UNIVERSIDADE FEDERAL DO CEARÁ

DÉBORA FARIAS FROTA

UMA LÓGICA DE DESCRIÇÃO *DEFAULT*

FORTALEZA, CEARÁ

2011

DÉBORA FARIAS FROTA

UMA LÓGICA DE DESCRIÇÃO *DEFAULT*

Dissertação de Mestrado apresentada à Coordenação do Programa de Pós-graduação em Computação da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de **Mestre em Computação**.

Área de Concentração: Inteligência Artificial

Orientadora: Ana Teresa Martins

Co-Orientador: João Fernando Alcântara

FORTALEZA, CEARÁ

2011

DÉBORA FARIAS FROTA

UMA LÓGICA DE DESCRIÇÃO *DEFAULT*

Dissertação de Mestrado apresentada à Coordenação do Programa de Pós-graduação em Computação da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de **Mestre em Computação**.

Aprovada em: __/__/____

Banca Examinadora

Profa. Dra. Ana Teresa Martins
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. João Fernando Alcântara
Universidade Federal do Ceará - UFC
Co-orientador

Prof. Dr. Marcelino Cavalcante Pequeno
Universidade Federal do Ceará - UFC

Prof. Dr. Edward Hermann Haeusler
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio

AGRADECIMENTOS

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo financiamento de minha pesquisa, meus pais e esposo pelo apoio incondicional e meus orientadores pelos esforços e tempo a mim dedicados.

RESUMO

A formalização do conhecimento e a automatização do raciocínio são assuntos centrais de pesquisa da Inteligência Artificial. A Lógica de Primeira Ordem tem sido tradicionalmente utilizada para tais propósitos. No entanto, ela é mais adequada para lidar com conhecimento completo em circunstâncias ideais. Em situações reais, nas quais o conhecimento é parcial, a Lógica de Primeira Ordem não é suficiente. Lógicas não-monotônicas têm sido propostas para melhor lidar com o raciocínio prático. Uma formalização do raciocínio não-monotônico bem-sucedida é a Lógica *Default* de Reiter que estende a Lógica de Primeira Ordem com regras *default*. Infelizmente, a Lógica *Default* é indecidível. Nesta dissertação, propomos uma Lógica de Descrição *Default* expressiva o suficiente para formalizar o raciocínio prático sobre bases de conhecimento. Ela tem como base monotônica a Lógica de Descrição $\mathcal{ALC}$. Adicionamos algumas restrições à aplicação dos *defaults* a fim de obter propriedades interessantes, tais como a coerência e a eliminação de extensões anômalas. Apresentamos os principais algoritmos usados para construir uma extensão com um passo-a-passo e suas análise de complexidade.
Palavras-Chave: Lógica de Descrição, Lógica Default, Complexidade EXPTIME

ABSTRACT

Knowledge formalization and reasoning automatization are central within Artificial Intelligence. First Order Logic has been traditionally used for such purposes. However, it is better suited to deal with complete knowledge in ideal circumstances. In real situations, in which the knowledge is partial, First Order Logic is not sufficient. Nonmonotonic logics have been proposed to better cope with practical reasoning. A successful formalization of nonmonotonic reasoning is the Reiter's default logic which extends classical logic with default rules. Unfortunately, default logic is undecidable. In this work, we propose a description default logic expressible enough to formalize practical reasoning in knowledge bases. It has as its monotonic basis the $\mathcal{ALC}$ Description Logic. We add some restrictions to the application of defaults in order to obtain nice properties such as coherence and the elimination of anomalous extensions. We present the main algorithms used to build an extension with a step by step complexity analysis.
Keywords: *Description Logic, Default Logic, EXPTIME Complexity*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação e Caracterização do Problema

Com o advento das novas tecnologias, a Internet vem se popularizando cada dia mais e tornando-se ferramenta essencial no cotidiano do ser humano. A *World Wide Web* (também conhecida como *Web* e WWW) é um sistema de documentos em hipermídia que são interligados e executados na Internet. Os documentos podem estar na forma de vídeos, sons, hipertextos e figuras. Para visualizar a informação, pode-se usar um programa de computador chamado navegador para descarregar informações (chamadas "documentos" ou "páginas") de servidores *Web* (ou "*sites*") e mostrá-los na tela do usuário. O usuário pode então seguir as hiperligações na página para outros documentos ou mesmo enviar informações de volta para o servidor para interagir com ele.

A *Web* atualmente possui muitas páginas, cada qual carregada com informações e dados que serão de utilidade para algum usuário no mundo. Resta-nos saber: tal usuário conseguirá encontrar essa informação? Como ele encontrará essa informação? Será que seguindo por hiperligações entre páginas, ele conseguirá encontrar os dados que precisa? Essas e outras perguntas foram base para a construção de uma extensão da *Web* que serve para facilitar que computadores possam atribuir um significado (sentido) aos conteúdos publicados na Internet de modo que seja perceptível tanto pelo humano como pelo computador: a *Web* Semântica.

O objetivo principal da *Web* Semântica não é, apenas treinar as máquinas para que se comportem como pessoas, mas sim desenvolver tecnologias e linguagens que tornem a informação legível para as máquinas. A finalidade passa pelo desenvolvimento de um modelo tecnológico que permita a partilha global de conhecimento assistido por máquinas (W3C 2001). A integração das linguagens ou tecnologias *eXtensible Markup Language* (XML), *Resource Description Framework* (RDF), arquiteturas de meta-dados, ontologias, agentes computacionais, entre outras, favorece o aparecimento de serviços *Web* que garantam a interoperabilidade e cooperação.

Em 2001, (BERNERS-LEE; HENDLER; LASSILA, 2001) definiu o conceito de *Web* Semântica bem como uma possível arquitetura para aplicações sob o mesmo contexto. A arquitetura passou por várias modificações e a sua configuração atual é ilustrada na figura abaixo:

A Lógica é um dos componentes principais da *Web* Semântica. A pergunta que

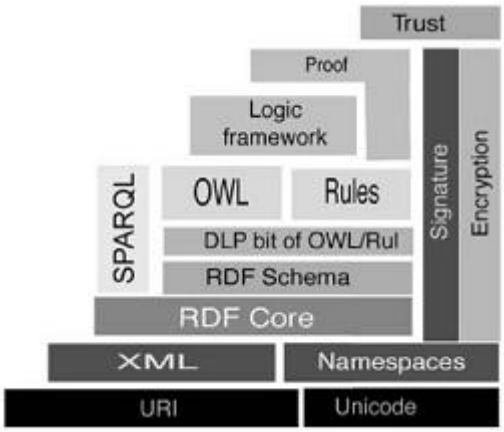

Figura 1: *Web* Semântica - Representação em Camadas

surge agora é qual Lógica será utilizada. Uma opção seria trabalharmos com a Lógica de Primeira Ordem (FOL). Ela possui como grande vantagem ser uma lógica bem madura e robusta. O problema principal é que, para utilizá-la na *Web* Semântica, esta precisa ser decidível. Isto nem sempre acontece.

Outro aspecto importante é que grande parte das informações da *Web* Semântica são expressas utilizando sistemas hierárquicos de simples representação. Logo, não se mostra necessário tanta expressividade quanto a FOL oferece. Sendo assim, partiremos para outra opção. Uma lógica que tem sido bastante utilizada é a Lógica de Descrição (DL) (BAADER et al., 2003). Focaremos nossa pesquisa nessa lógica.

Entretanto, por que não utilizamos Programação em Lógica (PL) ao invés da Lógica de Descrição? Segundo (ANTONIOU; HARMELEN, 2004), é interessante notarmos que DL e PL são ortogonais no sentido de que nenhuma delas é subconjunto da outra. Em outras palavras, existem informações que a DL não consegue expressar e que a PL facilmente expressa. O inverso também acontece.

Como exemplo,

Exemplo 1.1. ***(Definindo a relação Tio em DL e PL)***

Note que é impossível expressarmos a relação Tio(X,Y) em DL. Descrevemo-la a seguir em FOL:

$$\forall X, Y. \exists Z. (IrmãoDe(X, Z) \wedge FilhoDe(Z, Y) \rightarrow Tio(X, Y))$$

Essa relação é facilmente expressa em PL:

$$IrmãoDe(X, Z), FilhoDe(Z, Y) \rightarrow Tio(X, Y)$$

.

Por outro lado, PL não consegue (no caso geral) expressar, por exemplo:

- negação/complemento de classes
- informação disjuntiva
- quantificação existencial
- quantificação universal explicitamente

Concluimos, assim, que a escolha entre PL e DL é feita de acordo com o que se quer representar.

Outra camada importante da *Web* Semântica é a camada de Regras. Esta camada mostra-se de extrema importância ao inferirmos novas informações a partir do conhecimento já existente. A Lógica tem muito a contribuir para o desenvolvimento da *Web* Semântica nesse aspecto, uma vez que já foram desenvolvidos diversos trabalhos tanto na área de raciocinadores que utilizam Programação em Lógica como pela área voltada para o raciocínio não-monotônico (GINSBERG, 1987a).

Por que precisamos introduzir o raciocínio não-monotônico em nossa pesquisa? As lógicas monotônicas, por exemplo a FOL, são perfeitas para modelar situação ideais nas quais o conhecimento obtido é preciso e completo o suficiente para efetuarmos inferências cujas conclusões tem o *status* de certeza. Em situações reais (no nosso caso, as expressas na *Web*), nas quais o conhecimento é incompleto e impreciso, a FOL que é monotônica, não é suficiente. As lógicas não-monotônicas são perfeitas para representação e raciocínio em bases de conhecimento parciais.

As Lógicas Não-monotônicas são sistemas lógicos nos quais a propriedade da monotonicidade falha. Essa propriedade garante que, se a conclusão é inferida a partir de algumas premissas, nenhuma premissa adicional invalidará a conclusão. Entretanto, em situações reais, é comum concluirmos informações a partir de nossas experiências em fatos que achamos serem costumeiros, os quais podem ser refutáveis em face de novas informações. São diversas as áreas de aplicação do raciocínio não-monotônico, dentre elas (MARTINS, 1997), (BASSILIADES; ANTONIOU; VLAHAVAS, 2006):

Raciocínio sobre ações representação de situações onde uma ação pode falhar. Como exemplo, quando damos a partida em um carro, ele funcionará, a menos que não tenha combustível, a bateria descarregou ou tenha algum outro problema mecânico;

Regras com exceção Essa área trabalha buscando soluções para conflitos entre regras. Como exemplo, citaremos novamente o caso dos pinguins não voarem ser uma exceção ao caso geral dos pássaros voarem;

Herança de propriedades Valores são assumidos para o caso geral. Quando descemos no nível de especialização de uma hierarquia, valores assumidos podem ser sobrepostos por outros mais específicos;

Informação negativa expressar informações negativas sempre foi assunto de muitos estudos devido ao seu alto custo de expressividade. Diversas formas foram desenvolvidas buscando aproximações. Citamos como um exemplo destes trabalhos a *negação por falha*;

Diagnóstico Essa área possui como objetivo principal achar um conjunto minimal de possíveis causas que podem explicar o mal funcionamento do objeto em questão;

Definições indutivas definições recursivas podem ser vistas como generalizações de definições caso a caso. Dentre as definições caso a caso sempre temos uma chamada cláusula de fechamento que permite, dentre outras coisas, provas de resultados negativos;

Fusão de conhecimento São vários os problemas encontrados quando visamos a fusão de trabalhos de autores diferentes. Contradição é um dos problemas que ocorrem mais frequentemente. O uso de regras de prioridade pode ser uma possível solução para o caso.

Dentre as formalizações do raciocínio não-monotônico, citamos as principais Lógicas: *Defeasible*(BASSILIADES; ANTONIOU; VLAHAVAS, 2006), Circunscrição (MCCARTHY, 1980), Auto-epistêmica (GINSBERG, 1987a) e Default de Reiter (R, 1987). Decidimos utilizar a lógica *Default* de Reiter. Essa escolha foi feita devido ao fato de termos como objetivo expressar as informações incompletas como regras em que descrevemos os casos gerais ou mais frequentes das informações juntamente com suas exceções.

1.2 Objetivos da Dissertação

O objetivo de nossa pesquisa é investigar, apresentar e desenvolver um sistema de representação de conhecimento não-monotônico que seja aplicável a *Web* e que suas inferências sejam decidíveis.

Ao utilizarmos a Lógica *Default*, nos deparamos com uma lógica indecidível. FOL, que é a parte monotônica da Lógica *Default* também não é decidível, sendo somente semi-decidível. Isso ocorre pelo fato de que o conjunto de seus teoremas é recursivamente enumerável. Já para a Lógica *Default*, seu conjunto de teoremas não é nem recursivamente enumerável.

Uma possível solução para obtermos uma lógica não-monotônica decidível é restringirmos suas partes monotônica e não-monotônica.

Restringimos a parte monotônica ao utilizarmos agora a DL $\mathcal{ALC}$, abordada no capítulo 2. $\mathcal{ALC}$ pode ser descrita como um fragmento de FOL com somente relações unárias e binárias acrescida da noção de complemento de conceitos arbitrários. Como $\mathcal{ALC}$ pode ser traduzida para L_2, o fragmento relacional de FOL acrescida de constantes que utilizam somente duas variáveis em suas fórmulas, $\mathcal{ALC}$ tem a propriedade de modelo

finito e, assim, o problema de satisfatibilidade é decidível em $NTIME(2^{O(n)})$, como limite superior (BÖRGER; GRÄDEL; GUREVICH, 1997).

Uma vez que a parte monotônica agora é decidível, devemos limitar a Lógica *Default* de forma que ela preserve a decidibilidade para o raciocínio não-monotônico. Isso é garantido pela imposição de uma ordem sobre a aplicação das regras *default*, apresentadas no capítulo 3, que respeitam o princípio das "exceções primeiro" (PEQUENO, 1994), levando também em conta o conhecimento representado pela lógica $\mathcal{ALC}$. Esse princípio é a chave para resolver o problema de extensões anômalas como o introduzido pelo "Yale Shooting Problem" (GINSBERG, 1987b) que será apresentado no capítulo 3.

1.3 Organização da Dissertação

No capítulo 2, faremos um estudo sobre as Lógicas de Descrição (DL), uma família de lógicas que servirão de base para a camada de regras previamente mencionada. Primeiramente, será abordada a lógica $\mathcal{AL}$, por ser a DL mais simples. Em seguida, o capítulo abordará o sistema de representação de conhecimento, a sintaxe e semântica das DL. Por fim, será apresentado dois algoritmos de *tableaux* que nos permitem raciocinar sobre as informações descritas em $\mathcal{ALC}$, a DL mais simples acrescida da noção de complemento de conceitos arbitrários. O primeiro algoritmo, apresentado em (BAADER et al., 2003), é o mais simples e somente raciocina sobre axiomas de igualdade. Ele possui complexidade PSPACE-complete. Já o segundo, foi desenvolvido em (DONINI; MASSACCI, 2000). Ele permite axiomas gerais e possui complexidade exponencial simples, ou seja, EXPTIME.

O capítulo 3 abordará o estudo da sintaxe e semântica da Lógica *Default* de Reiter, bem como a noção de extensões possíveis para essa lógica. Alguns problemas podem ocorrer às extensões existentes e estes também são abordados. Em seguida, é apresentado o algoritmo de construção de extensões para Lógica *Default* proposto por Etherington e seus problemas (ETHERINGTON, 1988).

Já no capítulo 4, apresentamos uma proposta de Lógica de Descrição *Default* (DDL). A DDL é composta a partir da Lógica de Descrição acrescida de regras *default*. Ainda, desenvolveremos algoritmos juntamente com o cálculo de suas complexidades computacionais que nos permitirão construir de forma decidível uma extensão para a DDL e que evitará todos os problemas encontrados no algoritmo de construção de extensões apresentado no capítulo 3.

As conclusões acerca do trabalho realizado, uma comparação entre nossa pesquisa e os trabalhos relacionados e os trabalhos futuros podem ser encontrados no capítulo 5.

2 LÓGICAS DE DESCRIÇÃO

Neste capítulo, apresentaremos uma família de lógicas conhecida como Lógica de Descrição (DL). Para simplificar, usaremos indistintamente o termo "Lógica de Descrição" para designar tanto uma família de Lógicas de Descrição como uma Lógica de Descrição particular. Iniciaremos introduzindo a Lógica de Descrição mais simples, a lógica $\mathcal{AL}$, e depois apresentaremos algumas de suas extensões. Abordaremos detalhes sobre o sistema de representação de conhecimento, a sintaxe e semântica destas lógicas. Finalizaremos este capítulo apresentando dois algoritmos de *tableaux* que nos permitem raciocinar sobre as informações descritas em $\mathcal{ALC}$, ou seja, um exemplo de extensão de $\mathcal{AL}$. O primeiro, mais simples, somente permite um tipo restrito de axiomas. O segundo já aborda o caso mais geral.

Utilizamos (BAADER et al., 2003) como fonte de referência principal para este capítulo. Definições, lemas, teoremas e outras informações aqui apresentadas e não referenciadas supõem-se retiradas dessa fonte. As informações obtidas de outras fontes serão explicitamente referenciadas.

A DL surgiu a partir das Redes Semânticas (SOWA, 1992). As Redes Semânticas são estruturas formadas a partir dos elementos:(1) os nós, que representam conceitos, conjuntos ou classes de objetos individuais, e (2) arcos, que representam, na maioria das vezes, uma relação entre os nós cujo sentido é de "está contido".

A DL foi desenvolvida para solucionar algumas desvantagens das Redes Semânticas, principalmente a pouca formalização de sua linguagem. Ela possui como característica principal a preservação das vantagens das Redes Semânticas, tais como simplicidade e eficiência dos mecanismos de inferências, facilidade no raciocínio sobre herança e transparência das informações apresentadas. Outras características de bastante destaque são que as Lógicas de Descrição são decidíveis.

Na seção a seguir, iniciaremos nosso estudo sobre a DL.

2.1 Tbox e Abox

Assim como todo sistema de representação de conhecimento, temos uma base de conhecimento (KB) formado pelas informações já obtidas e armazenadas sobre o domínio em questão. No caso da DL, seu KB é composto pelo Tbox *(Terminological Box)* e Abox

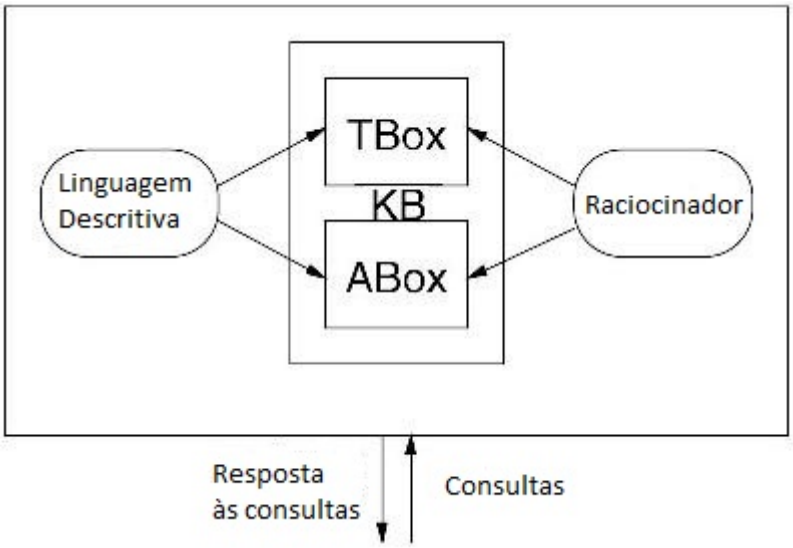

Figura 2: Sistema de Conhecimento Lógica de Descrição

(Assertional Box).

O Tbox introduz a terminologia, ou seja, define os conceitos e papéis assim como estabelece relações de inclusão entre eles. Os conceitos e papéis, na linguagem da Lógica de Primeira Ordem (FOL), são denominados de relações unárias e binárias, respectivamente. No caso do Abox, ele contém asserções sobre os indivíduos. Apresentaremos a seguir alguns exemplos de Tbox e Abox:

Exemplo 2.1. ***(Tbox)***

$$Mae \equiv Pessoa \sqcap \exists temFilho.Pessoa \tag{2.1}$$

$$Mulher \sqsubseteq Pessoa \tag{2.2}$$

Em (2.1)*, temos um Tbox que descreve a relação familiar Mãe da seguinte forma: Mãe é uma pessoa que tem filho o qual também é uma pessoa. Utilizamos os conceitos Mãe e Pessoa e o papel temFilho.*

Note que, através da especificação de características determinantes definimos novos conceitos existentes na nossa realidade. Infelizmente, nem sempre é possível definir conceitos de forma completa através da DL uma vez que nem sempre conseguimos descrever todas as características de certos elementos da realidade. Ao escolhermos as características principais, conseguimos uma descrição bem próxima da realidade.

O segundo Tbox (2.2) *descreve o fato de ser Mulher implica ser uma Pessoa. São utilizados os conceitos Mulher e Pessoa.*

Exemplo 2.2. ***(Abox)***

$$Pai(PEDRO) \tag{2.3}$$

$$temFilho(PEDRO, HENRIQUE) \tag{2.4}$$

Em (2.3)*, temos o Abox que transmite a informação que PEDRO é Pai. Já em* (2.4)*, o Abox descreve que PEDRO tem um filho que se chama HENRIQUE.*

Para completar o sistema, a DL possui tarefas de raciocínio ou inferências. Essas tarefas são usadas para inferir o conhecimento que se encontra implícito na KB. Um exemplo dessas tarefas é a subsunção. Ela nos leva a uma hierarquização dos conceitos introduzidos pelos Tbox e Abox.

Como exemplo, analisemos o Tbox $Mulher \sqsubseteq Pessoa$. Dele podemos inferir que o conceito $Mulher$ é mais especializado que o conceito $Pessoa$. Analogamente, inferimos que o conceito $Pessoa$ é mais generalizado que o conceito $Mulher$.

A figura 2 resume as estruturas do sistema de representação de conhecimento de DL.

Na próxima seção, apresentaremos algumas definições importantes da linguagem padrão básica da Lógica de Descrição referida pela literatura como *Attributive Language* ($\mathcal{AL}$) (BAADER et al., 2003).

2.2 Definição da Sintaxe e da Semântica de $\mathcal{AL}$

A Lógica de Descrição é, na verdade, uma família de Lógicas de Descrição de características variadas. Iremos abordar neste trabalho a lógica que é básica para todas as outras da família: a $\mathcal{AL}$.

Apresentaremos nas próximas seções definições a respeito de sua sintaxe e semântica.

