Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Luís Felipe de Melo Costa Silva

Revisão de Crenças em Lógica de Descrição

São Paulo
Novembro de 2018

Revisão de Crenças em Lógica de Descrição

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisora: Profª. Drª. Renata Wassermann

São Paulo
Novembro de 2018

Resumo

As ontologias são sistemas usados para representar conhecimento de algum domínio, como a saúde ou o cinema. Elas são baseadas em classes, propriedades das classes e relações entre elas. As Lógicas de Descrição, que são sublinguagens da Lógica de Primeira Ordem, podem ser usadas para formalizar as ontologias.

Pode acontecer, em dado momento, que a inclusão de algum novo conhecimento torne a base de dados já existente inconsistente, ou seja, o novo conhecimento entra em conflito com algum que já estava lá. Nestes casos, é preciso reparar a ontologia, restaurando a sua consistência. Técnicas de Revisão de Crenças podem ser usadas para isso.

Um sistema utilizado para construir e usar ontologias é o *Protégé*. Com ele se definem as classes, axiomas e relações da ontologia. Com o auxílio de *plug-ins*, é possível fazer inferências e até encontrar as inconsistências.

Nesse trabalho é feito um estudo teórico que vai desde as origens das ontologias, passando pelas Lógicas de Descrição e chegando, enfim, à área de Revisão de Crenças. Além disso, a partir de um *plug-in* construído e de diversas outras implementações, é construído um programa que implementa algumas operações de Revisão de Crenças.

Palavras-chave: Revisão de Crenças, OWL, Lógicas de Descrição, Ontologias, Representação de Conhecimento.

Abstract

ONTOLOGIES are systems used to store knowledge from some domain, such as health or film. They are based in classes, their properties and the relationships among them. The Description Logics, subset of First Order Logics, are used to represent ontologies, since they have the formalism needed for it.

The inclusion of a new information, that is inconsistent with the knowledge base, is something very common to happen. In such cases, the new knowledge conflicts with the old knowledge. The repair of the ontology becomes something necessary, in order to restore its consistency. Belief Revision techniques are used to achieve that.

A system used to build and use ontologies is called Protégé. One can define classes, axioms and relationships of a ontology with it. With the help of some plug-ins, some inferences can be made, and even inconsistencies can be found.

On this work, a theorical study is made, starting with the ontologies' origin, passing through Description Logics and ending with Belief Revision. Furthermore, using an alrready build plug-in and several other implementations, a program that executes some Belief Revision operations was made.

Keywords: Belief Revision, OWL, Description Logics, Ontologies, Knowlege Representation.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

A obtenção de novos conhecimentos é inerente a todo agente inteligente. Nós, seres humanos, sempre estamos em busca de aprender mais. Nesse processo, alguns velhos conhecimentos podem ser desconsiderados. Além disso, pode acontecer de uma nova informação entrar em conflito com o que já se conhece, causando uma reflexão.

O mesmo pode ser aplicado para os sistemas baseados em conhecimento. Esse ó o objetivo da área de Revisão de Crenças que possui, como seu marco inicial, o artigo *On the Logic of Theory Change: Partial Meet Contraction and Revision Functions* (Alchourrón *et al.*, 1985). Ele apresenta três operações básicas: a Expansão, equivalente a receber um novo conhecimento; a Contração, análoga à apagar uma informação; e a Revisão, correspondente à adição de um novo conhecimento e com o cuidado de tirar tudo aquilo que pode entrar em conflito.

Para que os sistemas computacionais possam passar por uma revisão de suas crenças, é necessário que haja alguma codificação do conhecimento e do sistema. Para tal são usadas as ontologias, que são uma reunião de axiomas que modelam certa parte de um domínio do conhecimento. As ontologias são compostas por sentenças em Lógicas de Descrição, por conta do formalismo que essas últimas possuem.

Existe um padrão para a representação de Ontologias. Ele é conhecido por OWL (*Web Ontology Language*) (W3C, 2012a). Possui diversas linguagens, que acompanham sublinguagens equivalentes de Lógicas de Descrição. Para tornar a construção e uso de ontologias mais fácil, foi desenvolvido o *Protégé* [1], pelo *Stanford Center for Biomedical Informatics Research*, da Universidade de Stanford. A integração com diversos *plug-ins* o tornou muito popular.

Graças a diversos estudos, foi construído um *plug-in* que possui quatro operações: a Contração *Kernel* (Hansson, 1994), a Contração *Partial Meet* (Alchourrón *et al.*, 1985), a Pseudocontração SRW (Santos *et al.*, 2015) e a Revisão *Kernel* (Ribeiro e Wassermann, 2008). Fo-

[1] http://protege.stanford.edu/

ram feitos testes para ver o comportamento desses *plug-ins*, frente a diferentes ontologias.

