Normalize-Axiom($N \sqsubseteq e$)
8. **end if**
9. **return** N

Algorithm 5, 6, 7, 8, 9, 10: O conjunto de métodos chamados de *visit* diferem em qual expressão em lógica de descrições cada um suporta. O método *visit (stack,* $\bigcup^{n} e_i$*, side, kind)*, por exemplo, dá suporte às uniões entre expressões.

Quatro parâmetros são passadados para os métodos, uma pilha *stack* que é usada para auxiliar a construção de uma nova expressão normalizada (os métodos não podem retornar objetos por uma limitação da OWLAPI 3.0, biblioteca usada na implementação em

[7] documentação do método visit: http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/model/OWLClassExpressionVisitor.html

[8] javadoc da documentação da OWLAPI 3.0: http://owlapi.sourceforge.net/javadoc/

Java). Uma expressão em lógica de descrições, que pode ser uma união $\bigcup^{n} e_i$, interseção $\bigcap^{n} e_i$, restrição de valor $\forall rE$, restrição existencial $\exists rE$, complemento $\neg e$ ou conceito C. O lado *side* em que a expressão original estava (*left* ou *right*). E o tipo *kind* de expressão que o método estava esperando.

Quando uma conjunção é passada para o método *visit* e o tipo (*kind*) é igual a *disjunction*, significa que uma impureza foi encontrada e ela deve ser removida pelo método *Extract-Impurity()*. De forma análoga, quando uma disjunção é visitada por um dos métodos *visit* e o tipo esperado é *conjunction*, significa que uma impureza foi encontrada e ela deve ser removida pelo método *Extract-Impurity()*.

Algorithm 5 visit (stack, $\bigcup^{n} e_i$, side, kind)

Require: stack: uma pilha
Require: side: é $left$ ou $right$
Require: $\bigcup^{n} e_i$: união de expressões e_i

1. **if** kind = $conjunction$ **then**
2. N = Extract-Impurity($\bigcup^{n} e_i$, side)
3. PUSH (stack, N)
4. **else**
5. **for all** $e_i \in \bigcup^{n} e_i$ **do**
6. visit (stack, e_i, side, kind)
7. **end for**
8. PUSH (stack, $\bigcup^{n}$ POP(stack))
9. **end if**

Algorithm 6 visit (stack, $\bigcap^{n} e_i$, side, kind)

Require: stack: uma pilha
Require: side: é $left$ ou $right$
Require: $\bigcap^{n} e_i$: interseção de expressões e_i

1. **if** kind = $disjunction$ **then**
2. N = Extract-Impurity($\bigcup^{n} e_i$, side)
3. PUSH (stack, N)
4. **else**
5. **for all** $e_i \in \bigcap^{n} e_i$ **do**
6. visit (stack, e_i, side, kind)
7. **end for**
8. PUSH (stack, $\bigcap^{n}$ POP(stack))
9. **end if**

Para uma melhor comparação da matrizes que são geradas com o algoritmo proposto, o exemplo que Freitas et al [**?**] também será o exemplo deste agoritmo. O axioma de equivalencia no Exemplo 1 é a base de conhecimento usada.

Exemplo 1. *Deseja-se normalizar o axioma:*

Algorithm 7 visit (stack, $\forall rE$, side, kind)

Require: stack: uma pilha
Require: side: é $left$ ou $right$
Require: $\forall rE$: restrição de valor com a relação r e a expressão E

1. **if** kind = $conjunction$ **then**
2. N = Extract-Impurity($\bigcup^{n} e_i$, side)
3. PUSH (stack, N)
4. **else**
5. E' = NNF(E)
6. visit (stack, E', side, kind)
7. E'' = POP(stack)
8. PUSH (stack, $\forall rE''$)
9. **end if**

Algorithm 8 visit (stack, $\exists rE$, side, kind)

Require: stack: uma pilha
Require: side: é $left$ ou $right$
Require: $\exists rE$: restrição existencial com a relação r e a expressão E

1. **if** kind = $disjunction$ **then**
2. N = Extract-Impurity($\bigcup^{n} e_i$, side)
3. PUSH (stack, N)
4. **else**
5. E' = NNF(E)
6. visit (stack, E', side, kind)
7. E'' = POP(stack)
8. PUSH (stack, $\exists rE''$)
9. **end if**

Algorithm 9 visit (stack, $\neg e$, side, kind)

Require: stack: uma pilha
Require: side: é $left$ ou $right$
Require: $\neg e$: negação de uma expressão e

1. **if** $E \notin S_{concept}$ **then**
2. e' = NNF($\neg e$)
3. visit (stack, e', side, kind)
4. **else**
5. PUSH(stack, $\neg e$)
6. **end if**