2.2.1 Sintaxe

A definição seguinte define a noção de conceitos de maneira precisa:

Definição 2.3. ***($\mathcal{AL}$-conceito)***

A linguagem dos $\mathcal{AL}$-conceitos (ou somente conceitos) é o menor conjunto definido indutivamente como segue:

- $\top$ *(Conceito Universal) é um $\mathcal{AL}$-conceito;*
- $\bot$ *(Conceito Bottom) é um $\mathcal{AL}$-conceito;*
- *se C é um conceito atômico (predicado unário), C é um $\mathcal{AL}$-conceito;*
- *se C é um $\mathcal{AL}$-conceito atômico, $\neg C$ é um $\mathcal{AL}$-conceito*
- *se C e D são $\mathcal{AL}$-conceitos e R é um papel (predicado binário), então os seguintes elementos também são $\mathcal{AL}$-conceitos:*

– $C \sqcap D$ *(Interseção)*

– $\exists R.\top$ *(Quantificação Existencial Limitada)*

– $\forall R.C$ *(Restrição de Valor)*

Observações:

- Utilizaremos as letras C, D para nos referir a conceitos e R, S para papel;
- O conceito universal $\top$ só pode ser usado com o quantificador existencial limitado ($\exists R.\top$);
- A negação só pode ser aplicada a conceitos atômicos.

A linguagem $\mathcal{AL}$-conceito se mostra como base para definirmos os axiomas terminológicos.

Definição 2.4. ***(Axiomas Terminológicos)***

A seguir, definiremos os axiomas terminológicos. Existem dois tipos:

- *Axiomas de inclusão: $C \sqsubseteq D$, com C e D sendo $\mathcal{AL}$-conceitos;*
- *Axiomas de igualdade: $C \equiv D$, com C e D sendo $\mathcal{AL}$-conceitos.*

Após estas definições, podemos finalmente definir os Tbox:

Definição 2.5. ***(Tbox)***

Um Tbox é um conjunto finito de axiomas terminológicos.

Continuaremos as definições agora para os Abox. Estes podem ser de dois tipos:

Definição 2.6. ***(Asserção de Conceitos)***

"a" é chamado de instância de C, onde C é um conceito e "a" um indivíduo. Representa-se esse fato da seguinte forma: $C(a)$ ou $a : C$.

Definição 2.7. ***(Asserção de papéis)***

$R(a, b)$ é chamado de instância de R, onde R é um papel e a e b são indivíduos. Representa-se este fato da seguinte forma: $R(a, b)$ ou $\langle a, b\rangle : R$.

Após essas definições, podemos definir os Abox:

Definição 2.8. ***(Abox)***

Um Abox é um conjunto finito de asserções de conceitos e papéis.

A definição a ser apresentada a seguir é sobre a base de conhecimento (KB) de $\mathcal{AL}$. Conforme mencionado anteriormente (veja figura 2), temos que sua KB é formado pelos Tbox e Abox. Assim,

Definição 2.9. ***($\mathcal{ALKB}$)***

Uma base de conhecimento de $\mathcal{AL}$ ($\mathcal{ALKB}$) é definida da seguinte forma:

$\mathcal{ALKB} = \langle \mathcal{T}, \mathcal{A} \rangle$, *onde* $\mathcal{T}$ *e* $\mathcal{A}$ *são os Tbox e Abox, respectivamente.*

2.2.2 Sêmantica

Passando agora para a parte semântica da linguagem $\mathcal{AL}$, começamos com a definição da semântica da linguagem de $\mathcal{AL}$-conceito:

Definição 2.10. ***(Semântica de $\mathcal{AL}$-conceitos)***

Dada uma interpretação $\mathcal{I} = \langle \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$ *onde* $\Delta^{\mathcal{I}}$ *representa o domínio da interpretação e* $.^{\mathcal{I}}$, *uma função que mapeia cada conceito* C *ao conjunto* $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ *e para todo papel* R, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, *temos:*

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \varnothing^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \forall b(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.\top)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \exists b(a,b) \in R^{\mathcal{I}}\}
\end{aligned}$$

A noção de satisfação para os axiomas terminológicos e para os Tbox segue abaixo:

Definição 2.11. ***(Satisfação dos axiomas terminológicos)***

Os axiomas terminológicos são satisfeitos por uma interpretação $\mathcal{I}$, *ou seja,*

- $\mathcal{I} \models C \sqsubseteq D$ *se e somente se* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;
- $\mathcal{I} \models C \equiv D$ *se e somente se* $C^{\mathcal{I}} = D^{\mathcal{I}}$;

Definição 2.12. ***(Satisfação para os Tbox)***

Um Tbox é satisfeito por uma interpretação $\mathcal{I}$, *ou seja,* $\mathcal{I} \models \mathcal{T}$ *se e somente se para todo* $C \sqsubseteq D$ *e* $C \equiv D$ *em* $\mathcal{T}$, $\mathcal{I} \models C \sqsubseteq D$ *e* $\mathcal{I} \models C \equiv D$.

Quando uma interpretação $\mathcal{I}$ *satisfaz um Tbox* $\mathcal{T}$, $\mathcal{I}$ *é chamada de modelo de* $\mathcal{T}$.

Na seção 2.5, veremos as definições das tarefas de raciocínio ou inferência principais de DL envolvendo Tbox. Essas definições também envolvem aspectos semânticos.

Assim, finalizamos as definições para os Tbox. Agora, abordaremos as definições semânticas para os Abox:

As definições seguintes são referentes à noção de satisfação dos Abox:

Definição 2.13. ***(Satisfação de uma asserção de conceito)***

Uma asserção de conceito $C(a)$ é satisfeita por uma interpretação $\mathcal{I}$, ou seja, $\mathcal{I} \vDash C(a)$ se e somente se $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

Definição 2.14. ***(Satisfação de uma asserção de papel)***

Uma asserção de papel $R(a,b)$ é satisfeita por uma interpretação $\mathcal{I}$, ou seja, $\mathcal{I} \vDash R(a,b)$ se e somente se $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$.

Definição 2.15. ***(Satisfação de um Abox)***

Um Abox é satisfeito por uma interpretação $\mathcal{I}$, ou seja, $\mathcal{I} \vDash \mathcal{A}$ se e somente se para toda asserção da forma $\alpha = C(a)$ ou $\alpha = R(a,b)$ em $\mathcal{A}$, $\mathcal{I} \vDash \alpha$.

A definição a seguir refere-se à noção de satisfação para a base de conhecimento $\mathcal{ALC}KB = \langle \mathcal{T}, \mathcal{A} \rangle$ a qual só ocorre quando $\mathcal{ALC}KB$ satisfaz tanto o Tbox como o Abox. Assim,

Definição 2.16. ***(Satisfação de um $\mathcal{ALC}KB$)***

Uma base de conhecimento $\mathcal{ALC}KB = \langle \mathcal{T}, \mathcal{A} \rangle$ é satisfeita por uma interpretação $\mathcal{I}$, ou seja, $\mathcal{I} \vDash \mathcal{ALC}KB$ se e somente se $\mathcal{I} \vDash \mathcal{T}$ e $\mathcal{I} \vDash \mathcal{A}$

2.3 Extensões da Linguagem $\mathcal{AL}$

A linguagem $\mathcal{AL}$ é uma linguagem utilizada por outras Lógicas de Descrição como linguagem base na qual são acrescentadas novas funcionalidades. Na tabela 1, temos alguns exemplos dessas linguagens.

Pela tabela, podemos formar diversas linguagens de acordo com a funcionalidade que desejamos adicionar.

Denominamos essa nova linguagem de $\mathcal{AL}x_1x_2\ldots x_n(y_1,\ldots,y_m)$, onde $x_1,\ldots,x_n$ são os símbolos da tabela acima que representam as novas funcionalidades aplicadas aos conceitos acrescidas e $y_1,\ldots,y_m$ os símbolos que representam as novas funcionalidades aplicadas aos papéis.

Alguns exemplos de linguagens DL estendidas mais conhecidas são: a $\mathcal{ALC}$,$\mathcal{ALC}(\cup)$, $\mathcal{ALCN}$ e $\mathcal{ALCQIO}$ ou $\mathcal{SHOIQ}$, como é mais conhecida. Outra DL de bastante destaque que não segue a regra de escrita citada acima é a $\mathcal{SHOIN}$. Esta DL é suportada pelo editor de ontologias e raciocinador *Protegé*(PROTéGé,) atualmente bastante utilizado pela área de *Web* Semântica (ANTONIOU; HARMELEN, 2004).

Adicionando a DL	Símbolo	Sintaxe	Semântica
Negação de conceitos	$\mathcal{C}$	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
União de Conceitos	$\mathcal{U}$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Quantificação Existencial Geral	$\mathcal{E}$	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
Restrição Numérica Superior	$\mathcal{N}$	$\geq n.R$	$\{a \in \Delta^{\mathcal{I}} \mid \left\|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right\| \geq n\}$
Restrição Numérica Inferior	$\mathcal{N}$	$\leq n.R$	$\{a \in \Delta^{\mathcal{I}} \mid \left\|\{b \mid (a,b) \in R^{\mathcal{I}}\}\right\| \leq n\}$
Restrição Numérica Qualificada Superior	$\mathcal{Q}$	$\geq nR.C$	$\{a \in \Delta^{\mathcal{I}} \mid \left\|\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\right\| \geq n\}$
Restrição Numérica Qualificada Inferior	$\mathcal{Q}$	$\leq nR.C$	$\{a \in \Delta^{\mathcal{I}} \mid \left\|\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\right\| \leq n\}$
Funções	$\mathcal{F}$	$\leq 1.R$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b, c\ (a,b) \in R^{\mathcal{I}} \wedge (a,c) \in R^{\mathcal{I}} \rightarrow b = c\}$
Nominais	$\mathcal{O}$	$\{a_1, \ldots, a_n\}$	$\{a_1, \ldots, a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$
Interseção de Papéis	$(\cap)$	$R \cap S$	$R^{\mathcal{I}} \cap S^{\mathcal{I}}$
União de Papéis	$(\cup)$	$R \cup S$	$R^{\mathcal{I}} \cup S^{\mathcal{I}}$
Complemento de Papéis	$(\neg)$	$\neg R$	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} / R^{\mathcal{I}}$
Inverso de Papéis	$\mathcal{I}$	R^{-}	$\{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}$
Composição de Papéis	$(\circ)$	$R \circ S$	$\{(a,c) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge (b,c) \in S^{\mathcal{I}}\}$
Transitividade de Papéis	$\mathcal{S}$	R^{+}	$\cup_{i \geq 1} (R^{\mathcal{I}})^i$
Hierarquia de Papéis	$\mathcal{H}$	$R \subseteq S$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b (a,b) \in R^{\mathcal{I}} \rightarrow (a,b) \in S^{\mathcal{I}}\}$

Tabela 1: Exemplos de Lógicas de Descrição e suas novas funcionalidades

2.4 $\mathcal{AL}$ como um fragmento de FOL

Para fins comparativos, apresentaremos a tabela 2 de correspondência entre as fórmulas escritas em DL (e algumas de suas extensões) e as escritas em FOL.

O foco principal da tabela 2 é nos mostrar que tudo expresso em DL pode ser expresso em FOL.

A DL vem sendo amplamente utilizada pelos profissionais das áreas de banco de dados, modelagem e engenharia de *software* para descrição de ontologias da *web*. Diante deste grande uso, há uma grande necessidade de investigação sobre as propriedades deste fragmento particular da FOL. Por exemplo, o problema de saber se uma fórmula α é um teorema em FOL é semi-decidível enquanto que o mesmo problema em DL é decidivel.

2.5 Tarefas de Raciocínio ou Inferências

Seguindo nosso estudo sobre o sistema de representação do conhecimento das Lógicas de Descrição, abordaremos agora as tarefas de raciocínio, também chamadas de inferências. Essas podem ser efetuadas por qualquer sistema dedutivo de FOL (restringindo-nos ao fragmento da linguagem DL em consideração). Essas inferências nos permitem certo tipo de raciocínio sobre sua base de conhecimento, ou seja, seus Tbox e Abox. Esse raciocínio é feito sobre o conhecimento explícito em seus Tbox e Abox. O conhecimento que se encontra implícito pode ser tornado também explícito através das tarefas de raciocínio quando necessário.

DL	**FOL**
C	$C(x)$
R	$R(x,y)$
$a:C$	$C(a)$
$\langle a,b\rangle : R$	$R(a,b)$
$\neg C$	$\neg C(x)$
$C_1 \sqcap C_2$	$C_1(x) \wedge C_2(x)$
$\exists R.C$	$\exists y(R(x,y) \wedge C(y))$
$\forall R.C$	$\forall y(R(x,y) \rightarrow C(y))$
$C_1 \sqcup C_2$	$C_1(x) \vee C_2(x)$
$\geq nR$	$\exists y_1,\ldots,y_n \bigwedge_{1\leq i\leq n}(R(x,y_i)) \wedge \bigwedge_{1\leq i<n, i<j\leq n} y_i \neq y_j$
$\leq nR$	$\forall y_1,\ldots,y_{n+1} \bigwedge_{1\leq i\leq n+1}(R(x,y_i)) \rightarrow (\bigvee_{1\leq i<n+1, i<j\leq n+1} y_i = y_j)$
$\geq nR.C$	$\exists y_1,\ldots,y_n \bigwedge_{1\leq i\leq n}(R(x,y_i) \wedge C(y_i)) \wedge \bigwedge_{1\leq i<n, i<j\leq n} y_i \neq y_j$
$\leq nR.C$	$\forall y_1,\ldots,y_{n+1} \bigwedge_{1\leq i\leq n+1}(R(x,y_i) \wedge C(y_i)) \rightarrow (\bigvee_{1\leq i<n+1, i<j\leq n+1} y_i = y_j)$
$\leq 1R$	$\forall x,y(R(x,y) \wedge R(x,z)) \rightarrow y = z$
$\{a_1,\cdots,a_n\}$	$x = a_1 \vee \cdots \vee x = a_n$
$R_1 \cap R_2$	$R_1(x) \wedge R_2(x)$
$R_1 \cup R_2$	$R_1(x) \vee R_2(x)$
$\neg R$	$\neg R(x,y)$
R^-	$\forall x,y R(y,x)$
$R \circ S$	$\exists y(R(x,y) \wedge S(y,z) \rightarrow T(x,z))$
R^+	$\forall x,y,z(R(x,y) \wedge R(y,z)) \rightarrow R(x,z)$
$R \sqsubseteq S$	$\forall x.R(x) \rightarrow S(x)$
$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
$C \equiv D$	$\forall x.((C(x) \rightarrow D(x)) \wedge (D(x) \rightarrow C(x)))$

Tabela 2: Correspondência entre algumas fórmulas de DL e FOL

Antes de definirmos formalmente tais tarefas, vejamos um exemplo que nos mostrará a obtenção de conhecimento ainda implicito na base de conhecimento em questão:

Exemplo 2.17. ***(Tbox e Abox de Relações Familiares)***

Seja $\mathcal{ALK}B = \langle \mathcal{T}, \mathcal{A}\rangle$*, onde* $\mathcal{T}$ *são axiomas* (2.5)-(2.15) *e* $\mathcal{A}$ *as asserções* (2.16)-

(2.20).

$$Mulher \equiv Pessoa \sqcap SexoFeminino \quad (2.5)$$
$$Homem \equiv Pessoa \sqcap SexoMasculino \quad (2.6)$$
$$Mae \equiv Mulher \sqcap \exists temFilho.Pessoa \quad (2.7)$$
$$Mamae \equiv Mae \quad (2.8)$$
$$Pai \equiv Homem \sqcap \exists temFilho.Pessoa \quad (2.9)$$
$$Papai \equiv Pai \quad (2.10)$$
$$Pais \equiv Pai \sqcup Mae \quad (2.11)$$
$$Avo \equiv Mae \sqcap \exists temFilho.Pais \quad (2.12)$$
$$MaeComMuitosFilhos \equiv Mae \sqcap \geq 3 temFilho \quad (2.13)$$
$$MaeSemFilhas \equiv Mae \sqcap \forall temFilho.\neg Mulher \quad (2.14)$$
$$Esposa \equiv Mulher \sqcap \exists temMarido.Homem \quad (2.15)$$

$$MaeSemFilhas(MARIA) \quad (2.16)$$
$$Pai(PEDRO) \quad (2.17)$$
$$temFilho(MARIA, PEDRO) \quad (2.18)$$
$$temFilho(MARIA, PAULO) \quad (2.19)$$
$$temFilho(PEDRO, HENRIQUE) \quad (2.20)$$

A partir desse Tbox (2.5)-(2.15) e Abox (2.16) - (2.20), conseguimos inferir que MARIA é uma avó: Analisando (2.16) e (2.14) deduzimos que MARIA é mãe. Aplicando (2.17) e (2.11) temos que PEDRO pertence a classe dos pais. Como MARIA tem filho PEDRO (2.18) e PEDRO tem filho HENRIQUE (2.20), deduzimos finalmente que MARIA é avó (2.12).

Nas seções seguintes, apresentaremos as principais tarefas de raciocínio. Dividiremos a apresentação em duas partes. Primeiramente, apresentaremos as principais tarefas de raciocínio para conceitos e Tbox. Logo após, mostraremos as principais tarefas de raciocínio para Abox.

2.5.1 Tarefas de Raciocínio para Tbox

Existem diversos tipos de raciocínios ou inferências referentes aos conceitos e Tbox. Focaremos nossa atenção nas quatro principais.

Definição 2.18. ***(Satisfação de um conceito com respeito a um Tbox)***

Um conceito C é satisfatível com respeito ao Tbox $\mathcal{T}$ se existe um modelo $\mathcal{I}$ de $\mathcal{T}$ tal que $C^{\mathcal{I}} \neq \emptyset$.

Definição 2.19. ***(Subsunção de um conceito C por outro com respeito a um***

Tbox)

Dizemos que um conceito C é subsumido pelo conceito D com respeito ao Tbox $\mathcal{T}$ (escrito como $C \sqsubseteq_{\mathcal{T}} D$ ou $\mathcal{T} \vDash C \sqsubseteq D$) se $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ para todo modelo $\mathcal{I}$ de $\mathcal{T}$ [1].

Definição 2.20. ***(Equivalência de Conceitos)***

Dois Conceitos são equivalentes com respeito ao Tbox $\mathcal{T}$ (escrito como $C \equiv_{\mathcal{T}} D$ ou $\mathcal{T} \vDash C \equiv D$) se eles possuem os mesmos modelos.

Definição 2.21. ***(Equivalência de Tbox)***

Dois Tbox são equivalentes (escrito como $\mathcal{T}_1 \equiv \mathcal{T}_2$) se eles possuem os mesmos modelos.

Definição 2.22. ***(Disjunção de dois conceitos com respeito a um Tbox)***

Dois conceitos C e D são disjuntos com respeito a um Tbox $\mathcal{T}$ se $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ for vazio para todo modelo $\mathcal{I}$ de $\mathcal{T}$.

Definição 2.23. ***(Disjunção de Tbox)***

Dois Tbox são disjuntos se a interseção de seus modelos for vazia.

Exemplo 2.24. *O que podemos inferir do exemplo 2.17 que traz o Tbox de algumas relações familiares?*

Primeiramente, podemos dizer que o conceito $Avô \equiv Pai \sqcap \exists temFilhos.Pais$ é satisfatível com respeito ao Tbox em questão. Falando agora da subsunção, inferimos que Pessoa subsume Mulher e Homem subsume Pai. Como exemplos de disjunções, temos que Mulher é disjunto de Homem e que Pai é disjunto de Mãe. Finalmente, temos como exemplos de equivalências, os conceitos Pai e Papai.

Como estas tarefas de raciocínio se relacionam? O teorema a seguir nos responde:

Teorema 2.25. ***(Redução à Subsunção)***

Para conceitos C e D, temos:

- *C é insatisfatível se e somente se C é subsumido por $\bot$;*
- *C e D são equivalentes se e somente se C é subsumido por D e D é subsumido por C;*
- *C e D são disjuntos se e somente se $C \sqcap D$ e subsumido por $\bot$.*

Essas afirmações também valem com respeito a Tbox.

[1] Na literatura sobre lógica clássica ((ETHERINGTON, 1988), (DALEN, 1994), (ENDERTON, 2001)), é usual dizer que $C \sqsubseteq D$ é consequência lógica de $\mathcal{T}$ quando se usa a notação $\mathcal{T} \vDash C \subseteq D$. Similarmente para $\mathcal{T} \vDash C \equiv D$.

Em outras palavras, todos os outros três tipos de inferência podem ser reduzidos para a subsunção. Concluímos que a subsunção é um dos mecanismos básicos de sistemas baseados em DL. Com isso, encontramos um limite superior da complexidade dos algoritmos das inferências: a complexidade do algoritmo da subsunção que é PSpace-complete.

É possível, entretanto, reduzir a subsunção à insatisfatibilidade. Nesse sentido, a (in)satisfatibilidade é também uma questão central pois é possível reduzir as demais tarefas de inferência a ela, bastando que a linguagem da DL em consideração suporte conjunção e negação de conceitos arbitrários.

Teorema 2.26. ***(Redução a Insatisfatibilidade)***

Para conceitos C e D, temos:

- *C é subsumido por D se e somente se $C \sqcap \neg D$ é insatisfatível;*
- *C e D são equivalentes se e somente se ambos $(C \sqcap \neg D)$ e $(\neg C \sqcap D)$ são insatisfatíveis;*
- *C e D são disjuntos se e somente se $C \sqcap D$ é insatisfatível.*

Estas afirmações também valem com respeito a Tbox.

A partir de agora, vamos assumir que os Tbox são acíclicos e formados apenas de axiomas de igualdade. Incluiremos os Tbox cíclicos e com o axioma de inclusão na seção 2.6.1. Apresentaremos agora uma definição de grande importância para os Tbox que nos ajudará a implementar as tarefas de raciocínio citadas acima: expansão de conceitos.

Definição 2.27. ***(Expansão de Conceitos)***

Seja C um conceito de um Tbox acíclico $\mathcal{T}$. Obtemos o conceito C', ou seja, o conceito expandido de C com respeito a $\mathcal{T}$, através da substituição de todas as ocorrências do símbolo nominal[2] A em C pelo conceito D, onde $A \equiv D$ é a definição de A em $\mathcal{T}'$, ou seja, a expansão de $\mathcal{T}$.

Note que na definição 2.27 utilizamos um tipo de Tbox específico, ou seja, os acíclicos. Precisamos inicialmente definir algumas relações antes de proseguirmos definindo Tbox acíclicos.

Definição 2.28. ***(Relação "usa diretamente")***

Sejam A e B conceitos atômicos que ocorrem no Tbox $\mathcal{T}$. Dizemos que A "usa diretamente" B em $\mathcal{T}$ se B ocorre no lado direito da definição de A.

