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. Web Semântica

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.

> *"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*

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 de forma mais natural.

2. 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.

2.1. 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.

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:

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

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

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).

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.

2.2. OWL: Web Ontology Language

A *Web Ontology Language*, OWL, foi escolhida pela w3c [1], 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

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

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 numérica não qualificada	$\geq nR$ $\leq nR$ $= nR$	$\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota\}\vert \geq n\}$ $\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota\}\vert \leq n\}$ $\{a \in \Delta^\iota \vert \vert \{b \in \Delta^\iota \vert (a,b) \in R^\iota\}\vert = n\}$	N

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

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.

3. Normalização para o método das conexões

O método das conexões proposto por W. Bibel [Bibel 1982] é um método para prova automática de teoremas descritos em lógica de primeira ordem [Bibel and Hölldobler 1993]. Um dos trabalhos recentes de Freitas et al [Freitas et al. 2010] foi a extenção desse método para lógica de descrição ALC.

O artigo intitulado *A Connection Method for Reasoning with the Description Logic ALC* [Freitas et al. 2010] 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 [Freitas et al. 2010]; 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 do último algoritmo citado e também descreverá a sua implementação.

3.1. Tradução de ontologias ALC para forma normal disjuntiva

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* [Otten 2011] 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 deste artigo descreve o algoritmo que propomos para a normalização para a forma normal positiva.

3.1.1. Algoritmo Proposto

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 precisa estar na forma normal positiva, ou seja, precisa 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. 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: 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 1 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. $\quad O = \{O \cup (pure\_left \sqsubseteq pure\_right)\}$
6. **else**
7. $\quad O = \{O \cup (pure\_left \sqsubseteq N)\}$
8. $\quad O = \{O \cup (N \sqsubseteq pure\_right)\}$
9. **end if**

Algorithm 2: 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*. O método *visit* em sua implementação em Java [2] utiliza o padrão de projetos visitor, que permite criar novas operações sem mudar a definição das estruturas de dados [Gamma et al. 1995].

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 [3] [4].

Algorithm 3: 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: O método *visit* é utilizado para visitar cada nó da estrutura de árvore, que é a representação em memória da ontologia. O símbolo $\bigcup$ será usado na escrita do pseudo-código mas o algoritmo é o mesmo para os constructos $\forall, \bigcup, \bigcap$ e $\exists$, sendo que para $\exists$ e $\forall$ so haverá uma expressão filha.

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

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

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

Algorithm 2 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 3 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

Quatro parâmetros são passadados para os métodos, uma pilha *stack* que é usada para auxiliar a construção de uma nova expressão normalizada. 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 (*conjunction* ou *disjunction*).

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()*.

O Exemplo 1 mostra o caso em que o algoritmo original usa mais memória que o algoritmo proposto. O original é quadrático em relação à alocação memória em quanto que o proposto é linear.

Exemplo 1. *No exemplo dado por Freitas et al [Freitas et al. 2010], 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

Algorithm 4 visit (stack, $\bigcup^{n} e_i$, side, kind)

Require: stack: uma pilha
Require: side: é $left$ ou $right$
Require: $\bigcup^{n} e_i$: constructo em ALC

1. **if** KIND$\bigcup$!= kind **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**

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:

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

4. 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 descrição. Como a w3c [5] 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.

Na proposta inicial deste trabalho estava prevista a implementação do algoritmo de normalização de Freitas et al [Freitas et al. 2010], porém, apesar do algoritmo não

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

incluir novos símbolos à base de conhecimento, não é fácil de ser entendido. A seção 3.1.1 mostra em pseudo-código a implementação que foi feita neste trabalho. O algoritmo foi produzido devido a uma provocação de simplificar o algoritmo de Freitas et al [Freitas et al. 2010]. 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.

5. 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.

[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.

[Bibel 1982] Bibel, W. (1982). *Automated theorem proving*. F. Vieweg.

[Bibel and Hölldobler 1993] Bibel, W. and Hölldobler, S. (1993). *Deduction: automated logic*. Academic Press.

[Freitas et al. 2010] Freitas, F., Schlicht, A., and Stuckenschmidt, H. (2010). A connection method for reasoning with the description logic alc. Technical report, UFPE and University of Mannheim.

[Gamma et al. 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.

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

[Otten 2011] Otten, J. (2011). A non-clausal connection calculus. In Brünnler, K. and Metcalfe, G., editors, *TABLEAUX*, volume 6793 of *Lecture Notes in Computer Science*, pages 226–241. Springer.