Algorithm 10 visit (stack, C, side, kind)

Require: stack: uma pilha
Require: side: é $left$ ou $right$
Require: C: é um conceito

1. PUSH (stack, C)

$$PizzaMargherita \equiv Pizza \sqcap \exists hasTopping.Tomate \sqcap \exists hasTopping.Muzzarella \sqcap \neg\exists hasTopping.\neg(Tomate \sqcup Muzzarella)$$

O primeiro passo do algoritmo (método Normalize-Ontology(), página 12) separa o axioma de equivalência em dois axiomas de inclusão. Obtendo-se então:

$$PizzaMargherita \sqsubseteq Pizza \sqcap \exists hasTopping.Tomate \sqcap \exists hasTopping.Muzzarella \sqcap \neg\exists hasTopping.\neg(Tomate \sqcup Muzzarella)$$

$$Pizza \sqcap \exists hasTopping.Tomate \sqcap \exists hasTopping.Muzzarella \sqcap \neg\exists hasTopping.\neg(Tomate \sqcup Muzzarella) \sqsubseteq PizzaMargherita$$

O método Normalize-Axiom() será chamado então com cada um dos axiomas obtidos. O lado direito no primeiro axioma e o lado esquerdo do segundo são conjunções, então o método visit de interseções será chamado com o kind sendo conjunction tanto para a primeira expressão quanto para a segunda.

Quando o método visit (stack, $\neg e$, side, kind) for chamado com a expressão $\neg\exists hasTopping.\neg(Tomate \sqcup Muzzarella)$, tanto para o lado esquerdo, quanto para o lado direito ele a transformará na expressão $\forall hasTopping.(Tomate \sqcap Muzzarella)$, que é uma disjunção. Os axiomas que serão obtidos são:

$$PizzaMargherita \sqsubseteq Pizza \sqcap \exists hasTopping.Tomate \sqcap \exists hasTopping.Muzzarella \sqcap Extracted1$$

$$Extracted1 \sqsubseteq \forall hasTopping.(Tomate \sqcup Muzzarella)$$

$$Pizza \sqcap \exists hasTopping.Tomate \sqcap \exists hasTopping.Muzzarella \sqcap Extracted2 \sqsubseteq PizzaMargherita$$

$$\forall hasTopping.(Tomate \sqcup Muzzarella) \sqsubseteq Extracted2$$

Em formato matricial, temos (axiomas 1 e 2):

$$\begin{bmatrix} PM & PM & PM & PM & PM & PM & Extracted1 \\ \neg Pizza & \neg hasTopping & \neg Tomate & \neg hasTopping & \neg Muzzarella & \neg Extracted1 & \neg hasTopping \\ & & & & & & \neg Tomate \\ & & & & & & \neg Muzzarella \end{bmatrix}$$

e (axiomas 3 e 4):

$$\begin{bmatrix} Pizza & \neg hasTopping & Tomate & Muzzarella \\ hasTopping & \neg Extracted2 & \neg Extracted2 & \neg Extracted2 \\ Tomate & & & \\ hasTopping & & & \\ Muzzarella & & & \\ Extracted2 & & & \\ \neg PizzaMargherita & & & \end{bmatrix}$$

*O algoritmo de Freitas et al [**?**] chegou às mesmas matrizes. O que indica que ambos são lineares em relação às impurezas deste exemplo.*

Exemplo 2. *No exemplo dado por Freitas et al [**?**], o seu algoritmo se comportava de forma linear (em espaço) em relação às impurezas nas expressões. Porém, com o caso $\hat{C} \sqsubseteq \hat{D}$, onde $\hat{C}$ e $\hat{D}$ são conjunções puras, o algoritmo se comporta de forma quadrática em relação a memória.*

O algoritmo proposto irá dividir o axioma em dois, ficando $\hat{C} \sqsubseteq A$, e $A \sqsubseteq \hat{D}$. Já o algoritmo de Freitas et al, irá construir vários axiomas, um para cada expressão do lado direito, e.g., $\{\hat{C} \sqsubseteq D_1, \hat{C} \sqsubseteq D_2, ..., \hat{C} \sqsubseteq D_{m-1}, \hat{C} \sqsubseteq D_m\}$. O que irá resultar em uma matriz do tipo:

$$\begin{bmatrix} C_1 & C_1 & \dots & C_1 & C_1 \\ C_2 & C_2 & \dots & C_2 & C_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ C_{n-1} & C_{n-1} & \dots & C_{n-1} & C_{n-1} \\ C_n & C_n & \dots & C_n & C_n \\ \neg D_1 & \neg D_2 & \dots & \neg D_{m-1} & \neg D_m \end{bmatrix}$$

enquanto que o algoritmo proposto irá produzir uma matriz do tipo:

$$\begin{bmatrix} C_1 & A & A & \dots & A & A \\ C_2 & \neg D_1 & \neg D_2 & \dots & \neg D_{m-1} & \neg D_m \\ \vdots & & & & & \\ C_{n-1} & & & & & \\ C_n & & & & & \\ \neg A & & & & & \end{bmatrix}$$