Definição 2.29. ***(Relação "usa")***

Sejam A e B conceitos atômicos que ocorrem no Tbox $\mathcal{T}$. Definimos também a relação "usa" como o fecho transitivo da relação "usa diretamente".

[2]Símbolos nominais são os simbolos que ocorrem no lado esquerdo dos axiomas terminológicos. Como exemplo, o símbolo nominal do Tbox $A \equiv C \sqcap D$ é A.

Segue a definição de Tbox acíclico:

Definição 2.30. ***(Tbox cíclico e acíclico)***

Seja $\mathcal{T}$ um Tbox . $\mathcal{T}$ é cíclico, ou seja, contém um ciclo se e somente se existe um conceito atômico que "usa" ele mesmo. Caso contrário, chamamos $\mathcal{T}$ de acíclico.

Exemplo 2.31. ***(Tbox cíclico)***

$$Humano \equiv Animal \sqcap \forall temParente.Humano$$

Neste Tbox, definimos como humano aqueles que são animais e todos seus parentes são também humanos. O ciclo ocorre na utilização do conceito Humano tanto do lado esquerdo como no direito do axioma.

Agora, podemos passar a definição de expansão de Tbox acíclicos,

Definição 2.32. ***(Expansão de Tbox acíclicos)***

A expansão de Tbox $\mathcal{T}'$ é obtida através da substituição de todos os conceitos do Tbox acíclico $\mathcal{T}$ por seus conceitos expandidos.

Exemplificando as novas definições,

Exemplo 2.33. ***(Expansão de Conceitos)***

Utilizando o Tbox a seguir (2.21)-(2.22), iremos expandir o conceito $C \equiv Mulher \sqcap Homem$.

$$Mulher \equiv Pessoa \sqcap SexoFeminino \tag{2.21}$$
$$Homem \equiv Pessoa \sqcap \neg Mulher \tag{2.22}$$

$$C \equiv Mulher \sqcap Homem \tag{2.23}$$
$$C' \equiv (Pessoa \sqcap SexoFeminino) \sqcap \neg(Pessoa \sqcap SexoFeminino) \tag{2.24}$$

Note que em (2.24)*, substituímos a definição de Mulher* (2.21) *e Homem* (2.22) *em C gerando C'.*

Apresentaremos um fato importante sobre os Tbox e suas expansões.

Teorema 2.34. ***(Satisfatibilidade de Conceitos e suas Expansões)***

Seja C um conceito de um Tbox $\mathcal{T}$ e C' sua expansão. C é satisfatível com respeito ao Tbox $\mathcal{T}$ se e somente se C' é satisfatível.

Reduzimos as nossas tarefas de inferência a verificação de satisfatibilidade do conceito expandido.

2.5.2 Tarefas de Raciocínio para Abox

Existem diversas tarefas de raciocínios referentes aos Abox. A seguir, iremos citar as principais.

Checagem de Consistência de um Abox com respeito a um Tbox Um Abox A é consistente com respeito a Tbox $\mathcal{T}$, se existe uma interpretação $\mathcal{I}$ que é modelo tanto de $\mathcal{T}$ como de $\mathcal{A}$ [3]. Quando dizemos que um Abox A é consistente, na verdade, estamos checando a consistência de A com respeito ao Tbox vazio.

Após a checagem de satisfatibilidade dos Tbox, podemos definir o Abox com asserções sobre os indivíduos do domínio. Após esse processo, precisamos checar a consistência do Abox em conjunto com o Tbox. Por que isso é necessário? A base de conhecimento deve ser obrigatoriamente consistente uma vez que, se ela for inconsistente, conseguiríamos concluir qualquer coisa.

Checagem de Instância É realizada a verificação de quando uma asserção α é acarretada por um Abox $\mathcal{A}$, ou seja, $\mathcal{A} \vDash \alpha$. Isso ocorre se toda interpretação que satisfaz $\mathcal{A}$ também satisfaz α. Se α é da forma $C(a)$, este problema pode ser reduzido ao problema de checagem de consistência de Abox uma vez que temos o seguinte teorema:

Teorema 2.35. ***(Redução da Checagem de Instância à Checagem de Consistência)***

$\mathcal{A} \vDash C(a)$ *se e somente se* $\mathcal{A} \cup \{\neg C(a)\}$ *é inconsistente.*

Problema de Recuperação Dado um Abox $\mathcal{A}$ e um conceito C, achar todos os indivíduos "a" tal que $\mathcal{A} \vDash C(a)$. Em outras palavras, podemos dizer que esta inferência trabalha recuperando todos os indivíduos que são instâncias de um conceito C.

Problema de Realização Apresenta-se como o problema dual ao de recuperação descrito acima. Dado um indivíduo "a" e um conjunto de conceitos, busca-se o conceito C mais específico, com respeito à ordem de subsunção $\sqsubseteq$, pertencente ao conjunto tal que $\mathcal{A} \vDash C(a)$.

Observe que a satisfatibilidade de conceitos também pode ser reduzida à checagem de consistência de Abox. Vimos na proposição 2.26 que os principais problemas de inferência para conceitos podem ser reduzidos a (in)satisfatibilidade de conceitos. Similarmente, a satisfatibilidade de conceitos pode ser reduzida a checagem de consistência de Abox. O teorema a seguir justifica esta afirmação:

[3] Na literatura padrão da lógica clássica (EBBINGHAUS et al., 1994), uma fórmula é satisfatível se ela tem um modelo. No contexto da Lógica de Descrição, usa-se o conceito de "consistência de Abox" quando, na verdade, deveríamos estar falando de "satisfatibilidade de Abox". Pela corretude, se uma fórmula é satisfatível então ela também é consistente. Logo, podemos buscar a satisfação de uma fórmula para provar a consistência da mesma.

Teorema 2.36. ***(Redução da Satisfatibilidade de Conceitos à Checagem de Consistência de Abox)***

Para todo conceito C, C é satisfatível se e somente se $\{C(a)\}$ é consistente, em que "a" é escolhido arbitrariamente.

As mais conhecidas implementações das tarefas de raciocínios citadas acima são: FaCT (FACT,), FaCT++ (FACT,), Pellet (SIRIN et al., 2007), Racer (RACER,) e Protegé (PROTéGé,). Para detalhes e mais informações sobre estes e outros raciocinadores, veja (DESCRIPTION...,).

Assim como foi feito para os conceitos, a checagem de consistência para um Abox com respeito a um Tbox acíclico pode ser reduzido a checagem de um Abox expandido. A seguir, definiremos a expansão de um Abox:

Definição 2.37. ***(Expansão de Abox com respeito a um Tbox)***

Definimos a expansão do Abox $\mathcal{A}$ com respeito ao Tbox $\mathcal{T}$ como o Abox $\mathcal{A}'$ que é obtido a partir de substituições das asserções de conceito $C(a)$ em $\mathcal{A}$ pela asserção de conceito $C'(a)$, onde C' é a expansão de C com respeito a $\mathcal{T}$.

Para todo modelo $\mathcal{I}$ do Tbox $\mathcal{T}$, o conceito C e sua expansão C' são interpretados da mesma maneira. Desta forma, $\mathcal{A}'$ é consistente com respeito ao Tbox $\mathcal{T}$ se e somente se $\mathcal{A}$ também é consistente. Como em $\mathcal{A}'$ encontramos asserções de conceito expandidas o máximo possível, ou seja, todas as asserções de conceito de $\mathcal{A}$ foram substituídas por suas expansões, $\mathcal{A}'$ é consistente com respeito ao Tbox $\mathcal{T}$ se e somente se ele é consistente. Concluímos assim:

Teorema 2.38. *$\mathcal{A}$ é consistente com respeito ao Tbox $\mathcal{T}$ se e somente se $\mathcal{A}'$ é consistente.*

A seguir, iremos apresentar o sistema dedutivo que implementa essas tarefas de raciocínio: o *tableaux*.

2.6 Algoritmos de *Tableaux* para $\mathcal{ALC}$

Nesta seção apresentaremos dois algoritmos de *tableaux* para a lógica $\mathcal{ALC}$. O primeiro somente permite axiomas de igualdade. O segundo, a ser apresentado no final desse capítulo, generalizará o primeiro permitindo axiomas tanto de igualdade como de inclusão. Como vimos no teorema 2.25, todas as tarefas de raciocínio podem ser reduzidas à subsunção. Só que esses algoritmos, ao invés de testar a subsunção de conceitos, utilizam a negação do que desejamos provar para reduzir a subsunção a (in)satisfatibilidade de conceitos, conforme visto no teorema 2.26 ($C \sqsubseteq D$ se e somente se $C \sqcap \neg D$ é insatisfatível).

Em (BAADER et al., 2003) encontramos o primeiro *tableaux* a ser apresentado para a lógica $\mathcal{ALCN}$. Aqui, descreveremos o algoritmo de *tableaux* para $\mathcal{ALC}$ uma vez que essa linguagem é a combinação da linguagem DL mais básica acrescida da negação de conceitos

arbitrários. Seu *tableaux* se tornará, assim, aproveitado para todas as outras extensões de $\mathcal{ALC}$.

O algoritmo inicia expandindo o conceito C que representa o conceito a ser analisado. Note que, pelo teorema 2.34, C é satisfatível com respeito ao Tbox $\mathcal{T}$ se e somente se C' é satisfatível, onde C' é o conceito expandido de C. Agora, podemos continuar verificando a satisfatibilidade do conceito C' ao invés de C.

Após este passo, precisamos internalizar as negações de C' de forma que estas somente ocorram diante de conceitos atômicos, ou seja, colocar C' na *forma normal de negação*. Obtemos assim o novo conceito C_0. Essa internalização das negações nos levam a não necessitar definir uma regra de inferência específica para a negação, uma vez que a negação já se encontra associada somente a fórmulas atômicas.

Pelo teorema 2.36, que nos garante que C_0 é satisfatível se e somente se $\{C_0(x_0)\}$ é consistente, onde x_0 é escolhido arbitrariamente, passamos a verificar se o Abox $\{C_0(x_0)\}$ é consistente. Novamente, pelo teorema 2.38, verificaremos a consistência do Abox expandido a partir de $\{C_0(x_0)\}$ o qual nominaremos $\{C'_0(x_0)\}$ ao invés de verificarmos a consistência do Abox $\{C_0(x_0)\}$.

Prosseguindo com o algoritmo, construiremos uma interpretação $\mathcal{I}$ tal que $C'^{\mathcal{I}}_0 \neq \varnothing$, ou seja, deve existir pelo menos um elemento em $\Delta^{\mathcal{I}}$ que é um elemento de $C'^{\mathcal{I}}_0$.

Não será imposto a "suposição do nome único" [4] para os Abox considerados pelo algoritmo. Em vez disso, é permitido asserções de inequações da seguinte forma: $x \neq y$. Definimos, assim, a semântica dessas asserções de inequações da seguinte forma: Uma interpretação $\mathcal{I}$ satisfaz $x \neq y$ se e somente se $x^{\mathcal{I}} \neq y^{\mathcal{I}}$, para indivíduos x e y. Assumimos que essas asserções são simétricas, ou seja, $x \neq y$ é equivalente a $y \neq x$.

Em seguida, o algoritmo aplica as regras de inferência inicialmente para o Abox $\mathcal{A}_0 = \{C'_0(x_0)\}$ (descritas a seguir), expandindo-o até que não haja mais regra passível de aplicação.

Se o Abox final resultante não contiver nenhuma contradição (chamado *clash*), então $\mathcal{A}_0$ é consistente (e consequentemente, C_0 é satisfatível). Caso contrário, $\mathcal{A}_0$ é inconsistente (e consequentemente, C_0 é insatisfatível). Definimos as *clashes* da seguinte forma:

Definição 2.39. ***(Clash em Abox)***

O Abox $\mathcal{A}$ contem uma clash se e somente se uma das duas situações abaixo ocorrer:

- $\{\bot(x)\} \subseteq \mathcal{A}$ *para algum indivíduo* x*;*
- $\{A(x), \neg A(x)\} \subseteq \mathcal{A}$ *para algum indivíduo* x *e algum papel* A

[4] A "suposição do nome único" nos garante que nomes diferentes sempre se referem a elementos distintos do domínio em questão.

Definição 2.40. ***(Regras de Inferência do Tableaux para $\mathcal{ALC}$)***

Suponha $\mathcal{A}$ o Abox em questão e $\mathcal{A}'$ o Abox expandido pela aplicação de uma das regras descritas a seguir sobre $\mathcal{A}$. As regras de inferência para o tableaux de $\mathcal{ALC}$ são:

Regra $\sqcap$

Condição $\mathcal{A}$ *contém* $(C_1 \sqcap C_2)(x)$, *mas não contém* $C_1(x)$ *nem* $C_2(x)$.

Ação $\mathcal{A}$'$= \mathcal{A} \cup \{C_1(x), C_2(x)\}$.

Regra $\sqcup$

Condição $\mathcal{A}$ *contém* $(C_1 \sqcup C_2)(x)$, *mas não contém* $C_1(x)$ *nem* $C_2(x)$.

Ação $\mathcal{A}$'$= \mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}$"$= \mathcal{A} \cup \{C_2(x)\}$.

Regra $\exists$

Condição $\mathcal{A}$ *contém* $(\exists R.C)(x)$), *mas não existe nenhum individuo z tal que C(z) e R(x,z) estão em* $\mathcal{A}$.

Ação $\mathcal{A}$'$= \mathcal{A} \cup \{C(y), R(x,y)\}$, *onde y é um individuo que não ocorre em* $\mathcal{A}$.

Regra $\forall$

Condição $\mathcal{A}$ *contém* $(\forall R.C)(x)$ *e R(x,y), mas não contém C(y).*

Ação $\mathcal{A}$'$= \mathcal{A} \cup \{C(y)\}$.

Um fato de grande importância e de destaque deste *tableaux* é que a regra de inferência que trata o quantificador existencial se diferencia da regra de inferência utilizada pelos *tableaux* de FOL. O algoritmo só introduz novos elementos se não existe nenhum individuo z tal que $C(z)$ e $R(x,z)$ estão em $\mathcal{A}$. Esse fato justifica a terminação deste *tableaux*.

A regra de inferência que trata a disjunção é não-determinística no sentido de que, ao aplicarmos a regra de inferência a um dado Abox, um número finito de novos Abox são gerados tal que o Abox original é consistente se e somente se os novos Abox também são. Por esta razão, consideraremos um conjunto finito de Abox $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ ao invés de Abox individuais.

Definição 2.41. *Um conjunto finito de Abox* $\mathcal{S} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ *é consistente se e somente se existe algum i,* $1 < i < k$, *tal que* $\mathcal{A}_i$ *é consistente.*

Uma regra de inferência é aplicada a um dado conjunto finito de Abox $\mathcal{S}$ da seguinte maneira: ele escolhe um elemento $\mathcal{A}$ de $\mathcal{S}$, e o substitui por uma Abox $\mathcal{A}$', por dois Abox $\mathcal{A}$' e $\mathcal{A}$", ou por um número finito de Abox $\mathcal{A}_{i,j}$. Os lemas a seguir são consequências das regras de inferência.

Lema 2.42. ***(Corretude)***[5]

Assuma que $\mathcal{S}$' é obtido de um conjunto finito de Abox $\mathcal{S}$ pela aplicação de regras de inferência descritas acima. Então $\mathcal{S}$ é consistente se e somente se $\mathcal{S}$' é consistente.

O lema seguinte destaca a propriedade de terminação do algoritmo de aplicação das regras de inferência:

Lema 2.43. ***(Terminação)***

Seja C_0 um conceito de $\mathcal{ALC}$ na forma normal de negação. Não existe uma sequência infinita de aplicações das regras de inferência.

A razão principal disto acontecer é descrita no lema seguinte:

Lema 2.44. *Seja um Abox contido em $\mathcal{S}_i$ para algum $i \geq 1$.*

- *Para todo indivíduo $x \neq x_0$ que ocorre em $\mathcal{A}$, existe uma única sequência $R_1, \ldots, R_l (l \geq 1)$ de papéis e uma única sequencia $x_1, \ldots, x_{l-1}$ de indivíduos tal que $\{R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_l(x_{l-1}, x)\} \subseteq \mathcal{A}$. Neste caso, podemos dizer que x ocorre no nível l em $\mathcal{A}$;*
- *Se $C(x) \in \mathcal{A}$ para um indivíduo x no nível l, então a profundidade máxima do papel de C (i.e., quantidade máxima de encadeamento de construtores envolvendo papeis) é limitada pela profundidade máxima de papeis de C_0 menos l. Consequentemente, o nível de qualquer indivíduo em $\mathcal{A}$ é limitado pela profundidade máxima do papel de C_0;*
- *Se $C(x) \in \mathcal{A}$, então C é uma sub-descrição de C_0. Consequentemente, o número de diferentes asserções de conceitos em x é limitado pelo tamanho de C_0;*
- *O número de diferentes sucessores de papel de x em $\mathcal{A}$ (i.e, indivíduos y tal que $R(x, y) \in \mathcal{A}$ para um papel R) é limitado pelo número de diferentes restrições existenciais em C_0.*

Começando com $\{\{C_0(x_0)\}\}$, portanto, podemos obter a partir de um número finito de aplicações das regras de inferência, um conjunto de Abox $\widehat{\mathcal{S}}$ que não é mais possível aplicação destas regras.

Definição 2.45. *Um Abox $\mathcal{A}$ é chamado completo se e somente se nenhuma das regras de transformação se aplicam a ele.*

A consistência de um conjunto de Abox completos pode ser decidido verificando a existência de pelo menos uma *clash* (veja definição 2.39). Obviamente, um Abox que contem uma *clash* não pode ser consistente. Assim, se todas as Abox em $\widehat{\mathcal{S}}$ contém uma *clash*, então $\widehat{\mathcal{S}}$ é inconsistente e, pelo lema 2.42, $[C_0(x_0)]$ é inconsistente. Consequentemente, C_0 é insatisfatível.

[5]Note que, conforme nota de rodapé 3, ao invés de "consistência", deveríamos estar usando a palavra "satisfatibilidade".

Lema 2.46. *(Completude)*[6]

Qualquer Abox $\mathcal{A}$ completo e sem clash tem um modelo.

O algoritmo de satisfatibilidade baseado em *tableaux* para $\mathcal{ALC}$ apresentado anteriormente pertence à classe de complexidade *PSpace-complete*. Note que o algoritmo apresentado aqui gera um Abox completo e sem *clash* cujo modelo é uma árvore binária de profundidade n e 2^{n+1} elementos. Isso nos leva à necessidade de tempo e espaço exponencial para esse algoritmo. Só que podemos alterar o algoritmo para que este gere não-deterministicamente os ramos desta árvore separadamente. Esse fato nos leva o algoritmo à complexidade NPSpace. Como NPSpace coincide com PSpace, podemos dizer que ele pertence à classe de complexidade PSpace. A completude desse problema é provada através de uma redução a partir do problema de verificação de validade de fórmulas booleanas quantificadas apresentado em (SCHMIDT-SCHAUBSS; SMOLKA, 1991).

2.6.1 Estendendo o *Tableaux* para Tbox cíclicos e com axiomas de inclusão

Até o momento, consideramos o problema de satisfatibilidade de Tbox acíclicos e sem os axiomas de inclusão. Para Tbox acíclicos, apenas é necessário expandir os axiomas terminológicos. Expansões, entretanto, não são mais possíveis de obter quando permitimos axiomas de inclusão de forma $C \sqsubseteq D$ em que C e D podem ser conceitos complexos e conter ciclos.

Como o algoritmo vai lidar com um conjunto de axiomas de inclusão? O primeiro passo é ao invés de considerarmos finitos axiomas de inclusão $C_1 \sqsubseteq D_1$, ..., $C_n \sqsubseteq D_n$, é suficiente considerar um axioma de inclusão único de seguinte forma $\top \sqsubseteq \widehat{C}$ em que $\widehat{C} = (\neg C_1 \sqcup D_1) \sqcap \ldots \sqcap (\neg C_n \sqcup D_n)$.

O axioma $\top \sqsubseteq \widehat{C}$ nos diz que todos os indivíduos devem pertencer ao conceito $\widehat{C}$. O algoritmo de *tableaux* apresentado acima, pode ser modificado para tratar de tais axiomas da seguinte maneira: todos os indivíduos x_0 (tanto os originais como os gerados pela regra de inferência existencial) são simplesmente instanciados a pertencerem a $\widehat{C}$, ou seja, $\widehat{C}(x_0)$. Essa modificação leva esse algoritmo ao não-determinismo. Este fato é entendido a partir do exemplo abaixo.

Exemplo 2.47. *Considere o que acontece se o algoritmo é aplicado para testar a consistência do Abox $\mathcal{A}_0 = \{A(x_0), (\exists R.A)(x_0)\}$ com respeito ao axioma $\top \sqsubseteq \exists R.A$: o algoritmo gera uma sequência infinita de Abox $\mathcal{A}_1, \mathcal{A}_2, \ldots$ e elementos $x_1, x_2, \ldots$ tal que $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{R(x_i, x_{i+1}), A(x_{i+1}), \exists R.A)(x_{i+1})\}$. Como todos indivíduos x_i recebem a mesma asserção de conceito que x_0, dizemos que o algoritmo entrou num ciclo.*

A terminação do algoritmo pode ser recuperada pela tentativa de detecção de tais ciclos e então bloquear a aplicação da regra geradora:

[6] Consistencia no sentido usualmente aplicado na literatura da lógica clássica (EBBINGHAUS et al., 1994).

Definição 2.48. ***(Bloqueio de Aplicação da Regra de Inferência Existencial)***

A aplicação da regra de inferência existencial a um indivíduo x é bloqueada por um individuo y num Abox $\mathcal{A}$ se e somente se $\{D|D(x) \in \mathcal{A}\} \subseteq \{D'|D'(y) \in \mathcal{A}\}$.

A idéia principal do bloqueio é que o indivíduo x bloqueado pode usar o sucessor de papel (i.e, indivíduos y tal que $R(x,y) \in \mathcal{A}$ para um papel R) de y ao invés de gerar novos. Continuando com o exemplo 2.47, ao invés de gerar um novo R-sucessor (i.e., sucessor do papel R) para x_1, nós poderíamos utilizar o R-sucessor de x_0. Isso produz uma interpretação $\mathcal{I}$ com $\Delta^{\mathcal{I}} = \{x_0, x_1\}$, $A^{\mathcal{I}} = \Delta^{\mathcal{I}}$, e $R^{\mathcal{I}} = \{(x_0, x_1), (x_1, x_1)\}$. Obviamente, $\mathcal{I}$ é modelo de $\mathcal{A}_0$ e do axioma $\top \sqsubseteq \exists R.A$.

Para evitar bloqueios cíclicos (de x por y e vice versa), consideramos uma enumeração de todos os indivíduos, e definimos que um indivíduo x só pode ser bloqueado por indivíduos y que ocorrem antes de x nessa enumeração.

Assim, a consistência de Abox de $\mathcal{ALC}$ com respeito a axiomas de inclusão e com ciclos é decidível. Deve ser destacado que a complexidade do algoritmo não é mais PSpace-complete, uma vez que ele pode gerar uma quantidade exponencial de indivíduos antes do bloqueio ocorrer. A satisfatibilidade de $\mathcal{ALC}$ com respeito a um axioma de inclusão único é conhecido ser EXPTIME-hard. Em (DONINI; MASSACCI, 2000), foi desenvolvido o melhor *tableaux*, ou seja, o mais eficiente, para $\mathcal{ALC}$ com axiomas de inclusão de complexidade EXPTIME. Apresentaremos na seção a seguir esse *tableaux*.

2.6.2 Um *Tableaux* EXPTIME para $\mathcal{ALC}$

O componente básico do *tableaux* apresentado em (DONINI; MASSACCI, 2000) é o par $e : \mathcal{C}$, composto pelo prefixo e e um conjunto de conceitos $\mathcal{C}$. Chamamos esse par de um **conjunto prefixado**. Um prefixo é uma sequência alternada de inteiros e papéis que inicia com 1. Formalmente,

Definição 2.49. ***(Prefixo)***

Se R é um papel e n um inteiro, temos que a sintaxe dos prefixos é:

$$e ::= 1 \mid e.R.n$$

Exemplo 2.50. ***(Exemplos de Prefixos e Conjuntos Prefixados)***

Se R e S são papéis,

- $1.R.5.S.11.R.12$ *é um prefixo e*
- $1.S.8.R.9 : \{A, B \sqcup \forall R.C\}$ *é um conjunto prefixado.*

Intuitivamente, um prefixo e no par $e : \mathcal{C}$ é o nome para um elemento do domínio de um modelo que o *calculus* tenta construir. Se o modelo é realmente construído, e satisfaz todos os conceitos contidos no conjunto $\mathcal{C}$.

Para definirmos um *tableaux*, adaptamos suas regras a partir de (SMULLYAN, 1995) e usamos uma notação uniforme para evitar qualquer redução a forma normal negada. Apresentamos a seguir a notação uniforme:

α	α_1	α_2
$C \sqcap D$	C	D
$\neg(C \sqcup D)$	$\neg C$	$\neg D$

Tabela 3: Notação Uniforme α

β	β_1	β_2
$\neg(C \sqcap D)$	$\neg C$	$\neg D$
$C \sqcup D$	C	D

Tabela 4: Notação Uniforme β

neg	pos
$\neg\neg C$	C

Tabela 5: Notação Uniforme neg

$v(R)$	v_0
$\forall R.C$	C
$\neg\exists R.C$	$\neg C$

Tabela 6: Notação Uniforme $v(R)$

$\pi(R)$	π_0
$\neg\forall R.C$	$\neg C$
$\exists R.C$	C

Tabela 7: Notação Uniforme $\pi(R)$

As regras, chamadas de Regras de Conjuntos Prefixados (CP-regras), são as seguintes:

- $\frac{e{:}\mathcal{C}\cup\{\alpha\}}{e{:}\mathcal{C}\cup\{\alpha_1,\alpha_2\}}(\alpha)$
- $\frac{e{:}\mathcal{C}\cup\{\beta\}}{e{:}\mathcal{C}\cup\{\beta_1\} \quad e{:}\mathcal{C}\cup\{\beta_2\}}(\beta)$
- $\frac{e{:}\mathcal{C}\cup\{neg\}}{e{:}\mathcal{C}\cup\{pos\}}(dneg)$
- $\frac{e{:}\mathcal{C}}{e{:}\mathcal{C}\cup\{\neg C\sqcup D\}}(KB)$, onde $C \sqsubseteq D \in KB$
- $\frac{e{:}\mathcal{C}\cup\{\pi(R)}{e.R.n{:}\{\pi_0\}}(\pi(R))$, em que e.R.n é novo no *tableaux* $\mathcal{T}$
- $\frac{e{:}\mathcal{C}\cup\{v(R)\} \quad e.R.n{:}\mathcal{D}}{e.R.n{:}\mathcal{D}\cup\{v_0\}}(v(R))$

Um *tableaux* para um conceito C é uma árvore diádica ordenada cujos pontos são ocorrências de conjuntos prefixados, os quais são contruídos da seguinte maneira: Iniciamos por $1 : \{C\}$ na raiz da árvore. Agora, suponha $\mathcal{T}$ um *tableaux* para o conceito C o qual já foi construído; Seja $\mathcal{B}$ um ramo em $\mathcal{T}$, isto é, um caminho da raiz até uma folha. Queremos expandir $\mathcal{T}$ utilizando as regras apresentadas acima da seguinte maneira: se os antecedentes da regra aparecem ao longo de $\mathcal{B}$, adicionamos a $\mathcal{B}$ o(s) consequente(s) da mesma. Para a regra β, nós simultaneamente adicionamos os dois consequentes como o sucessor direito e o sucessor esquerdo da folha. Note que o antecedente da regra não precisa ser uma folha.

Dizemos que um regra é aplicada a um conjunto prefixado $e : \mathcal{C}$ se $e : \mathcal{C}$ é o antecedente de uma regra sendo aplicada. Para a regra $v(R)$, a qual possui dois antecedentes, dizemos que a regra é aplicada ao primeiro antecedente (o conjunto prefixado $e : \mathcal{C} \sqcup \{v(R)\}$).

Analisando a regra $\pi(R)$, um prefixo está presente no *tableaux* $\mathcal{T}$, se existe algum conjunto prefixado em $\mathcal{T}$ com esse prefixo, e ele é novo se ele não está presente.

A mesma regra pode ser aplicável a diferentes conjuntos prefixados, diante da presença no conjunto do mesmo conceito. Por exemplo, iniciando o *tableaux* a partir de $1 : \{A \sqcap B, C \sqcup D\}$ e aplicando a regra α , o conjunto prefixado $1 : \{A, B, C \sqcup D\}$ é adicionado ao *tableaux* como sucessor da raiz. Agora, a regra β é aplicável a ambos raiz e seu sucessor; mas obviamente, não existe razão em aplicarmos a regra a raiz, uma vez que ela contém um conceito já decomposto. Para previnir ambiguidade, impomos a seguinte restrição: não podemos aplicar uma regra a um conjunto prefixado $e : \mathcal{C}$ se ele pertence a um ramo que já existe um conjunto prefixado $e' : \mathcal{C}$ (com o mesmo prefixo).

Para otimizarmos a aplicação das CP-regras, precisamos gerar maneira de reutilizarmos computações para a satisfatibilidade. Para tal propósito, precisamos da seguinte noção preliminar referente a conceitos reduzidos:

Definição 2.51. ***(Conceitos Reduzidos)***

Um conceito $C \in \mathcal{C}$ é CP-reduzido para um conjunto prefixado $e : \mathcal{C}$ no ramo $\mathcal{B}$ se e somente uma das seguintes condições for satisfeita:

1. *se C é do tipo α, e ambos α_1 e α_2 estão em $\mathcal{C}$;*
2. *se C é do tipo β, e β_1 ou β_2 está em $\mathcal{C}$;*
3. *se C é do tipo neg, e pos está em $\mathcal{C}$;*
4. *se C é do tipo $\pi(R)$, e existe outro conjunto prefixado $e.R.n : \mathcal{D}$ em $\mathcal{B}$ tal que $\pi_0 \in \mathcal{D}$;*
5. *se C é do tipo $v(R)$, e para todo conjunto prefixado $e.R.n : \mathcal{D}$ presente em $\mathcal{B}$, ele é $v_0 \in \mathcal{D}$.*

Além disso, uma inclusão $C \sqsubseteq D \in KB$ é CP-reduzido para um conjunto prefixado $e : \mathcal{C}$ em um ramo $\mathcal{B}$ se e somente se $\neg C \sqcup D \in \mathcal{C}$.

Definição 2.52. *(Conjunto Prefixado CP-reduzido)*

Um conjunto prefixado $e:\mathcal{C}$ é CP-reduzido em um ramo $\mathcal{B}$ se todo conceito $C \in \mathcal{C}$ é CP-reduzido para $e:\mathcal{C}$ e toda inclusão $C \sqsubseteq D \in KB$ é CP-reduzido para $e:\mathcal{C}$.

Não precisamos aplicar uma regra a um conceito que já é CP-reduzido , nem considerar conjuntos prefixados CP-reduzidos.

Entretanto, isso pode ser suficiente para evitarmos computações desnecessárias, mas não é o suficiente para prover a terminação. Precisamos introduzir a noção de conceitos implicitamente reduzidos e, para tal intuito, precisamos primeiramente da noção de ordenação lexicográfica padrão de prefixos:

Definição 2.53. ***(Ordenação Lexicográfica Padrão de Prefixos)***

A Ordenação Lexicográfica Padrão de Prefixos é obtida através do fecho transitivo (mas não reflexivo) das seguintes relações:

- $e \prec e.R.n$
- $e.R.n \prec e.R'.n'$, *em que* $n < n'$
- $e.R.e_1 \prec e'.R'.e_2$, *em que* $e \prec e'$

Exemplo 2.54. ***(Ordenação Lexicográfica Padrão de Prefixos)***

Por exemplo,

- $1.R.6 \prec 1.Q.11$
- $1.R.12.Q.20 \prec 1.Q.15.R.16$

Por simplicidade, manteremos o mesmo símbolo $\prec$ para o fecho transitivo das relações descritas acima.

Para identificar laços em um ramo, definimos a seguinte noção de testemunha:

Definição 2.55. ***(Testemunha de um Ramo)***

Um conjunto prefixado $e:\mathcal{C}$ é uma testemunha do ramo $\mathcal{B}$ para um conjunto prefixado $e':\mathcal{C}$, se $e \prec e'$, $e:\mathcal{C}$ é CP-reduzido e não existe nenhum outro conjunto prefixado $e'':\mathcal{C}$ em $\mathcal{B}$ tal que $e'' \prec e$.

Na definição acima, $\mathcal{C}$ é o mesmo conjunto prefixado de $e:\mathcal{C}$ e $e':\mathcal{C}$. Note também que se $e:\mathcal{C}$ tem uma testemunha, então ele é reduzido com respeito aos pontos *1-3* da definição 2.51. Como a relação $\prec$ é bem definida, existe no máximo uma testemunha para um dado conjunto $\mathcal{C}$. Claramente, não existe razão para continuarmos reduzindo um conjunto prefixado que tem uma testemunha.

Definição 2.56. ***(Conjunto Prefixado Implicitamente CP-reduzido)***

Um conjunto prefixado $e : \mathcal{C}$ é implicitamente CP-reduzido em um ramos $\mathcal{B}$ se e somente se ele é CP-reduzido ou tem uma testemunha $e' : \mathcal{C}$ em $\mathcal{B}$.

Precisamos agora de noções que nos permitam desenvolver técnicas de reutilização de computações para a insatisfatibilidade. Começamos relembrando a definição usual da inconsistência explícita.

Definição 2.57. ***(Conjunto Prefixado Inconsistente)***

Um conjunto prefixado $e : \mathcal{C}$ é inconsistente se existe um conceito C tal que ambos $C \in \mathcal{C}$ e $\neg C \in \mathcal{C}$. Para um dado conceito C, chamamos de clash o conjunto $\{C, \neg C\}$.

Para reutilizarmos conjuntos de conceitos insatisfatíveis introduzimos um tipo especial de conjuntos prefixados chamado de conjuntos inconsistentes ($\bot$-conjuntos), denotado por $e : (\mathcal{C})^{\bot}$.

Um $\bot$-conjunto contém um conjunto de conceitos cuja conjunção já foi provada ser inconsistente para a dada base de conhecimento KB, durante a expansão do *tableaux*.

A intuição é que $\bot$ é um rótulo extra que podemos propagar pelos rótulos dos nós do *tableaux*. Por isso, temos uma série de regras para a atualização dos rótulos, descritos a seguir:

- $\frac{C, \neg C \in \mathcal{C}}{e:(\mathcal{C})^{\bot}}(\bot)$
- $\frac{e':(\mathcal{C})^{\bot} \quad e' \prec e}{e:(\mathcal{C})^{\bot}}(\bot - testemunha)$
- $\frac{e:(\mathcal{C} \cup \{\alpha_1, \alpha_2\})^{\bot}}{e:(\mathcal{C} \cup \{\alpha\})^{\bot}}(\bot - \alpha)$
- $\frac{e:(\mathcal{C} \cup \{\beta_1\})^{\bot} \quad e:(\mathcal{C} \cup \{\beta_2\}}{e:\mathcal{C} \cup \{\beta\})^{\bot}}(\bot - \beta)$
- $\frac{e:(\mathcal{C} \cup \{pos\})^{\bot}}{e:(\mathcal{C} \cup \{neg\})^{\bot}}(\bot - dneg)$
- $\frac{e:(\mathcal{C} \cup \{\neg C \sqcup D\})^{\bot} \quad C \sqsubseteq D \in KB}{e:(\mathcal{C})^{\bot}}(\bot - KB)$
- $\frac{\mathcal{D} \sqsubseteq \{v_0 | v(R) \in \mathcal{C}\} \quad e.R.n:(\mathcal{D} \cup \{\pi_0\})^{\bot}}{e:(\mathcal{C} \cup \{\pi(R)\})^{\bot}}(\bot - \pi(R))$

As regras são aplicadas da seguinte maneira: se os conjuntos prefixados antecedentes ocorrerem no *tableaux* e o consequente $e : \mathcal{D}$ também ocorrer no *tableaux*, então atualizaremos o rótulo do consequente como $e : (\mathcal{D})^{\bot}$.

As $\bot$-regras são na sua maioria versões duais das CP-regras. A maior diferença é que as CP-regras são aplicadas dentro dos ramos e as $\bot$-regras são aplicadas entre os ramos, isto é, antecedentes e consequentes podem aparecer em diferentes ramos do *tableaux*. Então, agora podemos ampliar a definição de um conjunto prefixado inconsistente:

Definição 2.58. ***(Conjunto Prefixado Inconsistente)***

Um conjunto prefixado $e : \mathcal{C}$ é implicitamente inconsistente se ele é um $\bot$-conjunto.

Agora, uma vez que temos mais regras que simplesmente as CP-regras, precisamos adicionar a noção de conjunto prefixado $\bot$-reduzido.

Definição 2.59. ***(Conjunto Prefixado $\bot$-reduzido)***

Um conjunto prefixado $e : (\mathcal{C})^{\bot}$ é $\bot$-reduzido se as $\bot$-regras, ao serem aplicadas ao referido conjunto, não introduzem um novo $\bot$-conjunto.

A presença de testemunhas e a noção de conjuntos implicitamente inconsistentes, CP-reduzidos e $\bot$-reduzidos requer uma nova definição que abre e fecha ramos com respeito às definições padrões de *tableaux*.

Definição 2.60. ***(Ramo Implicitamente Fechado)***

Um ramo $\mathcal{B}$ é implicitamente fechado se existe um conjunto prefixado implicitamente inconsistente em $\mathcal{B}$. Um tableaux é implicitamente fechado se todos seus ramos são implicitamente fechados.

Definição 2.61. ***(Ramo Aberto)***

Um ramo $\mathcal{B}$ é aberto se todo conjunto prefixado em $\mathcal{B}$ for implicitamente CP-reduzido ou $\bot$-reduzido e $\mathcal{B}$ não for implicitamente fechado. Um tableaux $\mathcal{T}$ é aberto se pelo menos um de seus ramos for aberto.

Agora que definimos todas as noções que precisávamos, podemos citar os dois resultados principais cujas provas estão em (DONINI; MASSACCI, 2000).

Teorema 2.62. ***(Resultado 1)***

Se existir um tableaux implicitamente fechado para o conceito C com respeito ao Tbox KB, então C é insatisfatível com respeito à KB.

Teorema 2.63. ***(Resultado 2)***

Se existir um tableaux aberto para o conceito C com respeito ao Tbox KB, então C é satisfatível com respeito à KB.

A chave para o problema é como iremos achar um *tableaux* aberto ou implicitamente fechado utilizando tempo exponencial simples (EXPTIME) no tamanho do Tbox KB e do conceito C. Para tal intuito, precisamos aplicar as seguintes técnicas de busca de alto nível:

Técnica 1 Nunca aplique uma regra a um conjunto prefixado CP-reduzido, nem a um conjunto prefixado $\bot$-reduzido.

Analisando os conjuntos CP-reduzidos, ou eles são CP-reduzidos (e assim nenhuma CP-regra pode ser aplicada a eles) ou eles têm uma testemunha. Observe que seguindo essa técnica, somente a primeira testemunha encontrada pode ser devidamente CP-reduzida. De fato, um conjunto prefixado possuindo uma testemunha não será CP-reduzido com respeito as condições ***iv-v*** da definição 2.51.

Se um conjunto prefixado $\perp$-reduzido é inconsistente, então não há razão para prosseguirmos expandindo.

Iremos agora impor que não expandiremos conjuntos prefixados os quais já foram "provados" inconsistentes:

Técnica 2 Nunca aplique uma CP-regra a um conjunto prefixado implicitamente inconsistente.

Para aplicarmos essa técnica o máximo possível , evitando assim expansões desnecessárias, precisamos "descobrir" os $\perp$-conjuntos o mais cedo possível. Então,

Técnica 3 Aplique as regras que adicionam novos $\perp$-conjuntos antes de qualquer outra regra.

Técnica 4 Para todo conjunto prefixado $e : \mathcal{C}$, aplique as regras α, *pos*, KB e $v(R)$ antes das outras regras, e aplique a regra β antes da regra $\pi(R)$.

Técnica 5 Um novo prefixo e.R.n gerado pela regra $\pi(R)$ deve ser tal que $n > m$ para todo inteiro m já presente no *tableaux*.

Intuitivamente, essa técnica simplesmente diz que usamos um contador global para geração dos sucessores de e : primeiro gerando $e : R'.1$, então $e.R''.2$, em seguida $e.R'''.3$ e assim por diante, onde R', R'' e R''' podem ser o mesmo papel.

Técnica 6 Aplique uma regra a um conjunto prefixado $e : \mathcal{C}$ somente se não existir nenhum conjunto prefixado, $e' : \mathcal{D}$ com $e' \prec e$, ao qual a regra pode ser aplicada.

Técnica 7 Utilizamos a estratégia de busca em profundidade para percorrer todos os ramos do *tableaux*.

A combinação das técnicas 4 e 6 força a aplicação da regra $v(R)$ somente após a aplicação da regra $\pi(R)$. Isto é, somente após a regra $\pi(R)$ introduzir um novo prefixo, todos os conceitos v_0 impostos pela fórmula universal $V(R)$ são transferidos para um novo prefixo gerado pela aplicação da regra $v(R)$.

Essas técnicas preservam a completude e a corretude das estratégias de busca que as aplicam.

Diante de todas as técnicas já apresentadas,

Teorema 2.64. ***(Terminação e Complexidade do Tableaux para $\mathcal{ALC}$)***

Qualquer estratégia de busca que respeita as técnicas 1-7 termina utilizando tempo determinístico exponencial (EXPTIME) com relação ao tamanho do conceito C e do Tbox KB.

A prova desse teorema pode ser encontrada em (DONINI; MASSACCI, 2000).

Finalizamos assim nossa abordagem sobre as Lógicas de Descrição. No próximo capítulo, apresentaremos outro assunto chave para o desenvolvimento de nosso trabalho: a Lógica *Default* de Reiter.

3 A LÓGICA *DEFAULT* DE REITER

Neste capítulo, iremos abordar a Lógica *Default* de Reiter. Apresentaremos, de início, sua sintaxe e sua semântica. Logo após, introduziremos a noção de extensões, essencial para essa lógica. Algumas definições alternativas de extensões serão também apresentadas. Analisaremos alguns problemas possíveis de ocorrerem às extensões. Após desenvolvermos soluções para esses problemas, finalizamos este capítulo com a apresentação e análise de complexidade do algoritmo de construção de extensões.

3.1 Teoria *Default* e Definição de Extensões

Precisamos, como primeiro passo no nosso estudo da Lógica *Default*, definir uma teoria *default*:

Definição 3.1. ***(Teoria Default)***

Uma teoria default Δ *é um par* $\langle W, D \rangle$*, onde* W *é um conjunto de fatos representados como fórmulas da FOL e* D *é um conjunto de regras default úteis para completar a descrição feita em* W*. As regras default possuem o seguinte formato:*

$$\frac{\alpha(x) : \beta(x)}{\omega(x)}$$

Algumas observações se mostram necessárias sobre a regra *default* acima:

- $\alpha(x), \beta(x)$ e $\omega(x)$ são fórmulas de FOL;
- $\alpha(x)$ é chamada de pré-requisito da regra *default*;
- $\beta(x)$ é chamada de justificativa da regra *default*;
- $\omega(x)$ é chamada de conclusão da regra *default*;
- interpretamo-la da seguinte forma: "para todos os indivíduos x , se $\alpha(x)$ é derivado e é consistente assumirmos $\beta(x)$, então $\omega(x)$ é derivado". Em outras palavras: "se temos $\alpha(x)$, então geralmente temos $\omega(x)$, a menos que $\beta(x)$ não seja consistente com tudo o que temos".

As regras *default* podem ser de três tipos:

***default* normal:** $\frac{\alpha:\beta}{\beta}$

Este tipo de *default* possui conclusão igual a sua justificativa. De forma intuitiva, um *default* normal que conclui β só não é aplicável se $\neg\beta$ for derivável;

***default* semi-normal:** $\frac{\alpha:\beta\wedge\gamma}{\beta}$

O *default* semi-normal possui justificativas γ que acarretam, mas não são acarretadas por suas próprias conclusões β;

***default* não-normal:** $\frac{\alpha:\beta}{\omega}$, onde $\beta \nvdash \omega$

Este tipo de *default* possui justificativas β que não acarretam e nem são acarretadas por suas próprias conclusões ω.

Uma vez que nesta regra as justificativas são distintas de suas conclusões, se mostrando contra-intuitivo, essas regras não serão abordadas. Entretanto, nas definições de extensão, usaremos $\frac{\alpha:\beta}{\omega}$ para representar um *default* qualquer: normal, semi-normal ou não-normal. Apenas nas definições em que se faz necessário fazer a distinção entre *defaults* não-normais e os demais, é que deixaremos isso explicito.