8.2. Algoritmo Original

Esta seção contém o algoritmo escrito por Freitas et al [**?**] para a normalização de bases de conhecimento em lógica de descrição ALC. A figura 3.1 descreve o método *Normalize-Ontology ()*, que chama os métodos *Normaize-LHS ()* e *Normalize-RHS ()* caso o lado esquerdo e direito, respectivamente, não estejam normalizados para cada axioma da ontologia.

Os métodos *Normalize-LHS ()* e *Normalize-RHS* são duais, ou seja, possuem a mesma lógica mas com o sentido invertido. A figura 3.2 mostra o pseudocódigo do método *Normalize-LHS()* e a figura 3.3 mostra o pseudocódigo do método *Normalize-RHS*.

Note que o primeiro algoritmo da seção 3.1.1 e a figura 3.1 são praticamente o mesmo, porém, o primeiro diferencia dentro do método os axiomas de equivalência e inclusão. No artigo de Freitas et al [**?**] é citado que isso deve ser feito, mas não explicita no algoritmo.

9. Conclusão

Os resultados de boa performance do leanCoP para lógica de primeira ordem dão indícios que o método das conexões pode ganhar espaço entre os métodos de prova para lógica de

```
Normalize-Ontology (Ontology O);
For each axiom A ∈ O  Normalize-Axiom (A,O);
Normalize-Axiom (Axiom A, Ontology O);
If A not in normal form then
      If LHS(A) ∉ N_C U S_C {not a concept or pure conjunction}
        Normalize-LHS (A, O);
      If RHS(A) ∉ N_C U S_D {not a concept or pure disjunction}
        Normalize-RHS (A, O);
```

Figure 3. Método *Normalize-Ontology ()*

```
Normalize-LHS (Axiom A, ontology O);
If LHS(A) ∈ S_C U S_npC (a conjunction) then
      For each D ⊆ LHS(A) | D ∈ S_D U S_npD (disjunction)
        LHS(A) ← (LHS(A) - D) ⊓ N, N ∉ O
        If D ∈ S_npD (non-pure) then
                O ← O U Normalize-LHS({D ⊑ N},O);
        Else    O ← O U {D ⊑ N};
      For each nC ⊆ LHS(A) | nC ∈ S_npC (non-pure conjunction)
        Find the first impurity I ⊆ nC | I ∈ S_I
        nc^σ ← nC{I/N}, N ∉ O
        LHS(A) ← (LHS(A) - nC) ⊓ nc^σ
        O ← O U Normalize-LHS({I ⊑ N},O}
Else {LHS(A) ∈ S_D U S_npD (disjunction)}
      O ← O - A;
      For each X ⊆ LHS(A)| X ∈ N_C U S_C
                (atomic concept or pure conjunction)
        If RHS(A) ∉ N_C U S_C  (not a concept or pure conjunction) then
                O ← O U Normalize-RHS({X ⊑ RHS(A)}, O};
        Else    O ← O U {X ⊑ RHS(A)};
      For each expression E ⊆ LHS(A)| E ∉ S_C
                        (not a pure conjunction)
        If E ⊑ RHS(A) is not in normal form then
                O ← O U Normalize-Axiom({E ⊑ RHS(A)}, O};
        Else    O ← O U E ⊑ RHS(A);
```

Figure 4. Método *Normalize-LHS ()*

Normalize-RHS (Axiom A, ontology O);

If RHS(A) $\in$ $S_D \cup S_{npD}$ (a disjunction) then

For each C $\subseteq$ RHS(A) | D $\in$ $S_C \cup S_{npC}$ (conjunction)

RHS(A) $\leftarrow$ (RHS(A) - C) $\sqcup$ N, N $\notin$ O

If C $\in$ S_{npC} (non-pure) then

O $\leftarrow$ O U Normalize-RHS({N $\sqsubseteq$ C},O);

Else O $\leftarrow$ O U {N $\sqsubseteq$ C};

For each nD $\subseteq$ RHS(A) | nD $\in$ S_{npD} (non-pure disjunction)