Abaixo, apresentaremos um exemplo clássico em que representamos a característica dos pássaros voarem (em geral, a menos que ele seja um pinguim):

Exemplo 3.2. ***(Exemplo Clássico de Default)***

$$\frac{Passaro(x) : Voa(x) \wedge \neg Pinguim(x)}{Voa(x)} \tag{3.1}$$

$$Passaro(PiuPiu) \tag{3.2}$$

$$Pinguim(PiuPiu) \tag{3.3}$$

Podemos concluir, neste caso, que PiuPiu voa? Não, uma vez que (3.3) *é uma das justificativas de* (3.1) *e estas se contradizem, o que não nos permite aplicar* (3.1).

Os *defaults* podem ser classificados como *abertos* e *fechados*:

Definição 3.3. *(LUKASZEWICZ, 1990)* ***(Default Aberto e Fechado)***

Um default $\frac{\alpha(x):\beta(x)}{\gamma(x)}$ *é chamado aberto se e somente se pelo menos um* $\alpha(x)$, $\beta(x)$ *ou* $\gamma(x)$ *contém variável livre; caso contrário, o default será chamado de fechado.*

Definição 3.4. *(LUKASZEWICZ, 1990)* ***(Teoria Default Aberta e Fechada)***

Definimos assim uma teoria default como aberta se e somente se ela contém pelo menos um default aberto; caso contrário, o default é chamado de fechada.

Passaremos agora a formalizar o conceito de conjunto de conclusões deriváveis a partir de uma teoria *default*, o qual chamaremos de extensão.

Definição 3.5. ***(Extensão de uma teoria default)****(GINSBERG, 1987b)*

Sejam $T = \langle W, D\rangle$ uma teoria default fechada sobre uma linguagem $\mathcal{L}$ e Th o operador de consequência da lógica clássica de primeira ordem (FOL). Para qualquer conjunto de fórmulas $S \subseteq \mathcal{L}$, seja C(S) o menor conjunto de fórmulas a partir de $\mathcal{L}$ que satisfaz as seguintes propriedades:

- *$C(S) = Th(C(S))$, em outras palavras, $C(S)$ é fechado sobre o operador de consequência clássico;*
- *$W \subseteq C(S)$;*
- *Se $\frac{\alpha:\beta}{\omega} \in C(S)$ $\alpha \in S$ e $\neg\beta \notin S$, então $\omega \in C(S)$.*

Assim, um conjunto de fórmulas $E \subseteq \mathcal{L}$ é uma extensão da teoria default $T = \langle W, D\rangle$ se, e somente se, $E = C(E)$; em outras palavras, se, e somente se, E é um ponto fixo do operador C.

Para melhor entendimento de extensões, vejamos o exemplo abaixo:

Exemplo 3.6. $\Delta = \langle \{P(a)\}, \{\frac{P(a):Q(a)}{Q(a)}\}\rangle$

Existem duas possíveis extensões que satisfazem as condições de fechamento, que são:

$$E_1 = Th(P(a), Q(a))$$
$$E_2 = Th(P(a), \neg Q(a))$$

Entretanto, somente E_1 é uma extensão para Δ, visto que $C(E_1) = E_1$, mas $C(E_2) \neq E_2$.

3.2 Definições Alternativas para Extensões

Apresentaremos a seguir algumas definições alternativas em forma de teorema para extensão apresentadas no apêndice de (ETHERINGTON, 1988). Algumas definições e teoremas necessários para a apresentação dos teoremas alternativos para extensão também serão descritos.

Teorema 3.7. ***(Definição alternativa 1)***

E é uma extensão para $\Delta = \langle W, D\rangle$ se e somente se $E = \bigcup_{i=0}^{\infty} E_i$, onde

1. *$E_0 = W$, e para $i > 0$*
2. *$E_{i+1} = Th(E_i) \cup \{\omega | \frac{\alpha:\beta}{\omega}$, onde $\alpha \in E_i$, e $\neg\beta \notin E\}$*

Note que, neste teorema, utilizamos a própria extensão E na sua construção.

A seguir, apresentaremos o segundo teorema alternativo. Antes, precisamos apresentar algumas definições que são essenciais para o teorema em questão.

Iniciaremos com a definição do conjunto de *Defaults* Geradores (GD), isto é, o conjunto de *defaults* aplicáveis em uma extensão E.

Definição 3.8. ***(Defaults Geradores)***

O conjunto de defaults geradores (GD) para E com respeito a Δ *é definido como:*

$GD(E, \Delta) = \{\frac{\alpha:\gamma}{\beta} \in D | \alpha \in E \text{ e } \neg\gamma \notin E\}$

Seguindo a sequência de definições, apresentaremos a definição do conjunto de consequentes de um conjunto de *defaults*.

Definição 3.9. ***(Consequentes)***

Se D é um conjunto de defaults, então:

$Consequentes(D) = \{\gamma | \frac{\alpha:\beta}{\gamma} \in D\}$

Assim, chegamos a seguinte definição, novamente apresentada como um teorema:

Teorema 3.10. ***(Definição alternativa 2)***

Se E é uma extensão para $\Delta = \langle W, D \rangle$*, então*

$E = Th(W \cup Consequentes(GD(E, \Delta)))$.

O problema principal que encontramos nestes dois teoremas é que eles supõem que já temos a extensão E construída na própria definição da mesma. Precisamos, assim, desenvolver outra forma de definir extensões de modo que a extensão seja construida de forma iterativa, ou seja, que somente necessite da suposição de que alguns dados já tenham sido calculados previamente. Uma nova definição será analisada como solução na seção 4.3 e o algoritmo será construído na seção 4.4.

3.3 Os Problemas da Inexistência de Extensões e das Extensões Anômalas

Apresentaremos nesta seção os problemas encontrados na geração da extensão de uma teoria *default* a partir de dois exemplos. O primeiro demonstra o problema da inexistência de extensões:

Exemplo 3.11. ***(Inexistência de Extensão)*** *(ETHERINGTON, 1988)*

Suponha a seguinte teoria default:

$$W = \{\}$$
$$D = \{\frac{: A}{\neg A}\}$$

Quando passamos a efetuar o calculo de sua extensão, verificamos a sua impossibilidade de chegar a um ponto fixo pois se assumirmos que A é consistente, ao derivarmos $\neg A$ iremos bloquear a aplicação deste mesmo default gerando assim um ciclo, ou seja, a justificação A é negada pela sua consequência $\neg A$.Desta forma, não aplicar a regra $\frac{:A}{\neg A}$ força sua aplicação e vice-versa.

Este problema não ocorre somente com teorias *default* com apenas um *default* gerando o ciclo como demonstrado no exemplo 3.11. Os ciclos podem ocorrer com mais de um *default*. Isto é demonstrado no exemplo 3.12.

Exemplo 3.12. ***(Inexistência de Extensão)*** *(ETHERINGTON, 1988)*

Suponha a seguinte teoria default:

$$W = \{\}$$
$$D = \{\frac{: A \wedge \neg B}{A}, \tag{3.4}$$
$$\frac{: B \wedge \neg C}{B}, \tag{3.5}$$
$$\frac{: C \wedge \neg A}{C}\} \tag{3.6}$$

Quando passamos a efetuar o calculo de sua extensão, verificamos a sua impossibilidade de chegar a um ponto fixo. Isso ocorre devido ao ciclo obtido da seguinte maneira: se assumirmos que $A \wedge \neg B$ é consistente, ao derivarmos A a partir do default (3.4)*, nossa extensão em questão inicialmente seria igual a $\{A\}$. Proseguindo na aplicação dos default, supomos que $B \wedge \neg C$ é consistente e derivamos B. Só que agora, $A \wedge \neg B$ não será mais consistente com a extensão o que bloqueará a aplicação do default* (3.4)*. Este fato nos leva a A não mais pertencer à extensão. Assim, a extensão agora é igual a $\{B\}$. A próxima aplicação dos default será feita com o default* (3.6)*. Supomos que $C \wedge \neg A$ é consistente e derivamos C. Novamente, ocorrerá que $B \wedge \neg C$ não será mais consistente com a extensão o que bloqueará a aplicação do default* (3.5)*, não podendo mais B pertencer a extensão. Assim, a extensão agora é igual a $\{C\}$. Podemos novamente supor que $A \wedge \neg B$ é consistente e aplicar o default* (3.4)*. Com isso, $C \wedge \neg A$ não será mais consistente com a extensão o que bloqueará a aplicação do default* (3.6)*. Nossa extensão seria $\{A\}$. Chegamos assim a um ciclo. Em poucas palavras, o problema que ocorreu foi que A depende da ausência de B na extensão, B depende da ausência de C e C depende da ausência de A.*

O exemplo seguinte demonstra o problema da extensão anômala. Ele é amplamente conhecido como o "The Yale Shooting Problem" (em português, o problema de tiro de Yale)(GINSBERG, 1987b):

Exemplo 3.13. ***(Extensão Anômala)***

"The Yale Shooting Problem" consiste em determinar o que acontece com uma pessoa que a qualquer instante no tempo (situação) pode estar VIVO ou MORTO e uma arma que pode estar CARREGADA ou DESCARREGADA. Em um instante qualquer (situação) S_0*, a pessoa está viva* (3.7) *e a arma é carregada a qualquer momento em que o evento CARREGAR acontece* (3.8)*. O axioma* (3.9) *diz que a qualquer instante que a pessoa é atingida por uma arma carregada, ela morre. Em outras palavras, quando o evento ATIRAR ocorre na situação em que a arma está CARREGADA pode tornar o fato da pessoa estar VIVA ser falso. O axioma* (3.10) *trata de descrever que os fatos persistem a medida que os eventos ocorrem se este fato não é exceção para a ocorrência deste evento. Note que um fato anômalo seria quando uma pessoa é atingida pela arma carregada e esta pessoa permanece viva. Utilizamos a regra default para minimizar estes fatos anômalos, ou seja, um fato só é anômalo se é explicitamente dito isto usando o predicado AB.*

$$
\begin{aligned}
W = \{ & T(VIVO, S_0), && (3.7)\\
& \forall s T(CARREGADA, RESULTA(CARREGA, s)), && (3.8)\\
& \forall s T(CARREGADA, s) \rightarrow AB(VIVO, ATIRA, s) \wedge \\
& \quad T(MORTO, RESULTA(ATIRA, s)), && (3.9)\\
& \forall f, e, s T(f, s) \wedge \neg AB(f, e, s) \rightarrow T(f, RESULTA(e, s)) && (3.10)\\
\} & \\
D = \{ & \frac{: \neg AB(f, e, s)}{\neg AB(f, e, s)} \}. && (3.11)
\end{aligned}
$$

Descrevemos a seguir uma sequência de situações que utilizaremos para desenvolver as extensões desta teoria default:

$$
\begin{aligned}
& S_0, && (3.12)\\
& S_1 = RESULTA(CARREGA, S_0), && (3.13)\\
& S_2 = RESULTA(ESPERA, S_1), && (3.14)\\
& S_3 = RESULTA(ATIRA, S_2), && (3.15)\\
& \quad = RESULTA(ATIRA, RESULTA(ESPERA, RESULTA(CARREGA, S_0))).
\end{aligned}
$$

Em outras palavras, nosso indivíduo está inicialmente vivo (3.12)*. Então, a arma é carregada* (3.13)*. Em seguida, esperamos um momento* (3.14) *e então a arma é disparada contra o indivíduo* (3.15)*. Esperamos que ao final, este indivíduo morra. A*

primeira extensão passível de ser obtida (apresentaremos somente parte dela):

$$
\begin{aligned}
E_0 = \{&T(VIVO, S_0), && (3.16)\\
&\neg AB(VIVO, CARREGA, S_0), && (3.17)\\
&T(VIVO, S_1), && (3.18)\\
&T(CARREGADA, S_1), && (3.19)\\
&\neg AB(VIVO, ESPERA, S_1), && (3.20)\\
&\neg AB(CARREGADA, ESPERA, S_1), && (3.21)\\
&T(VIVO, S_2), && (3.22)\\
&T(CARREGADA, S_2), && (3.23)\\
&AB(VIVO, ATIRA, S_2), && (3.24)\\
&\neg AB(CARREGADA, ATIRA, S_2), && (3.25)\\
&T(MORTO, S_3), && (3.26)\\
&T(CARREGADA, S_3) && (3.27)\\
\}.&
\end{aligned}
$$

Note que nesta extensão temos que, ao final, o indivíduo morre (3.26).

Agora note a outra extensão também possível:

$$
\begin{aligned}
E_1 = \{&T(VIVO, S_0), && (3.28)\\
&\neg AB(VIVO, CARREGA, S_0), && (3.29)\\
&T(VIVO, S_1), && (3.30)\\
&T(CARREGADA, S_1), && (3.31)\\
&\neg AB(VIVO, ESPERA, S_1), && (3.32)\\
&AB(CARREGADA, ESPERA, S_1), && (3.33)\\
&T(VIVO, S_2), && (3.34)\\
&\neg T(CARREGADA, S_2) && (3.35)\\
&\neg AB(VIVO, ATIRA, S_2), && (3.36)\\
&T(VIVO, S_3), && (3.37)\\
\}.&
\end{aligned}
$$

Aplicando o default, obtemos $\neg AB(VIVO, ATIRA, S_2)$ (3.36) *e depois pela contrapositiva de (3.9), obtemos* $\neg T(CARREGADA, S_2)$ (3.35). *Mas como* $T(CARREGADA, S_1)$ (3.31)*, pela contrapositiva de (3.10), obtemos* $AB(CARREGADA, ESPERA, S_1)$ (3.33). *Note que nesta extensão temos que, ao final, o indivíduo está vivo* (3.37)*, mesmo a arma tendo sido disparada. Encontramos, assim, uma extensão anômala.*

Todas as definições apresentadas até agora não evitam os problemas exemplificados acima. Surge então uma pergunta: Qual o problema de inexistência de extensões para

Δ? O problema está nos ciclos. Como ciclos ocorrem nestes casos e ciclos não possuem ponto fixo, não conseguimos assim definir uma extensão.

E quanto ao caso das extensões anômalas? O que interessa gerar extensões que possuem informações errôneas ou impossíveis de ocorrerem devido à sua semântica? Uma solução para esse problema foi desenvolvida em (PEQUENO, 1994). A solução proposta por (PEQUENO, 1994), a qual será utilizada em nosso algoritmo de construção de extensões, é que as exceções sejam analisadas em primeiro lugar para depois podermos aplicar as regras *default*.

Nosso objetivo agora é encontrar a causa principal do problema da inexistência de extensões.

Começamos analisando o caso em que $\Delta = \langle W, D \rangle$, em que $D = \varnothing$. De (ETHERINGTON, 1988) temos:

Teorema 3.14. *Para $\Delta = \langle W, D \rangle$, em que $D = \varnothing$, sempre obteremos uma extensão única .*

Assim, vamos agora analisar os casos onde $W = \varnothing$ e $D \neq \varnothing$. Analisamos agora cada tipo de regra *default* e verificamos se elas são as causadoras de problema na definição das extensões.

***Default* Normais** Note que a conclusão dos *default* normais é também a sua justificativa. De forma intuitiva, um *default* normal que conclui β só não é aplicável se $\neg\beta$ for derivável. Tais *defaults* não podem introduzir inconsistências, nem podem refutar as justificativas de outras regras *default* normais já aplicadas, nem suas próprias justificativas. Chegamos assim a uma proposição:

Teorema 3.15. *(ETHERINGTON, 1988) Qualquer teoria envolvendo apenas default normais (teoria normal), tem obrigatoriamente pelo menos uma extensão.*

***Default* Semi-normais** Note que os *default* semi-normais $\frac{\alpha:\beta\wedge\gamma}{\beta}$ diferem dos *default* normais $\frac{\alpha:\beta}{\beta}$ por possuirem justificativas γ que acarretam, mas não são acarretadas por suas próprias conclusões β. O bom comportamento demonstrado pelas teorias *default* não permanece sobre as teorias com regras semi-normais. Podemos assim concluir que o problema está nos *default* semi-normais e não-normais ($\frac{\alpha:\beta}{\omega}$, onde $\beta \neq \omega$). Como descartamos os *default* não-normais no início do capítulo pois são não-intuitivos dado que permitem concluir algo cuja consistência não foi testada, não se mostra necessário analisar este caso.

Assim, passamos a buscar um padrão que nos permita caracterizar quando uma teoria com regras *default* semi-normais possui pelo menos uma extensão, ou seja, buscamos um padrão para a classe de teorias semi-normais que são coerentes.

Podemos notar que, uma vez que um *default* for aplicado, somente os conjuntos de suas justificativas não acarretadas por suas consequências são suscetíveis de refutação por outros *default*. Esses conjuntos são a chave da questão e serão discutidos abaixo.

O conflito entre uma extensão ser fechada sobre o conjunto de *default* aplicáveis e o teste da consistência das justificativas pode ocorrer somente se alguma fórmula depender da ausência de outra e ao mesmo tempo servir de suporte para inferência desta fórmula, em outras palavras, existir um ciclo de dependência entre as fórmulas.

3.4 Ordenação de *Defaults*

Nosso objetivo agora é verificar se nosso conjunto de dependências nos levam a dependências circulares não-resolvidas. A definição a seguir nos fornece a um método sintático de detecção de ciclos em uma teoria semi-normal. Observe que os *defaults* normais estarão sendo considerados nesta definição uma vez que eles são casos particulares dos *defaults* semi-normais.

Definição 3.16. *($\ll$ e $\underline{\ll}$)*

Seja $\Delta = \langle D, W\rangle$ uma teoria default semi-normal. Sem perda de generalidade, assuma que todas as fórmulas estão na forma de cláusulas. Abaixo definimos as relações parciais, $\ll$ e $\underline{\ll}$, em Literal $\times$ Literal, definimos:

1. *Se $\alpha \in W$, então $\alpha = (\alpha_1 \vee \ldots \vee \alpha_n)$, para algum $n \geq 1$. Para todo $\alpha_i, \alpha_j \in \{\alpha_1, \ldots, \alpha_n\}$, se $\alpha_i \neq \alpha_j$, seja $\neg\alpha_i \underline{\ll} \alpha_j$;*

2. *Se $\delta = \frac{\alpha : \beta \wedge \gamma}{\beta}$. Sejam $\alpha_1, \ldots, \alpha_r$, $\beta_1, \ldots, \beta_s$, e $\gamma_1, \ldots, \gamma_t$ os literais das formas clausais de α, β e γ, respectivamente. Então*

   (a) *Se $\alpha_i \in \{\alpha_1, \ldots, \alpha_r\}$ e $\beta_j \in \{\beta_1, \ldots, \beta_s\}$ então $\alpha_i \underline{\ll} \beta_j$;*

   (b) *Se $\gamma_i \in \{\gamma_1, \ldots, \gamma_t\}$ e $\beta_j \in \{\beta_1, \ldots, \beta_s\}$ então $\neg\gamma_i \ll \beta_j$;*

   (c) *$\beta = \beta_1 \wedge \ldots \wedge \beta_m$, para $m \geq 1$. Para cada $i \leq m$, $\beta_i = (\beta_{i,1} \vee \ldots \vee \beta_{i,m_j})$, onde $m_j \geq 1$. Então se $\beta_{i,j}$, $\beta_{i,k} \in \{\beta_{1,1}, \ldots, \beta_{m,m_m}\}$ e $\beta_{i,j} \neq \beta_{i,k}$ seja $\neg\beta_{i,j} \underline{\ll} \beta_{i,k}$.*

3. *A relação de transitividade vale para $\ll$ e $\underline{\ll}$, ou seja,*

   (a) *Se $\alpha \ll \beta$ e $\beta \ll \gamma$, então $\alpha \ll \gamma$;*

   (b) *Se $\alpha \underline{\ll} \beta$ e $\beta \underline{\ll} \gamma$, então $\alpha \underline{\ll} \gamma$;*

   (c) *Se $\alpha \ll \beta$ e $\beta \underline{\ll} \gamma$ ou $\alpha \underline{\ll} \beta$ e $\beta \ll \gamma$, então $\alpha \ll \gamma$.*

A definição acima é complexa, mas a intenção é que $\alpha \underline{\ll} \beta$ ou $\alpha \ll \beta$ se existe alguma maneira de α aparecer em uma inferência de β na teoria. A intuição por trás de *1* e *2.(c)* é que qualquer disjunção de n literais pode ser interpretada como uma implicação de qualquer um destes literais.

No item *2.(b)*, relacionamos o conjunto de literais $\{\gamma_1, \ldots, \gamma_t\}$ que fazem parte da justificativa com o conjunto de literais que fazem parte do consequente, ou seja, $\{\beta_1, \ldots, \beta_s\}$ através do uso da relação $\ll$. A negação, $\neg\gamma_i$, ocorre na parte *2.(b)* uma vez que não sabendo $\neg\gamma_i$ torna γ_i consistente com a extensão em questão.

Definição 3.17. ***(Teoria Default Ordenada)***

Uma teoria default é dita ordenada se e somente se não existem literais α, tal que $\alpha \ll \alpha$.

Em outras palavras, uma teoria ordenada não tem dependências circulares não-resolvidas.

A importância da ordenação para teorias *default* semi-normais é mostrada na proposição a seguir:

Teorema 3.18. ***(Coerência)***

Uma teoria default semi-normal ordenada tem no mínimo uma extensão.

Em outras palavras, a ordenação é uma condição suficiente para a existência de extensões.

Agora, podemos descrever um algoritmo de construção de extensões descrito em (ETHERINGTON, 1988).

3.5 Algoritmos de Construção de Extensões

Apresentaremos a seguir o algoritmo de contrução de extensões.

Ele constroi todas as extensões através de uma série de aproximações sucessivas. Cada aproximação, H_j, é construída a partir dos componentes de FOL em W aplicando os *defaults* um por vez. A cada passo, o *default* a ser aplicado é escolhido dentre aqueles, ainda não escolhidos, cujos pré-requisitos são conhecidos e cujas justificativas são consistentes com as aproximações anteriores e o estado atual da aproximação atual. Quando nenhum *default* for aplicável, o procedimento continua com a próxima aproximação. Se duas aproximações sucessivas são as mesmas, o procedimento é tido como concluído.

Em (ETHERINGTON, 1988) encontramos a prova de corretude e completude do algoritmo.

Teorema 3.19. *Existe uma computação convergente tal que $H_n = H_{n-1}$ e que $Th(H_n) = E$ se e somente se E é uma extensão para uma teoria default $\Delta = \langle D, W \rangle$.*

O algoritmo apresentado possui alguns problemas:

1. A escolha do *default* a ser aplicado a cada passo do laço interno pode fazer com que a construção da extensão leve muito mais tempo que o necessário. Isso ocorre quando o *default* escolhido para ser aplicado no momento tenha sido bloqueado;

2. O algoritmo gera todas as extensões, mas a geração das mesmas necessita do procedimento que implementa a não provabilidade;

Algoritmo 3.1 Algoritmo de construção de extensões(W,D)

$H_0 \leftarrow W$;
$j \leftarrow 0$;
Repetir
  $j \leftarrow j + 1$;
  $h_0 \leftarrow W$;
  $GD_0 \leftarrow \{\}$;
  $i \leftarrow 0$;
  Repetir
    $D_i \leftarrow \{\frac{\alpha:\beta}{\gamma} \in D | (h_i \vdash \alpha), (h_i \nvdash \neg\beta), (H_{j-1} \nvdash \neg\beta)\}$;
    Se $\neg\ null(D_i - GD_i)$ **Então**
      ESCOLHA δ **DE** $(D_i - GD_i)$;
      $GD_{i+1} \leftarrow GD_i \cup \{\delta\}$;
      $h_{i+1} \leftarrow h_i \cup \{CONSEQUENTES(\delta)\}$;
    Fim Se
    $i \leftarrow i + 1$;
  Até null$(D_{i-1} - GD_{i-1})$;
  $H_j = h_{i-1}$;
Até $H_j = H_{j-1}$;

3. Como o algoritmo gera todas as extensões, este também produz as extensões anômalas, descritas na seção 3.3.

No capítulo 4 apresentaremos algumas soluções para os problemas citados. Revisitaremos o teorema da coerência 3.18 apresentado em (ETHERINGTON, 1982) que nos propõe solução para o item 1. Apresentaremos nossa proposta de solução para o item 2. Finalmente, resolveremos o item 3 definindo uma ordem na aplicação das regras *defaults*.

4 A LÓGICA DE DESCRIÇÃO *DEFAULT*

Neste capítulo, apresentaremos a Lógica de Descrição *Default*. A Lógica de Descrição *Default* (DDL) é composta a partir da Lógica de Descrição acrescida de regras *default*. Conforme já mencionado, algumas simplificações são necessárias serem aplicadas à Lógica *Default* para que não nos deparemos com um problema indecidível: saber se um fórmula α pertence à uma Extensão. Abordaremos, a seguir, as simplificações a serem assumidas:

- Como conjunto de fatos (W) que compõem a teoria de descrição *default* utilizaremos a Lógica de Descrição (apresentada no capítulo 2) para representá-los;
- Buscando obtermos a decidibilidade do problema acima citado, assumiremos que o único tipo de regras *default* permitidas são as do tipo semi-normal. Os *defaults* normais serão considerados como caso particular dos *defaults* semi-normais. Os *defaults* não-normais não serão considerados conforme explicado na seção 3.1.
- Nosso domínio em questão deve ser finito, pois, na prática, nosso objetivo principal é descrever domínios existentes na *web* que sempre são finitos.

Por que não escolhemos trabalhar somente com teorias *default* normais? Se somente permitíssemos esse tipo específico de regras, limitaríamos muito nosso poder expressivo. Acima de tudo, temos o resultado apresentado no teorema 3.18 o qual nos garante que toda teoria *default* semi-normal ordenada tem no mínimo uma extensão. Com esse resultado, apenas precisamos definir esta ordenação.

Nesse capítulo, iniciamos apresentando a definição formal da DDL. Em seguida, descrevemos as definições para formalização das relações parciais necessárias para definição de ordenação de uma teoria de descrição *default* juntamente com os algoritmos escritos em pseudocódigo que as implementam e suas respectivas complexidades. Finalizamos o capítulo com a apresentação do algoritmo em pseudocódigo de construção de extensões para a DDL decidível. Nossa pesquisa envolve, assim, investigar, apresentar e desenvolver um sistema de representação de conhecimento não-monotônico que seja aplicável a *web* e que suas inferências sejam decidíveis, uma vez que para a FOL, esse problema é indecidível.

4.1 Teoria de Descrição *Default* e Definição de Extensões

A seguir, apresentamos a definição da DDL:

Definição 4.1. ***(Teoria de Descrição Default)***

Uma teoria de descrição default Δ é um trio $\langle T, A, D\rangle$, onde T e A representam os Tbox e Abox que descrevem o conjunto de fatos (para mais detalhes sobre a Lógica de Descrição, veja capítulo 2) e D é um conjunto de regras default do tipo semi-normal ($\frac{\alpha:\beta\wedge\gamma}{\beta}$) ou do tipo normal ($\frac{\alpha:\beta}{\beta}$), onde α, β, γ são fórmulas da Lógica de Descrição.

Os *defaults* da teoria de descrição *default* podem ser classificados como *abertos* e *fechados*:

Definição 4.2. *(LUKASZEWICZ, 1990)* ***(Default Aberto e Fechado)***

Um default $\frac{\alpha(x):\beta(x)}{\gamma(x)}$ é chamado aberto se e somente se pelo menos um $\alpha(x)$, $\beta(x)$ ou $\gamma(x)$ contém variável livre; caso contrário, o default será chamado de fechado.

Definição 4.3. ***(Teoria de Descrição Default Aberta e Fechada)***

Definimos assim uma teoria de descrição default como aberta se e somente se ela contém pelo menos um default aberto; caso contrário, o default é chamado de fechada.

Passaremos agora a formalizar seu conceito de extensão:

Definição 4.4. ***(Extensão de uma Teoria de Descrição Default)***

Sejam $\Delta = \langle T, A, D\rangle$ uma teoria de descrição default fechada sobre uma linguagem $\mathcal{L}$ e Th_d o operador de consequência da Lógica de Descrição (definição 2.19). Para qualquer conjunto de fórmulas $S \subseteq \mathcal{L}$, seja $C(S)$ o menor conjunto de fórmulas obtido a partir de $\mathcal{L}$ que satisfaz as seguintes propriedades:

- *$C(S) = Th_d(C(S))$, em outras palavras, $C(S)$ é fechado sobre o operador de consequência da Lógica de Descrição;*
- *$T \subseteq C(S)$;*
- *$A \subseteq C(S)$;*
- *Se $\frac{\alpha:\beta\wedge\gamma}{\beta} \in C(S)$, $\alpha \in S$, $\neg\beta \notin S$ e $\neg\gamma \notin S$, então $\beta \in C(S)$. No caso de default normal, desconsidere a restrição $\neg\gamma \notin S$.*

Assim, um conjunto de fórmulas $E \subseteq \mathcal{L}$ é uma extensão de Δ se, e somente se $E = C(E)$, em outras palavras, se, e somente se, E é um ponto fixo do operador C.

4.2 Pré-compilação para Construção de Extensões de DDL: Definição, Algoritmos e Complexidade

Iniciaremos nessa seção o desenvolvimento de um algoritmo de construção de extensões para a DDL. Tentaremos resolver os problemas 1 (otimizar a construção das extensões), 2 (algoritmos de provabilidade e não-provabilidade) e 3 (evitar as extensões anômalas) encontrados no algoritmo proposto na seção 3.5. Antes de passarmos ao algoritmo principal, precisamos realizar um pré-processamento da teoria.

O pré-processamento necessário consiste basicamente em:

1. Transformação das fórmulas da teoria de descrição *default* para a Forma Normal Clausal. Isso é necessário porque algumas partes do procedimento necessitam que as fórmulas envolvidas estejam nesse formato (veja **Algoritmos 4.3, 4.4, 4.10, 4.11, 4.12, 4.13, 4.15, 4.17, 4.18**). Esse processo será descrito na seção 4.2.1;

2. Verificação da existência de extensão da teoria de descrição *default* em questão. Para resolvermos esse problema precisamos verificar se existe algum ciclo de dependência entre as fórmulas da teoria (veja seção 3.3). Na seção 3.4, uma solução foi analisada e consiste no cálculo das relações $\underline{\ll}$ e $\ll$ e a verificação da existência de algum ciclo ($\alpha \ll \alpha$) em $\ll$ os quais, se existirem, impossibilitam a construção da extensão da teoria em questão. Essa solução será apresentada com detalhes na seção 4.2.2;

3. Organização dos *defaults* em partições. Organizamos os *defaults* em partições geramos uma ordem sobre aplicação dos *defaults* que respeita o principio das exceções primeiro proposto por (PEQUENO, 1994) para resolver o problema "Yale Shooting Problem", descrito na seção 3.3. Essa ordenação também otimiza processo de construção da extensão, resolvendo assim os problemas 1 e 3.

Após esses passos descritos acima, podemos passar para o algoritmo de construção da extensão.

Todos os algoritmos que serão apresentados nas seções a seguir terão suas complexidades analisadas. Todas elas serão escritas em função de **n**, onde **n** é igual ao número de ocorrências dos símbolos da teoria analisada. A seguir, apresentaremos com detalhes a definição de **n**. Iniciamos apresentando uma definição indutiva do número de ocorrências dos símbolos de um conceito:

Definição 4.5. ***(Número de Ocorrências de Símbolos de um Conceito)***

Seja α um $\mathcal{AL}$-Conceito. Definiremos indutivamente uma função T_C ($T_C : \alpha \to \mathbb{N}$) a partir de α que pode ser das seguintes formas (definição 2.3):

- *Se $\alpha = \top$, então $T_C(\alpha) = 1$;*
- *Se $\alpha = \bot$, então $T_C(\alpha) = 1$;*

- *Se* $\alpha = C$, *em que* C *é um* $\mathcal{AL}$*-conceito atômico, então* $T_C(\alpha) = 1$;
- *Se* $\alpha = \neg C$, *em que* C *é um* $\mathcal{AL}$*-conceito, então* $T_C(\alpha) = T_C(C) + 1$;
- *Se* $\alpha = C \sqcap D$, *em que* C *e* D *são* $\mathcal{AL}$*-conceitos, então* $T_C(\alpha) = T_C(C) + T_C(D) + 1$;
- *Se* $\alpha = C \sqcup D$, *em que* C *e* D *são* $\mathcal{AL}$*-conceitos, então* $T_C(\alpha) = T_C(C) + T_C(D) + 1$;
- *Se* $\alpha = \exists.R.C$, *em que* C *é um* $\mathcal{AL}$*-conceito e* R *é um papel, então* $T_C(\alpha) = T_C(C)+2$;
- *Se* $\alpha = \forall.R.C$, *em que* C *é um* $\mathcal{AL}$*-conceito e* R *é um papel, então* $T_C(\alpha) = T_C(C)+2$.

Assim, podemos definir o número de ocorrências de símbolos de axiomas terminológicos:

Definição 4.6. ***(Número de Ocorrências de Símbolos de Axiomas Terminológicos)***

Seja α *um axioma terminológico. Definiremos indutivamente uma função* T_{AX} *(*$T_{AX} : \alpha \to \mathbb{N}$*) a partir de* α *que pode ser das seguintes formas (definição 2.4):*

- *Se* $\alpha = C \sqsubseteq D$, *em que* C *e* D *são* $\mathcal{AL}$*-conceitos, então* $T_{AX}(\alpha) = T_C(C) + T_C(D) + 1$;
- *Se* $\alpha = C \equiv D$, *em que* C *e* D *são* $\mathcal{AL}$*-conceitos, então* $T_{AX}(\alpha) = T_C(C) + T_C(D) + 1$;

Finalmente, podemos definir o número de ocorrências de símbolos de um Tbox:

Definição 4.7. ***(Número de Ocorrências de Símbolos de um Tbox)***

Seja T *um Tbox. Definiremos indutivamente uma função* T_{Tbox} *(*$T_{Tbox} : T \to \mathbb{N}$*) a partir de* $T = \{\alpha_1, \ldots, \alpha_n\}$, *em que* $|T| = n$, *então o número de ocorrências de símbolos de* T, *é igual a:*

$$T_{Tbox}(T) = \sum_{i=1}^{n} T_{AX}(\alpha_i)$$

Agora iremos definir o número de ocorrências de símbolos de uma asserção:

Definição 4.8. ***(Número de Ocorrências de Símbolos de uma Asserção)***

Seja α *uma asserção. Definiremos indutivamente uma função* T_A *(*$T_A : \alpha \to \mathbb{N}$*) a partir de* α *que pode ser das seguintes formas (definições 2.6 e 2.7):*

- *Se* $\alpha = C(a)$, *em que* C *é um* $\mathcal{AL}$*-conceito e "a" um indivíduo, então* $T_C(\alpha) = 4$;
- *Se* $\alpha = R(a, b)$, *em que* R *é um papel e "a" e "b" são indivíduos, então* $T_C(\alpha) = 6$.

Definiremos o número de ocorrências de símbolos de um Abox:

Definição 4.9. ***(Número de Ocorrências de Símbolos de um Abox)***

Seja A um Abox. Definiremos indutivamente uma função T_{Abox} ($T_{Abox} : \alpha \to \mathbb{N}$) a partir de $A = \{\alpha_1, \ldots, \alpha_n\}$, em que $|A| = n$, então o número de ocorrências de símbolos de A, é igual a:

$$T_{Abox}(A) = \sum_{i=1}^{n} T_A(\alpha_i)$$

A seguir, definiremos o número de ocorrência de símbolos de uma regra *default*:

Definição 4.10. ***(Número de Ocorrências de Símbolos de uma Regra Default)***

Seja δ uma regra default. Definiremos indutivamente uma função T_D ($T_D : \delta \to \mathbb{N}$) a partir de δ que pode ser das seguintes formas:

- *Se $\delta = \frac{\alpha:\beta}{\beta}$, em que α e β são $\mathcal{AL}$-conceitos, então $T_D(\delta) = T_C(\alpha) + 2 \times T_C(\beta) + 1$;*
- *Se $\alpha = \frac{\alpha:\beta \wedge \gamma}{\beta}$, em que α, β e γ são $\mathcal{AL}$-conceitos, então $T_D(\delta) = T_C(\alpha) + 2 \times T_C(\beta) + T_C(\gamma) + 2$.*

Definiremos o número de ocorrências de símbolos de um conjunto de *defaults*:

Definição 4.11. ***(Número de Ocorrências de Símbolos de um Conjunto de Defaults)***

Seja D um conjunto de Defaults. Definiremos indutivamente uma função $T_{Default}$ ($T_{Default} : D \to \mathbb{N}$) a partir de $D = \{\delta_1, \ldots, \delta_n\}$, em que $|D| = n$, então o número de ocorrências de símbolos de D, é igual a:

$$T_{Default}(D) = \sum_{i=1}^{n} T_A(\delta_i)$$

Finalizando, definiremos o número de ocorrência de símbolos de uma teoria de descrição *default*:

Definição 4.12. ***(Número de Ocorrências de Símbolos de uma Teoria de Descrição Default)***

Seja $\Delta = \langle T, A, D \rangle$ uma teoria de descrição default. Definiremos indutivamente uma função T_{Teoria}($T_{Teoria} : \Delta \rangle \to \mathbb{N}$), então o número de ocorrências de símbolos de Δ, é igual a:

$$T_{Teoria}(\Delta) = T_{Tbox}(T) + T_{Abox}(A) + T_{Default}(D)$$

Iniciamos na seção a seguir, apresentando o passo 1, ou seja, a transformação das fórmulas escritas em Lógica de Descrição para a a Forma Normal Clausal.

4.2.1 Transformação para a Forma Normal Clausal: Definição, Algoritmos e Complexidade

Antes de apresentarmos a transformação das fórmulas da Lógica Proposicional para a Forma Normal Clausal, precisamos definir qual será o formato das fórmulas após a transformação. A Lógica Proposicional difere da Lógica de Primeira Ordem por não possuir os quantificadores. Utilizaremos essa formalização uma vez que eliminaremos os quantificadores em um dos passos mais a frente.

Definição 4.13. *(DALEN, 1994)* ***(Definição de uma fórmula na Forma Normal Clausal)***

Supondo:

$$\begin{cases} \bigwedge_{i\leq 0} \varphi_i = \varphi_0 \\ \bigwedge_{i\leq n+1} \varphi_i = \bigwedge_{i\leq n} \varphi_i \wedge \varphi_{n+1} \end{cases}$$

E

$$\begin{cases} \bigvee_{i\leq 0} \varphi_i = \varphi_0 \\ \bigvee_{i\leq n+1} \varphi_i = \bigvee_{i\leq n} \varphi_i \vee \varphi_{n+1} \end{cases}$$

Toda fórmula na Forma Normal Clausal possui o seguinte formato:

$\bigwedge_{i\leq n} \bigvee_{j\leq m_i} \varphi_{ij}$, *onde* φ_{ij} *é uma fórmula atômica ou sua negação.*

Antes de passarmos para o algoritmo principal, necessitamos aplicar alguns procedimentos preliminares responsáveis de transformar todas as fórmulas da teoria de tal forma que estas só possuam os conectivos $\wedge$, $\vee$ e $\neg$. Outro procedimento preliminar importante é o que transforma as fórmulas para a Forma Normal de Negação (NNF)(DALEN, 1994) preservando a equivalência entre as duas fórmulas. A tabela 8 aborda as regras de transformação aplicadas nesses passos. Note que a complexidade dessa etapa é igual ao número de conectivos da teoria. Esse valor é limitado superiormente por $O(n)$.

Regra	**Fórmula**	**Transformação**
Axioma de Igualdade	$\alpha \equiv \beta$	$(\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$
Axioma de Inclusão	$\alpha \sqsubseteq \beta$	$(\neg\alpha \vee \beta)$
Dupla Negação	$\neg\neg\alpha$	α
Interseção	$\alpha \sqcap \beta$	$(\alpha \wedge \beta)$
União	$\alpha \sqcup \beta$	$(\alpha \vee \beta)$
De Morgan I	$\neg(\alpha \sqcap \beta)$	$\neg\alpha \vee \neg\beta$
De Morgan II	$\neg(\alpha \sqcup \beta)$	$\neg\alpha \wedge \neg\beta$

Tabela 8: Regras de transformação das fórmula em DDL para fórmulas na forma normal negada e utilizando somente os conectivos $\wedge$ e $\vee$.

O próximo passo é a substituição dos quantificadores das fórmulas da teoria. Isso será feito da seguinte forma:

1. Substitua cada ocorrência de $\forall R.C$ por $(\neg R(x, a_1) \vee C(a_1)) \wedge \ldots \wedge (\neg R(x, a_m) \vee C(a_m))$, em que $a_1, \ldots, a_m$ são todas constantes que ocorrem no ABox;

2. Substitua cada ocorrência de $\neg\forall R.C$ por $(R(x,a_1) \wedge \neg C(a_1)) \vee \ldots \vee (R(x,a_m) \wedge \neg C(a_m))$, em que $a_1, \ldots, a_m$ são todas constantes que ocorrem no ABox;

3. Substitua cada ocorrência de $\exists R.C$ por $(R(x,a_1) \wedge C(a_1)) \vee \ldots \vee (R(x,a_m) \wedge C(a_m))$, em que $a_1, \ldots, a_m$ são todas constantes que ocorrem no ABox;

4. Substitua cada ocorrência de $\neg\exists R.C$ por $(\neg R(x,a_1) \vee \neg C(a_1)) \wedge \ldots \wedge (\neg R(x,a_m) \vee \neg C(a_m))$, em que $a_1, \ldots, a_m$ são todas constantes que ocorrem no ABox;

Esse passo possui complexidade igual a quantidade de quantificadores na fórmula multiplicado pela quantidade de constantes no domínio. Essa complexidade é limitada superiormente por $O(n^2)$.

Os principais algoritmos de Transformação para a Forma Normal Clausal são descritos em (JACKSON; SHERIDAN, 2004). Eles trabalham na aplicação da propriedade de distributividade da disjunção (ou seja, $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$). O algoritmo que mais se adapta a nossas necessidades é o *Standard* (JACKSON; SHERIDAN, 2004). Isso porque dentre os algoritmos descritos em (JACKSON; SHERIDAN, 2004), o único que preserva a equivalência entre as fórmulas original e transformada é o *Standard*. Os outros algoritmos somente preservam a satisfatibilidadade entre as fórmulas.

O algoritmo *Standard* busca, através da aplicação padrão da propriedade da distributividade da disjunção, internalizar as conjunções aos literais. Essa técnica produz uma fórmula equivalente a original. O problema desse algoritmo é que ao utilizarmo-lo em fórmulas na Forma Normal Disjuntiva ($\bigvee_{i\leq n} \bigwedge_{j\leq m_i} \varphi_{ij}$), obtemos o pior caso cuja complexidade é exponencial simples, ou seja, EXPTIME ($O(2^n)$).

Como precisamos de um algoritmo que transforme um conjunto de fórmulas, precisaremos criar um novo algoritmo que chama o algoritmo *Standard* para cada fórmula do conjunto a ser aplicado a transformação. Adaptaremos também ao caso do conjunto de *defaults* D. Assim, chegamos a necessidade de dois algoritmos. Eles são apresentados a seguir e possuem complexidade de $O(2^n \times n^2) = O(2^n)$ ou EXPTIME. O algoritmo 4.1 recebe como entrada os conjuntos de Tbox ou de Abox. Já o algoritmo 4.2 recebe o conjunto de *defaults* D.

Algoritmo 4.1 TransformaClausalTeA(C)

Entrada: Conjunto C que pode ser o conjunto de Tbox ou Abox da teoria em questão
Saída: O conjunto de entrada com todas as fórmulas na Forma Normal Clausal

1: $Res \leftarrow \varnothing$
2: **Para Todo** $\alpha \in C$ **Faça**
3: $\quad \alpha' \leftarrow Standard(\alpha)$
4: $\quad Res \leftarrow Res \cup \{\alpha'\}$
5: **Fim Para**
6: **Retorne** Res

Algoritmo 4.2 TransformaClausalD(D)

Entrada: Conjunto D de *defaults* da teoria em questão

Saída: O conjunto de entrada com todas as fórmulas na Forma Normal Clausal

1: $Res \leftarrow \varnothing$
2: **Para Todo** $\delta = \frac{\alpha:\beta\wedge\gamma}{\beta} \in D$ **Faça**
3: $\quad \alpha' \leftarrow Standard(\alpha)$
4: $\quad \beta' \leftarrow Standard(\beta)$
5: $\quad \gamma' \leftarrow Standard(\gamma)$
6: $\quad \delta' = \frac{\alpha':\beta'\wedge\gamma'}{\beta'}$
7: $\quad Res \leftarrow Res \cup \{\delta'\}$
8: **Fim Para**
9: **Retorne** Res

Assim, já obtemos um algoritmo EXPTIME ($O(2^n)$) que transforma uma teoria para a Forma Normal Clausal.

4.2.2 Ordenação de *Defaults*: Definição, Algoritmos e Complexidade

Pelos mesmos motivos apresentados na seção 3.3, a existência de pelo menos uma extensão para uma teoria de descrição *default* se mostra de extrema importância.