Find the first impurity I $\subseteq$ nD | I $\in$ S_I

$nd^{\sigma} \leftarrow$ nD{I/N}, N $\notin$ O

LHS(A) $\leftarrow$ (LHS(A) - nD) $\sqcup nd^{\sigma}$

O $\leftarrow$ O U Normalize({N $\sqsubseteq$ I},O}

Else {RHS(A) $\in$ $S_C \cup S_{npC}$ (conjunction)}

O $\leftarrow$ O - A;

For each X $\subseteq$ RHS(A)| X $\in$ $N_C \cup S_D$
(atomic concept or pure disjunction)

If LHS(A) $\notin$ $N_C \cup S_D$ (not a concept or pure disjunction) then

O $\leftarrow$ O U Normalize-LHS({LHS(A) $\sqsubseteq$ X)}, O};

Else O $\leftarrow$ O U {LHS(A) $\sqsubseteq$ X};

For each expression E $\subseteq$ RHS(A)| E $\notin$ S_D
(not a pure disjunction)

If LHS(A) $\sqsubseteq$ E is not in normal form

O $\leftarrow$ O U Normalize-Axiom({LHS(A) $\sqsubseteq$ E)}, O};

Else O $\leftarrow$ O U LHS(A) $\sqsubseteq$ E;

Figure 5. Método *Normalize-RHS ()*

descrição. Como a w3c [9] ainda não definiu uma tecnologia padrão para a camada de lógica e prova da Web Semântica, trabalhos como o que este está inserido são de importância estratégica para a Web, eles desenvolvem soluções que poderão ser adotados em larga escala pelo mundo.

O leanCoP foi envolvido neste trabalho para realizar atividades de raciocínio, como subsunção e equivalência, após a normalização da base de conhecimento. O leanCoP é escrito em Prolog e as APIs de manipulação de ontologias OWL são escritas em Java. Para fazer o leanCoP usar como base de conhecimento as ontologias OWL normalizadas, foi utilizado o formato TPTP como linguagem intermediária. Porém, para fazer as atividades de raciocínio de forma automática, um número exponencial de arquivos deveriam ser gerados em TPTP para serem usados com o leanCoP. Além disso, um trabalho de *parsing* desses arquivos deveria ser realizado para adicionar cada consulta à base de conhecimento de cada arquivo, o que se mostrou fora do escopo do projeto. O leanCoP foi utilizado então para fazer simples checagem de consistência na base de conhecimento, que traz como efeito colateral a validação da corretude do algoritmo de normalização.

Na proposta inicial deste trabalho estava prevista a implementação do algoritmo de normalização de Freitas et al [**?**], porém, apesar do algoritmo não incluir novos símbolos à base de conhecimento, não é fácil de ser entendido. A seção 3.1.1 mostra em pseudocódigo a implementação que foi feita neste trabalho. O algoritmo foi produzido devido a uma provocação de simplificar o algoritmo de Freitas et al [**?**]. O objetivo dos algoritmos é o mesmo, traduzir os axiomas de uma ontologia ALC para a forma normal positiva, mas o algoritmo da seção 3.1.1 além de ser mais simples de ser implementado, faz a matriz gerada após a normalização consumir menos memória, o que foi uma grande contribuição. A redução do uso de memória vai impactar no tempo de execução do método das conexões para lógica de descrição, já que a busca pelos caminhos na matriz vai ser reduzido.

10. Trabalhos futuros

Este trabalho não contempla todos os constructos de OWL, nem sequer de OWL Lite, já que é limitada à familia ALC. Trabalhos futuros serão para estender o algoritmo de normalização para incluir restrições com cardinalidade, domínio e contradomínio de propriedades, disjunção entre classes e assim por diante.

Este trabalho é apenas um dos módulos necessários para a implementação de um raciocinador escrito em java que use o método das conexões. O algoritmo em si que procura pelas conexões, ou caminhos, ainda não foi implementado.

E, por fim, quando o método das conexões estiver formalizado para uma família de DL que seja equivalente a uma família de OWL e a sua implementação estiver finalizada, poderá haver um trabalho para integrar o raciocinador a editores de ontologias existentes no mercado, como o Protégé.

References

[Antoniou and Harmelen 2008] Antoniou, G. and Harmelen, F. v. (2008). *A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)*. The MIT Press, 2 edition.

[9] site: http://w3.org

[Baader et al. 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge.

[Berners-Lee 1989] Berners-Lee, T. (1989). Information management: A proposal.

[Heflin 2004] Heflin, J. (2004). Owl web ontology language use cases and requirements.

[Russell and Norvig 2002] Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall series in artificial intelligence. Prentice Hall, 2 edition.