Para definirmos a ordenação, é necessário que as fórmulas da teoria em questão estejam na Forma Normal Clausal o qual já ocorre uma vez que estas passaram pelo procedimento descrito na seção 4.2.1.Vejamos a definição:

Definição 4.14. ***($\lessapprox$ e $\ll$ para DDL)***

Seja $\Delta = \langle A, T, D\rangle$ uma teoria de descrição default fechada (veja definição 4.3). Sem perda de generalidade, assuma que todas as fórmulas estão na Forma Normal Clausal (definida na seção 4.2.1). As relações parciais, $\lessapprox$ e $\ll$, em Literal $\times$ Literal, definimos:

1. *Se $\alpha \in (A \cup T)$, então $\alpha = (\alpha_1 \vee \ldots \vee \alpha_n)$, para algum $n \geq 1$. Para todo $\alpha_i, \alpha_j \in \{\alpha_1, \ldots, \alpha_n\}$, se $\alpha_i \neq \alpha_j$, seja $\neg\alpha_i \lessapprox \alpha_j$;*

2. *Se $\delta = \frac{\alpha:\beta\wedge\gamma}{\beta}$. Seja $\alpha_1, \ldots, \alpha_r$, $\beta_1, \ldots, \beta_s$, e $\gamma_1, \ldots, \gamma_t$ os literais das formas normais clausais de α, β e γ, respectivamente. Então*

   (a) *Se $\alpha_i \in \{\alpha_1, \ldots, \alpha_r\}$ e $\beta_j \in \{\beta_1, \ldots, \beta_s\}$ seja $\alpha_i \lessapprox \beta_j$*

   (b) *Se $\gamma_i \in \{\gamma_1, \ldots, \gamma_t\}$ e $\beta_j \in \{\beta_1, \ldots, \beta_s\}$ seja $\neg\gamma_i \ll \beta_j$*

   (c) *$\beta = \beta_1 \wedge \ldots \wedge \beta_m$, para $m \geq 1$. Para cada $i \leq m$, $\beta_i = (\beta_{i,1} \vee \ldots \vee \beta_{i,m_j})$, onde $m_j \geq 1$. Então se $\beta_{i,j}$, $\beta_{i,k} \in \{\beta_{1,1}, \ldots, \beta_{m,m_m}\}$ e $\beta_{i,j} \neq \beta_{i,k}$ seja $\neg\beta_{i,j} \lessapprox \beta_{i,k}$*

3. *A relação de transitividade vale para $\ll$ e $\lessapprox$, ou seja,*

(a) Se $\alpha \ll \beta$ *e* $\beta \ll \gamma$, *então* $\alpha \ll \gamma$;

(b) Se $\alpha \ll \beta$ *e* $\beta \ll \gamma$, *então* $\alpha \ll \gamma$;

(c) Se $\alpha \ll \beta$ *e* $\beta \ll \gamma$ *ou* $\alpha \ll \beta$ *e* $\beta \ll \gamma$, *então* $\alpha \ll \gamma$.

Para fins de implementação, nessa seção apresentaremos os algoritmos referentes a definição 4.14.

Algoritmo 4.3 Definição 4.14 item *1.*(A,T)

Entrada: Abox e Tbox

Saída: Conjunto com as tuplas da relação $\ll$

1: $A_c \leftarrow$ TransformaClausalTeA(A)
2: $T_c \leftarrow$ TransformaClausalTeA(T)
3: **Para Todo** $\alpha \in (A_c \cup T_c)$ **Faça**
4: **Para Todo** $\alpha_i \in \{\alpha_1, \ldots, \alpha_k\}$ **Faça**
5: **Para Todo** $\alpha_j \in \{\alpha_1, \ldots, \alpha_k\}$ **Faça**
6: **Se** $\alpha_i \neq \alpha_j$ **Então**
7: $\ll \leftarrow \ll \cup \{(\neg\alpha_i, \alpha_j)\}$
8: **Fim Se**
9: **Fim Para**
10: **Fim Para**
11: **Fim Para**
12: **Retorne** $\ll$

O **Algoritmo 4.3** possui complexidade EXPTIME uma vez que utiliza o **Algoritmo 4.1** nas linhas 1 e 2 cuja complexidade é a maior dentre os procedimentos realizados nesse algoritmo.

Algoritmo 4.4 Definição 4.14 item *2.*(D,$\underline{\ll}$)

Entrada: Conjunto de *Defaults* D e o conjunto de tuplas da relação $\underline{\ll}$

Saída: O conjunto de tuplas da relação $\underline{\ll}$ e o conjunto de tuplas da relação $\ll$

1: $D_c \leftarrow$ TransformaClausalD(D)
2: **Para Todo** $\delta = \frac{\alpha:\beta \wedge \gamma}{\beta} \in D_c$ **Faça**
3: **Para Todo** $\alpha_i \in \{\alpha_1, \ldots, \alpha_r\}$ **Faça**
4: **Para Todo** $\beta_j \in \{\beta_1, \ldots, \beta_m\}$ **Faça**
5: $\underline{\ll} \leftarrow \underline{\ll} \cup \{(\alpha_i, \beta_j)\}$
6: **Fim Para**
7: **Fim Para**
8: **Para Todo** $\gamma_i \in \{\gamma_1, \ldots, \gamma_t\}$ **Faça**
9: **Para Todo** $\beta_j \in \{\beta_1, \ldots, \beta_m\}$ **Faça**
10: $\ll \leftarrow \ll \cup \{(\neg\gamma_i, \beta_j)\}$
11: **Fim Para**
12: **Fim Para**
13: **Para** i =1 **Até** m **Faça**
14: **Para** j =1 **Até** m_i **Faça**
15: **Para** k =1 **Até** m_i **Faça**
16: **Se** $\beta_{i,j} \neq \beta_{i,k}$ **Então**
17: $\underline{\ll} \leftarrow \underline{\ll} \cup \{(\neg\beta_{i,j}, \beta_{i,k})\}$
18: **Fim Se**
19: **Fim Para**
20: **Fim Para**
21: **Fim Para**
22: **Fim Para**
23: **Retorne** $\underline{\ll}$,$\ll$

O **Algoritmo 4.4** possui complexidade EXPTIME uma vez que utiliza o **Algoritmo 4.2** na linha 1 cuja complexidade é a maior dentre os procedimentos realizados nesse algoritmo.

Algoritmo 4.5 Definição 4.14 item *3.(a)*($\lessdot$)

Entrada: Conjunto $\lessdot$

Saída: Conjunto $\lessdot$

1: Seja A um vetor inicializado vazio
2: $t \leftarrow 1$
3: **Para Todo** $(\alpha, \beta) \in \lessdot$ **Faça**
4: **Se** $\alpha \notin A$ **Então**
5: $A[t] \leftarrow \alpha$
6: t++
7: **Fim Se**
8: **Se** $\beta \notin A$ **Então**
9: $A[t] \leftarrow \beta$
10: t++
11: **Fim Se**
12: **Fim Para**
13: **Para** j = 1 **Até** t **Faça**
14: **Para** i = 1 **Até** t **Faça**
15: **Para** k = 1 **Até** t **Faça**
16: **Se** $((A[i], A[j]) \in \lessdot)$ **E** $((A[j], A[k]) \in \lessdot)$ **E** $((A[i], A[k]) \notin \lessdot)$ **Então**
17: $\lessdot \leftarrow \lessdot \cup \{(A[i], a[k])\}$
18: **Fim Se**
19: **Fim Para**
20: **Fim Para**
21: **Fim Para**
22: **Retorne** $\lessdot$

Algoritmo 4.6 Definição 4.14 item *3.(b)*($\ll$)

Entrada: Conjunto $\ll$

Saída: Conjunto $\ll$

1: Seja A um vetor inicializado vazio
2: $t \leftarrow 1$
3: **Para Todo** $(\alpha, \beta) \in \ll$ **Faça**
4: **Se** $\alpha \notin A$ **Então**
5: $A[t] \leftarrow \alpha$
6: t++
7: **Fim Se**
8: **Se** $\beta \notin A$ **Então**
9: $A[t] \leftarrow \beta$
10: t++
11: **Fim Se**
12: **Fim Para**
13: **Para** j = 1 **Até** t **Faça**
14: **Para** i = 1 **Até** t **Faça**
15: **Para** k = 1 **Até** t **Faça**
16: **Se** $((A[i], A[j]) \in \ll)$ **E** $((A[j], A[k]) \in \ll)$ **E** $((A[i], A[k]) \notin \ll$ **Então**
17: $\ll \leftarrow \ll \cup \{(A[i], a[k])\}$
18: **Fim Se**
19: **Fim Para**
20: **Fim Para**
21: **Fim Para**
22: **Retorne** $\ll$

Algoritmo 4.7 Definição 4.14 item *3.(c)*($\underline{\ll}$,$\ll$)

Entrada: Conjuntos $\underline{\ll}$ e $\ll$
Saída: Conjunto $\ll$

1: $C \leftarrow \underline{\ll} \cup \ll$
2: Seja A um vetor inicializado vazio
3: $t \leftarrow 1$
4: **Para Todo** $(\alpha, \beta) \in C$ **Faça**
5: **Se** $\alpha \notin A$ **Então**
6: $A[t] \leftarrow \alpha$
7: t++
8: **Fim Se**
9: **Se** $\beta \notin A$ **Então**
10: $A[t] \leftarrow \beta$
11: t++
12: **Fim Se**
13: **Fim Para**
14: **Para** j = 1 **Até** t **Faça**
15: **Para** i = 1 **Até** t **Faça**
16: **Para** k = 1 **Até** t **Faça**
17: **Se** $((A[i], A[j]) \in C)$ **E** $((A[j], A[k]) \in C)$ **E** $((A[i], A[k]) \notin C$ **Então**
18: $C \leftarrow C \cup \{(A[i], a[k])\}$
19: **Fim Se**
20: **Fim Para**
21: **Fim Para**
22: **Fim Para**
23: $\ll \leftarrow C$
24: **Retorne** $\ll$

Os **Algoritmos 4.5**, **4.6** e **4.7** implementam o fecho transitivo de de $\ll$ e $\underline{\ll}$. Eles foram desenvolvidos a partir do algoritmo de construção do fecho transitivo de um conjunto proposto em (LEWIS; PAPADIMITRIOU, 1997) cuja complexidade é igual a $O(n^3)$. Assim, a complexidade dos **Algoritmos 4.5, 4.6** e **4.7** é igual a $O(n^3)$.

Após a construção destes dois conjuntos, $\underline{\ll}$ e $\ll$, precisamos agora verificar se existem ciclos em $\ll$ uma vez que a definição 4.15 (equivalente a definição 3.17) nos diz que:

Definição 4.15. ***(Teoria Default Ordenada)***

Uma teoria default é dita ordenada se e somente se não existem literais α, tal que $\alpha \ll \alpha$.

Algoritmo 4.8 Verificação de Ciclos($\ll$)

Entrada: Conjunto $\ll$

Saída: **verdadeiro**, se existe um ciclo e **falso**, se não existir ciclo

1: **Para Todo** $(\alpha, \beta) \in \ll$ **Faça**
2: **Se** $\alpha = \beta$ **Então**
3: **Retorne verdadeiro**
4: **Fim Se**
5: **Fim Para**
6: **Retorne falso**

O **Algoritmo 4.8** verifica a existência de ciclos em $\ll$ e possui complexidade igual a $O(n)$.

Podemos agora escrever o algoritmo completo de ordenação dos *defaults*:

Algoritmo 4.9 Ordenação de *Defaults*(A,T,D)

Entrada: Conjuntos A, T e D

Saída: Conjuntos $\underline{\ll}$ e $\ll$ se não houverem ciclos em $\ll$, caso contrário, **falso**

1: $\underline{\ll} \leftarrow$ Definição 4.14 item *1*(A,T)
2: $\ll \leftarrow$ Definição 4.14 item *2*(D,$\underline{\ll}$)
3: $\underline{\ll} \leftarrow$ Definição 4.14 item *3.(a)*($\underline{\ll}$)
4: $\ll \leftarrow$ Definição 4.14 item *3.(b)*($\ll$)
5: $\underline{\ll}$,$\ll \leftarrow$ Definição 4.14 item *3.(c)*($\underline{\ll}$,$\ll$)
6: **Se** (Verificação de Ciclos($\ll$) = **verdadeiro**) **Então**
7: **Retorne falso**
8: **Fim Se**
9: **Retorne** $\underline{\ll}$,$\ll$

Assim, podemos concluir que a complexidade total da implementação da definição 4.14 é a complexidade máxima dentre os algoritmos descritos acima: EXPTIME, ou seja, a complexidade dos **Algoritmos 4.3 e 4.4** utilizados respectivamente nas linhas 1 e 2.

4.2.3 Organização dos *Defaults* em Partições: Definição, Algoritmos e Complexidade

A seguir, apresentaremos os passos necessários para a geração das partições dos *defaults* (item 3 da seção 4.2).

Para apresentarmos a definição e algoritmo da organização dos *defaults* em partições, precisamos de algumas definições preliminares.

Definiremos, a seguir, duas funções que serão amplamente utilizadas nas definições seguintes juntamente com seus algoritmos. A primeira, chama de ClausalFormula, é a função que retorna todas as cláusulas de certa fórmula. A outra função, chamada de

LiteraisFormula, retorna todos os literais de uma certa fórmula. Note que essa fórmula em DL já terá passado pela transformação para a Forma Normal Clausal apresentada na seção 4.2.1.

Definição 4.16. ***(Funções Clausais e Literais)***

Para $\beta = (\beta_{1,1} \vee \ldots \vee \beta_{1,m_1}) \wedge \ldots \wedge (\beta_{m,1} \vee \ldots \vee \beta_{m,m_m})$*, temos as seguintes funções:*

- *ClausalFormula*$(\beta) = \{(\beta_{i,1} \vee \ldots \vee \beta_{i,m_i}) | 1 \leq i \leq m\}$
- *LiteralFormula*$(\beta) = \{(\beta_{i,j} | 1 \leq i \leq m, 1 \leq j \leq m_i\}$

Em seguida, apresentamos seus algoritmos (**4.10** e **4.11**, respectivamente):

Algoritmo 4.10 Função ClausalFormula(β)
Entrada: Uma fórmula $\beta = \beta_1 \wedge \ldots \wedge \beta_m$
Saída: O conjunto Cl das cláusulas de β
1: **Para** i =1 **Até** m **Faça**
2: $Cl \leftarrow Cl \cup \{(\beta_i)\}$
3: **Fim Para**
4: **Retorne** Cl

O **Algoritmo 4.10** possui complexidade igual a $O(m)$ uma vez que possui um laço que percorre todas as cláusulas da fórmula β. Esse valor é limitado superiormente por $O(n)$.

Algoritmo 4.11 Função LiteralFormula(β)
Entrada: Uma fórmula $\beta = (\beta_{1,1} \vee \ldots \vee \beta_{1,m_1}) \wedge \ldots \wedge (\beta_{m,1} \vee \ldots \vee \beta_{m,m_m})$
Saída: O conjunto Li dos literais de β
1: **Para** i =1 **Até** m **Faça**
2: **Para** j =1 **Até** m_i **Faça**
3: $Li \leftarrow Li \cup \{\beta_{i,j}\}$
4: **Fim Para**
5: **Fim Para**
6: **Retorne** Li

O **Algoritmo 4.11** possui complexidade limitada superiormente por $O(n)$, uma vez que os dois laços percorrem todos os literais da fórmula β.

Estas funções foram definidas para entradas de fórmulas. Necessitaremos dessas mesmas funções contudo elas receberão como entrada um conjunto de fórmulas, assim os **Algoritmos 4.10** e **4.11** serão reescritos para tratarem deste novo tipo de entrada.

Algoritmo 4.12 Função ClausalConjunto(C)

Entrada: O conjunto C de fórmulas $\beta = \beta_1 \wedge \ldots \wedge \beta_m$
Saída: O conjunto CL das cláusulas do conjunto C

1: **Para Todo** $\beta = \beta_1 \wedge \ldots \wedge \beta_m \in$ C **Faça**
2: **Para** i =1 **Até** m **Faça**
3: $CL \leftarrow CL \cup \{(\beta_i)\}$
4: **Fim Para**
5: **Fim Para**
6: **Retorne** CL

O **Algoritmo 4.12** possui complexidade limitada superiormente por $O(n)$, uma vez que os dois laços aninhados percorrem todas as cláusulas do Conjunto C.

Algoritmo 4.13 Função LiteralConjunto(C)

Entrada: O conjunto C de fórmulas $\beta = (\beta_{1,1} \vee \ldots \vee \beta_{1,m_1}) \wedge \ldots \wedge (\beta_{m,1} \vee \ldots \vee \beta_{m,m_m})$
Saída: O conjunto C de literais

1: **Para Todo** $\beta = (\beta_{1,1} \vee \ldots \vee \beta_{1,m_1}) \wedge \ldots \wedge (\beta_{m,1} \vee \ldots \vee \beta_{m,m_m}) \in$ Lit **Faça**
2: **Para** i =1 **Até** m **Faça**
3: **Para** j =1 **Até** m_i **Faça**
4: $LI \leftarrow LI \cup \{\beta_{i,j}\}$
5: **Fim Para**
6: **Fim Para**
7: **Fim Para**
8: **Retorne** LI

O **Algoritmo 4.13** possui complexidade limitada superiormente por $O(n)$, uma vez que os dois laços aninhados percorrem todos literais do Conjunto C.

Agora, revisitaremos a definição 3.9. Apresentamos em seguida sua implementação.

Definição 4.17. ***(Consequentes)***

Se D é um conjunto de defaults, então:

$Consequentes(D) = \{\beta | \frac{\alpha:\beta \wedge \gamma}{\beta} \in D\}$

Algoritmo 4.14 Função Consequentes(D)

Entrada: O conjunto D de *defaults*
Saída: O conjunto Co de fórmulas

1: **Para Todo** $\delta = \frac{\alpha:\beta,\gamma}{\beta} \in D$ **Faça**
2: $Co \leftarrow Co \cup \{\beta\}$
3: **Fim Para**
4: **Retorne** Co

O **Algoritmo 4.14** possui complexidade igual a $O(n)$.

Continuando nossas definições, apresentaremos a definição do Universo de uma teoria Δ. Este conjunto possui todos os literais que formam todas as fórmulas da teoria em questão.

Definição 4.18. ***(Universo de Δ)***

Para uma teoria de descrição default, $\Delta = \langle T, A, D \rangle$, definimos:

$$
\begin{aligned}
U(\Delta) = & \{\alpha | \alpha \in Literais \\
& e\ [\exists \phi.[(\alpha \vee \phi) \in Clausulas(T \cup A \cup Consequentes(D))] \\
& ou\ [(\neg\alpha \vee \phi) \in Clausulas(T \cup A \cup Consequentes(D))]\} \\
& \cup \\
& \{\alpha_i | \exists \alpha, \beta, \gamma. \frac{\alpha : \beta, \gamma}{\beta} \in D\ e\ \alpha_i \in Literais(\alpha)\} \\
& \cup \\
& \{\neg\gamma_i | \exists \alpha, \beta, \gamma. \frac{\alpha : \beta, \gamma}{\beta} \in D\ e\ \gamma_i \in Literais(\gamma)\}
\end{aligned}
$$

Note que ϕ pode ser vazia.

O **Algoritmo 4.15** implementa a definição Universo de Δ.

Algoritmo 4.15 Universo(A,T,D)

Entrada: Os conjuntos A, T e D
Saída: O conjunto Un de fórmulas

1: $T_c \leftarrow$ TransformaClausalTeA(T)
2: $A_c \leftarrow$ TransformaClausalTeA(A)
3: $D_c \leftarrow$ TransformaClausalD(D)
4: **Para Todo** $\alpha \in$ LiteralConjunto$(A_c \cup T_c \cup Consequentes(D_c))$ **Faça**
5: **Se** $((\alpha \vee \beta)$ **OU** $(\neg\alpha \vee \beta)) \in$ ClausalConjunto$(A_c \cup T_c \cup Consequentes(D_c))$ **Então**
6: $Un \leftarrow Un \cup \{\alpha\}$
7: **Fim Se**
8: **Fim Para**
9: **Para Todo** $\delta = \frac{\alpha:\beta,\gamma}{\beta} \in D_c$ **Faça**
10: **Para Todo** $\alpha_i \in LiteralFormula(\alpha)$ **Faça**
11: $Un \leftarrow Un \cup \{\alpha_i\}$
12: **Fim Para**
13: **Fim Para**
14: **Para Todo** $\delta = \frac{\alpha:\beta,\gamma}{\beta} \in D_c$ **Faça**
15: **Para Todo** $\gamma_i \in LiteralFormula(\gamma)$ **Faça**
16: $Un \leftarrow Un \cup \{\neg\gamma_i\}$
17: **Fim Para**
18: **Fim Para**
19: **Retorne** Un

O **Algoritmo 4.15** possui complexidade igual a maior complexidade entre o algoritmo de Transformação para a Forma Normal Clausal (utilizado nas linhas 1, 2 e 3) que é EXPTIME e dos três laços do algoritmo.

O laço um (linhas 4-8) possui complexidade igual a $O(n^2)$: A $|ClausalConjunto(A \cup T \cup Consequentes(D))|$ é igual a $O(n)$ e a busca feita na linha 2 possui complexidade igual a $O(n)$.

O laço dois (linhas 9-13) possui complexidade igual a $O(n^2)$: A $|D|$ é limitada superiormente por $O(n)$ e o laço interno (linha 10-12) possui complexidade igual a $O(n)$.

O laço três (linhas 14-18) possui complexidade igual a $O(n^2)$A $|D|$ é limitada superiormente por $O(n)$ e o laço interno (linha 15-17) possui complexidade igual a $O(n)$.

Assim, podemos concluir que a complexidade do **Algoritmo 4.15** é igual a EXPTIME.

Para todos os elementos do universo de uma teoria, definiremos um valor que chamaremos de *length*. Esta função será utilizada na definição das partições.

Definição 4.19. ***(Length do Universo de Δ)***

Para uma teoria de descrição default semi-normal ordenada $\Delta = \langle A, T, D \rangle$, definimos a função $l : U(\Delta) \to \mathbb{N}$ da seguinte maneira:

- *Se $\alpha, \beta \in U(\Delta)$ e $\alpha \trianglelefteq \beta$, então $l(\alpha) \leq l(\beta)$. Se $\alpha \lhd \beta$, então $l(\alpha) < l(\beta)$*
- *Se $\beta \in U(\Delta)$ e para nenhum $\alpha \in U(\Delta)$ tal que $\alpha \trianglelefteq \beta$ ou $\alpha \lhd \beta$, então então $l(\beta) = 0$*
- *Se $n \in \mathbb{N}$, $\beta \in U(\Delta)$, e $l(\beta) > n$ então $\exists \alpha \in U(\Delta).(\alpha \lhd \beta)$ e $l(\alpha) = n$*

Em outras palavras, podemos dizer que $l(\alpha)$ é a maior cadeia de defaults que aparecem numa inferência de α.

O **Algoritmo 4.16** implementa $l : U(\Delta) \to \mathbb{N}$.

Algoritmo 4.16 *length*(A,T,D,$\lessdot$, $\ll$)

Entrada: Os conjuntos A, T, D, $\lessdot$ e $\ll$

Saída: O conjunto L com a relação entre as fórmulas e seus *lengths*

1: $Un \leftarrow$ Universo(A,T,D)
2: $L \leftarrow \varnothing$
3: **Para Todo** $\beta \in Un$ **Faça**
4: $aux \leftarrow$ **verdadeiro**
5: **Para Todo** $\alpha \in Un$ **Faça**
6: **Se** $(\alpha, \beta) \in (\lessdot \cup \ll)$ **Então**
7: $aux \leftarrow$ **falso**
8: **Fim Se**
9: **Fim Para**
10: **Se** $aux =$ **verdadeiro Então**
11: $L \leftarrow L \cup \{(\beta, 0)\}$
12: **Fim Se**
13: **Fim Para**
14: **Para Todo** $(\alpha, N) \in L$, onde N é um número natural **Faça**
15: **Para Todo** $(\alpha, \beta) \in \ll$ **Faça**
16: $L \leftarrow L \cup \{(\beta, n+1)\}$
17: **Fim Para**
18: **Fim Para**
19: **Para Todo** $(\alpha, N) \in L$, onde N é um número natural **Faça**
20: **Para Todo** $(\alpha, \beta) \in \lessdot$ **Faça**
21: $L \leftarrow L \cup \{(\beta, n)\}$
22: **Fim Para**
23: **Fim Para**
24: **Retorne** L

O **Algoritmo 4.16** possui complexidade igual a EXPTIME. Essa complexidade surge da maior complexidade entre a subrotina da construção do conjunto do Universo (linha 1) cuja complexidade é EXPTIME e os três laços. O primeiro laço é formado por um laço duplo (linhas 3-13, 5-9) e uma busca (linha 6) cuja complexidade é igual a $O(n^2)$ que é igual a $|\lessdot \cup \ll|$. Chegamos a complexidade igual a $O(n^6)$.

O segundo (linha 14-18) e terceiro (linha 19-23) são formados por laços duplos (no caso do segundo laço, linhas 14-18 e 15-17 e no caso do terceiro laço, linhas 19-23 e 20-22) que percorrem todo conjunto L, cuja cardinalidade é igual $O(n^2)$ uma vez que a função *length* mapeia todos os elementos do Universo que possui cardinalidade igual a $O(n^2)$. O laço mais interno percorre todos os elementos do conjunto $\lessdot$, no caso do segundo laço e $\ll$ no terceiro laço cuja cardinalidade é $O(n^2)$. Logo, a complexidade desses laços é $O(n^4)$.

Em seguida, temos a definição dos *length* máximo e mínimo de uma teoria DDL.

Definição 4.20. ***(l_{MAX}, l_{MIN})***

Para uma teoria de descrição default semi-normal ordenada, $\Delta = \langle A, T, D \rangle$ *e*

$\beta = (\beta_{1,1} \vee \ldots \vee \beta_{1,m_1}) \wedge \ldots \wedge (\beta_{m,1} \vee \ldots \vee \beta_{m,m_m})$.

- $l_{MAX} = MAX(l(\beta_{i,j}))$ *e*
- $l_{MIN} = MIN(l(\beta_{i,j}))$, *onde* $1 \leq i \leq m$ *e* $1 \leq j \leq m_i$

Para prosseguirmos, é necessário que β e as fórmulas da teoria em questão estejam na Forma Normal Clausal o que já ocorre uma vez que estas passaram pelo procedimento descrito na seção 4.2.1.

Algoritmo 4.17 $l_{MAX}(\beta, L)$

Entrada: Uma fórmula e o conjunto L
Saída: Um número natural

```
1:  max ← 0
2:  β_c ← Standard(β)
3:  Lm ← LiteralFormula(β_c)
4:  Para Todo α ∈ Lm Faça
5:     Para Todo (α, n) ∈ L Faça
6:        Se n > max Então
7:           max ← n
8:        Fim Se
9:     Fim Para
10: Fim Para
11: Retorne max
```

Algoritmo 4.18 $l_{MIN}(\beta, L)$

Entrada: Uma fórmula β e o conjunto L
Saída: Um número natural

```
1:  min ← 0
2:  β_c ← Standard(β)
3:  Lm ← LiteralFormula(β_c)
4:  Para Todo α ∈ Lm Faça
5:     Para Todo (α, n) ∈ L Faça
6:        Se n < min Então
7:           min ← n
8:        Fim Se
9:     Fim Para
10: Fim Para
11: Retorne min
```

A complexidade dos **Algoritmos 4.17** e **4.18** é dada pela maior complexidade entre a linha 2 que é EXPTIME, ou seja, a complexidade do algoritmo *Standard* apresentado na seção 4.2.1, da linha 3 que é $O(n)$ e do laço de linhas 4-10. Esse, por sua vez,

percorre os n elementos do conjunto Lm construído na linha 2 cuja complexidade é $O(n)$. Internamente, temos um laço (linhas 5-9) que possui complexidade $O(n^2)$. Chegamos a complexidade final de EXPTIME.

Finalmente, chegamos a definição das partições. Ela será descrita em forma de teorema:

Teorema 4.21. *(ETHERINGTON, 1988)* ***(Partição $\{P_i\}$)***

Seja uma teoria de descrição default semi-normal ordenada, então existe uma partição para D, $\{P_i\}$, induzido por

$$\forall \delta \in D.\delta = \frac{\alpha:\beta \wedge \gamma}{\beta} \text{ e } l_{MIN}(\beta) = i \Leftrightarrow \delta \in P_i$$

O algoritmo 4.19 implementa divisão dos *default* em partições.

Algoritmo 4.19 Particões(A,T,D,$\lessdot$,$\ll$)

Entrada: Os conjuntos A,T,D,$\lessdot$ e $\ll$

Saída: Um conjunto P com conjuntos P_i que dividem o conjunto D de *default* em partições

1: $max \leftarrow 0$
2: $L \leftarrow length(A, T, D, \lessdot, \ll)$
3: **Para Todo** $\delta = \frac{\alpha:\beta}{\beta} \in D$ **Faça**
4: $P_0 \leftarrow P_0 \cup \{\delta\}$
5: **Fim Para**
6: **Para Todo** $\delta = \frac{\alpha:\beta \wedge \gamma}{\beta} \in D$ **Faça**
7: $i \leftarrow l_{MIN}(\beta, L)$
8: **Se** $max < i$ **Então**
9: $max \leftarrow i$
10: **Fim Se**
11: $P_i \leftarrow P_i \cup \{\delta\}$
12: **Fim Para**
13: **Para** i = 0 **Até** max **Faça**
14: $P \leftarrow \{P_i\}$
15: **Fim Para**
16: **Retorne** P

O algoritmo que implementa divisão dos *default* em partições (4.19) possui complexidade igual a EXPTIME. Essa complexidade vem da linha 2 que é igual a complexidade da função *length*. Os laços que compõem o algoritmo possuem complexidade igual a $O(n)$ cujo valor é inferior a complexidade da geração do conjunto L (linha 2).

Apresentamos a seguir dois corolários que surgem a partir do teorema (4.21).

Corolário 4.22. *(ETHERINGTON, 1988) Se $\delta \in P_0$, então δ é um default normal.*

Temos esta informação de forma intuitiva uma vez que *default* normais não introduzem nenhum conflito com os outros *default*. Não necessitando assim a divisão dos *default* em mais de uma partição.

Corolário 4.23. *(ETHERINGTON, 1988) Se $i>0$ e $P_i \neq \{\}$, então existe pelo menos um default semi-normal em P_i*

Este corolário veio abordar o fato de que os *default* semi-normais podem introduzir conflitos. Uma vez que conflitos podem existir entre pelo menos dois *default*, estes não podem pertencer a mesma partição. Disso, surge a necessidade de uma nova partição P_1 (supondo P_0 ser a partição que um dos *default* pertença). Finalizamos essa seção apresentando o algoritmo de pré-compilação da teoria *default* $\Delta = \langle T, A, D\rangle$.

Algoritmo 4.20 Pré-Compilação(A,T,D)

Entrada: Os conjuntos A,T,D

Saída: Se a teoria não possuir ciclos, um conjunto P com conjuntos P_i que dividem o conjunto D de *default* em partições

1: **Se** (Ordenação de *Defaults*(A,T,D) = **falso**) **Então**
2: **Retorne falso**
3: **Fim Se**
4: $\underline{\ll}, \ll \leftarrow$ Ordenação de *Defaults*(A,T,D)
5: $P \leftarrow$ Particões(A,T,D,$\underline{\ll}, \ll$)
6: **Retorne** P

A complexidade do **Algoritmo 4.20** é igual a maior complexidade dos passos descritos nessa seção que é igual a EXPTIME que é a complexidade dos **Algoritmos 4.9** (linhas 1 e 4) **e 4.19** (linha 5).

4.3 Uma Definição Construtiva de Extensão

Apresentaremos nesta seção uma forma construtiva de gerarmos uma extensão. Ela será apresentada em forma de teorema o qual foi adaptado para $\mathcal{ALC}$ a partir da prova do teorema 1 de (ETHERINGTON, 1988).

Teorema 4.24. ***(Extensão Construtiva)***

Seja $\Delta = \langle T, A, D\rangle$ uma teoria de descrição default ordenada e D_i uma partição default. Seja E_0 uma extensão para $\langle T, A, D_0\rangle$. Para $i > 0$, construímos:

$$D_i' = \{\tfrac{\alpha:\beta}{\beta} | \tfrac{\alpha:\beta}{\beta} \in D_i, \text{ ou } \tfrac{\alpha:\beta\wedge\gamma}{\beta} \in D_i, \text{ e } \neg\gamma \notin E_{i-1}\}$$

Uma vez que $\langle T, A, D_i'\rangle$ é uma teoria default normal, ela tem pelo menos uma extensão E_i.

Assim, $E = \bigcup_{i=0}^{\infty} E_i$ é uma extensão para $\langle T, A, D\rangle$.

Note que, no teorema 4.24, um *default* $\frac{\alpha:\beta\wedge\gamma}{\beta}$ é considerado exatamente no momento em que garantimos que sua condição de exceção $\neg\gamma$ não é mais derivavel. Isso ocorre devido a ordenação dos *defaults* e a partição dos *defaults*. Se um *default* δ pertence a partição D_i, isso ocorre por que $l_{MAX}(\neg\gamma) < l_{MIN}(\beta) = i$. Isso reflete o Princípio das Exceções Primeiro (PEQUENO, 1994) que nos diz que devemos primeiramente checar a condição de exceção antes de aplicarmos um *default*. Como dissemos anteriormente, esse princípio é a chave para a resolução do problema das extensões anômalas introduzido pelo "Yale Shooting Problem" (GINSBERG, 1987b), descrito na seção 3.3.

Desenvolveremos o algoritmo de construção de extensões baseado nesse teorema na seção a seguir.

4.4 A Construção de Extensões

Vejamos o algoritmo principal deste capítulo baseado nos teoremas 3.10 e 4.24: O Algoritmo de Construção de Extensões.

Algoritmo 4.21 Extension(A, T, D)

Entrada: A teoria de descrição *default* $\langle A, T, D\rangle$

Saída: O Conjunto $A \cup T \cup \{$ as conclusões dos *defaults* aplicáveis na extensão E definida nos teoremas 3.10 e 4.24

1: **Se** (Pré-Compilação(A,T,D) = **falso**) **Então**
2: **Retorne** "Teoria Incoerente"
3: **Fim Se**
4: $P \leftarrow$ Pré-Compilação(A,T,D)
5: Conclusoes $\leftarrow \varnothing$
6: **Para Todo** $\delta = \frac{\alpha:\beta}{\beta} \in P_0$ **Faça**
7: **Se** $(A \cup T \vdash \alpha)$ AND $(A \cup T \nvdash \neg\beta)$ **Então**
8: Conclusoes $\leftarrow$ Conclusoes $\cup\{\beta\}$
9: **Fim Se**
10: **Fim Para**
11: $E_0 \leftarrow A \cup T \cup$ Conclusoes
12: **Para** i := 1 **Até** $|\{P_i\}| - 1$ **Faça**
13: Conclusoes $\leftarrow \varnothing$
14: **Para Todo** $\delta = \frac{\alpha:\beta,\gamma}{\beta} \in P_i$ **Faça**
15: **Se** $(E_{i-1} \vdash \alpha)$ **E** $(E_{i-1} \nvdash \neg\beta)$ AND $(E_{i-1} \nvdash \neg\gamma)$ **Então**
16: Conclusoes $\leftarrow$ Conclusoes $\cup\{\beta\}$
17: **Fim Se**
18: **Fim Para**
19: $E_i \leftarrow E_{i-1} \cup$ Conclusoes
20: **Fim Para**
21: **Retorne** $E_{|\{P_i\}|-1}$

A corretude e completude do **Algoritmo 4.21** são garantidos uma vez que ele implementa os teoremas 3.10 e 4.24. A cada passo i, testamos a aplicabilidade dos *defaults* $\delta \in P_i$ (note que P_i se refere a partição i) respeitando o Princípio das Exceções Primeiro já explicado anteriormente.

A terminação é obtida devido os conjuntos A, T e D serem finitos e os algoritmos que compõem a pré-compilação, descrita na seção 4.2, a provabilidade ($\vdash$) e a não-provabilidade ($\nvdash$) são todos computáveis (recursivos).

A entrada do **Algoritmo 4.21** é a teoria de descrição *default* $\langle A, T, D \rangle$.

O procedimento de Pré-compilação, utilizado nas linhas 1 e 4, possui complexidade EXPTIME. Para continuarmos a análise de complexidade do **Algoritmo 4.21**, precisamos analisar a complexidade dos testes da provabilidade ($E \vdash \alpha$) e da não-provabilidade ($E \nvdash \alpha$) os quais ocorrem nas linhas 7 e 15.

Para implementarmos esses procedimentos, utilizamos o algoritmo de *tableaux* de complexidade EXPTIME apresentado na seção 2.6.2.

Nessa seção, apresentamos um algoritmo de *tableaux* para checagem de satisfatibilidade de um conceito com respeito a um Tbox com axiomas de inclusão geral que possui complexidade exponencial simples (EXPTIME). Utilizando esse método de *tableaux* para testarmos a satisfatibilidade de um conceito com respeito a um Tbox, devemos verificar todos os ramos do *tableaux* resultante no pior caso. Diante desse fato, podemos concluir que o teste de satisfatibilidade é equivalente ao teste de insatisfatibilidade.

A prova da terminação desse algoritmo é provada em (DONINI; MASSACCI, 2000).

Conforme descrito em (DONINI; MASSACCI, 2000), o tableaux apresentado pode ser facilmente modificado para lidar também com Abox. Além disso, afirmou-se que "a satisfatibilidade de conceitos com repeito a um Tbox pode resolver outros problemas tais como o acarretamento". De fato, $E \vDash \alpha$ se e somente se $E \cup \{\neg\alpha\}$ é insatisfatível. Em termos equivalentes, $E \vdash \alpha$ se e somente se $E \cup \{\neg\alpha\}$ é inconsistente. Pela contrapositiva, $E \nvdash \alpha$ se e somente se $E \cup \{\neg\alpha\}$ é consistente se e somente se $E \cup \{\neg\alpha\}$ é satisfatível. Assim, esse algoritmo de *tableaux* pode ser perfeitamente utilizado para a implementação da provabilidade e da não-provabilidade.

Podemos assim, concluir que a complexidade de nosso Algoritmo de Construção de Extensões de uma Teoria de Descrição *Default* (**Algoritmo 4.21**) possui complexidade igual a complexidade dos algoritmos de Pré-compilação, provabilidade e não-provabilidade, ou seja, exponencial simples (EXPTIME).

5 CONCLUSÃO

Neste capítulo, abordaremos as conclusões alcançadas nessa dissertação. Apresentaremos as dificuldades encontradas ao longo de seu desenvolvimento. Abordaremos uma comparação entre essa dissertação e outros trabalhos relacionados. Finalizaremos apresentando alguns possíveis trabalhos futuros.

5.1 Conclusões

Nosso objetivo de apresentar e desenvolver um sistema de representação de conhecimento não-monotônico que seja aplicável a *Web* e que suas inferências sejam decidíveis foi alcançado.

5.2 Comparação com Trabalhos Relacionados e Trabalhos Futuros

Em (BAADER et al., 2003), os trabalhos relacionados são discutidos. Existem algumas diferenças entre nossa abordagem e os trabalhos lá apresentados:

- Usamos não somente *default* normais, mas também os semi-normais.
- Em nossa regra *default* $\frac{\alpha:\beta\wedge\gamma}{\beta}$, α, β, γ podem ser qualquer fórmula da Lógica de Descrição, enquanto que na Lógica *Default* original, podem ser quaisquer fórmulas de FOL.
- O processo de pré-compilação, descrito na seção 4.2, garante que construímos a extensão desejada devido a ordem de aplicação dos *defaults*, do mais específico para o menos específico. A ordem entre os *defaults* não reflete uma organização hierárquica dos conceitos, entretanto ela também funciona com qualquer conjunto de axiomas terminológicos uma vez que a ordem é baseada na derivabilidade.
- Algumas desvantagens de nosso trabalho são:
  - O raciocínio *default* é reduzido a indivíduos explícitos que ocorrem no Abox;

– A complexidade exponencial obtida no algoritmo de pré-compilação devido ao algoritmo de transformação das fórmulas para a Forma Normal Clausal (veja seção 4.2.1)

Intencionamos como trabalhos futuros eliminar o primeiro problema.

Além disso, procuraremos reduzir a complexidade para polinomial do algoritmo de Transformação Clausal. Para tal propósito, podemos utilizar os algoritmos polinomiais descritos em (JACKSON; SHERIDAN, 2004). O problema desses algoritmos é que eles não preservam a equivalência entre as fórmulas original e transformada. Tal fato pode implicar na alteração do cálculo de ordenação dos *defaults* (descrita na seção 4.2.2) processo essencial no processo de construção de extensões apresentado nessa dissertação.

Ainda como trabalho futuro, pode-se buscar outros algoritmos de Transformação Clausal mais eficientes e ajustar suas saídas para nosso problema.

Finalmente, outro possível trabalho futuro é utilizarmos uma Lógica de Descrição menos expressiva, como a $\mathcal{FL}$ (BAADER et al., 2003), e realizarmos toda a análise apresentada nessa dissertação. Ao final, poderemos verificar se houve redução na complexidade dos algoritmos aqui apresentados.

REFERÊNCIAS

ANTONIOU, G.; HARMELEN, F. van. *A Semantic Web Primer*. third. Cambridge, Mass.: MIT, 2004. 238 p.

BAADER, F. et al. *The Description Logic Handbook: Theory, Implementation and Applications*. [S.l.]: Cambridge University Press, 2003. ISBN 0521781760.

BASSILIADES, N.; ANTONIOU, G.; VLAHAVAS, I. P. A defeasible logic reasoner for the semantic web. *Int. J Semantic Web Inf Syst*, v. 2, n. 1, p. 1–41, 2006.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web: Scientific American. *Scientific American*, maio 2001. Disponível em: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&pageNumber=1&catID=2>.

BÖRGER, E.; GRÄDEL, E.; GUREVICH, Y. *The Classical Decision Problem*. [S.l.: s.n.], 1997. (Perspectives in Mathematical Logic).

DALEN, D. van. *Logic and Structure*. 3. ed. [S.l.]: Springer-Verlag, 1994.

DESCRIPTION LOGIC REASONERS. "`http://www.cs.man.ac.uk/~sattler/reasoners.html`".

DONINI, F. M.; MASSACCI, F. Exptime tableaux for alc. *ARTIFICIAL INTELLIGENCE*, v. 124, n. 1, p. 87–138, 2000.

EBBINGHAUS et al. *Mathematical logic, Undergraduate Text in Mathematics*. Berlin/New York: Springer-Verlag, 1994.

ENDERTON, H. B. *A Mathematical Introduction to Logic*. 2nd. ed. [S.l.]: Harcourt Academic Press, 2001.

ETHERINGTON, D. W. Finite default theories. m.sc. thesis. In: . [S.l.: s.n.], 1982.

ETHERINGTON, D. W. *Reasoning with Incomplete Information*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988. ISBN 0934613605.

FACT. "`http://www.racer-systems.com/products/racerpro/index.phtml`".

GINSBERG, M. L. (Ed.). *Readings in nonmonotonic reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987. ISBN 0-934613-45-1.

GINSBERG, M. L. (Ed.). *Readings in nonmonotonic reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987. ISBN 0-934613-45-1.

JACKSON, P.; SHERIDAN, D. Clause form conversions for boolean circuits. In: *In Theory and Appl. of Sat. Testing, 7th Int. Conf. (SAT04), volume 3542 of LNCS*. [S.l.]: Springer, 2004. p. 183–198.

LEWIS, H. R.; PAPADIMITRIOU, C. H. *Elements of the Theory of Computation*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997. ISBN 0132624788.

LUKASZEWICZ, W. *Non-Monotonic Reasoning: Formalization of Commonsense Reasoning*. New York: Ellis Horwood, 1990. ISBN 0136244467.

MARTINS, A. T. *A Syntactical and Semantical Uniform and Treatment for Idl and Lei Nonmonotonic System*. 1997. PhD Thesis. Department of Informatics, Federal University of Pernambuco.

MCCARTHY, J. Circumscription – A Form of Nonmonotonic Reasoning. *Artificial Intelligence*, v. 13 (1-2), p. 27–39, 1980.

PEQUENO, M. C. *University of London Defeasible Reasoning with Exceptions First*. 1994.

PROTéGé. "`http://protege.stanford.edu/`".

R, R. A logic for default reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 68–93, 1987.

RACER. "`http://www.racer-systems.com/products/racerpro/index.phtml`".

SCHMIDT-SCHAUBSS, M.; SMOLKA, G. Attributive concept descriptions with complements. *Artif. Intell.*, Elsevier Science Publishers Ltd., Essex, UK, v. 48, p. 1–26, February 1991. ISSN 0004-3702. Disponível em: <http://portal.acm.org/citation.cfm?id=114341.114342>.

SIRIN, E. et al. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, v. 5, n. 2, p. 51–53, 2007.

SMULLYAN, R. *First-order logic*. [S.l.]: Dover Publications, 1995.

SOWA, J. F. Semantic Networks. In: SHAPIRO, S. C. (Ed.). *Encyclopedia of Artificial Intelligence*. Second. Wiley, 1992. Disponível em: <http://www.jfsowa.com/pubs/semnet.htm>.