1.2 Estrutura do trabalho

O capítulo 2 mostra estudos na área de Ontologias, passando por definições teóricas, seus componentes, sua montagem e usabilidade. No capítulo 3 é feito um estudo sobre as Lógicas de Descrição, fazendo um rápido paralelo com as ontologias, comentando sobre possíveis linguagens e escrevendo sobre interpretação, consequência lógica e consistênicia. O capítulo 4 apresenta as operações de Revisão de Crenças desde sua motivação, passa pelas operações clássicas e termina com alguns construtores. Esses três capítulos compõem a parte teórica desse trabalho.

A parte prática é composta por três capítulos. O capítulo 5 exibe um rápido panorama de como as Ontologias são representadas computacionalmente, focando-se na linguagem OWL e no programa *Protégé*. No capítulo 6 é descrita a implementação do *plug-in* de Revisão de Crenças prometido para este trabalho. Por fim, o capítulo 7 mostra testes de performance e de comparação do *plug-in* implementado.

No final do trabalho, há uma seção de agradecimentos e de observações quanto ao trabalho.

Capítulo 2

Ontologias

A palavra Ontologia veio do grego, assim como vários outros termos que se referem a alguma área de estudo. Seu significado, no entanto, é muito mais abstrato. Diferentemente de Biologia, que é "o estudo da vida", a palavra cujo plural dá nome a este capítulo quer dizer "o estudo do ser enquanto ser". O dicionário Merriam-Webster [1] estende essa definição como: "um ramo da metafísica preocupado com a natureza e as relações do ser".

Importado da Filosofia, esse conceito começa a ser trabalhado muito antes da época dos computadores. Aristóteles já estudava Ontologia em suas Categorias (Dahlberg, 1978). No entanto, apenas em 1606, com o livro *Ogdoas Scholastica*, de Jacob Lorhard, foi que a palavra em si realmente surgiu. Esse termo ficou popular em 1729 com o livro *Philosophia Prima: sive Ontologia*, de Christian Wolff, com a definição "*Ontology or First Philosophy is the science of Being in general or as Being*"[2] (Nickles *et al.*, 2007).

Para a Ciência da Computação, a definição é um pouco diferente, embora possua muita semelhança com o conceito já explícito. Guarino definiu ontologia como "um artefato de engenharia, constituído por um vocabulário específico usado para descrever uma certa realidade, mais uma série de pressupostos explícitos acerca do significado que se atribui a esse vocabulário"(Guarino, 1998). Logo, uma ontologia é uma reunião de sentenças lógicas que exibem alguma informação sobre alguma área do mundo para resolver algum problema relacionado a ela.

Fazendo um paralelo entre ambas as disciplinas, pode-se observar que, enquanto a primeira faz um estudo sistemático da existência, na segunda existe um foco maior no que pode ser representado.

As ontologias são estudadas na área de Inteligência Artificial, que está preocupada com a automação do comportamento inteligente. Na prática, elas funcionam como um sistema "*tell and ask*" Russel e Norvig (1995). Algumas coisas são contadas para os agentes inteligentes (uma entidade autônoma com comportamento que simula inteligência), e então, perguntas podem ser feitas para eles, embora não precisem saber todas as respostas.

[1] http://www.merriam-webster.com

[2] "Ontologia ou Filosofia Primeira é a ciência do ser em real ou como um ser."

Surgiram de um contexto onde os cientistas desejam modelar e representar o mundo para as máquinas, e isso ocorre desde a origem dos computadores. Como cada pessoa possui uma visão de mundo, cada modelagem será diferente de algum jeito. Por isso, a construção de ontologias é um tópico que merece estudo.

As ontologias denotam uma ”especificação explícita de uma conceitualização”(Gruber, 1995), e, uma vez construídas, permitem comunicação, compartilhamento e reúso de conhecimentos.

2.1 Conceitualização

Existe um certo debate sobre a definição de Conceitualização. Depois de propor o que era uma ontologia, Gruber sugeriu que uma Conceitualização ”é uma visão abstrata e simplificada do mundo que se quer representar para algum propósito”. Essa definição parece boa, mas deixa algumas pontas soltas. O que seria uma ”visão”, por exemplo, deixa algumas dúvidas.

Para Guarino e Giaretta, uma Conceitualização pode ser entendida como “uma estrutura semântica intensional que codifica as regras implícitas que determinam a estrutura de uma porção da realidade” (Giaretta e Guarino, 1995). Essa definição é um pouco mais concreta, e já é possível pensar sobre ela computacionalmente.

Pode-se inferir que uma Conceitualização é uma modelagem de parte de algum domínio do conhecimento. O domínio nada mais seria do que alguma disciplina, como a Geografia, Música, Enologia, entre outros. Tal modelagem é feita a partir de alguma linguagem formal de representação (em geral, Lógicas de Descrição) e deve levar em conta a generalidade que se aplica ao domínio escolhido.

Vale lembrar que, embora sejam amplamente utilizadas na vida real, as linguagens naturais (como a língua portuguesa e a lingua inglesa) não são consideradas linguagens formais. Isso tem uma explicação simples. Basta lembrar das figuras de linguagem, como a metáfora e o eufemismo, amplamente usadas nos discursos escrito e falado.

2.2 Construindo uma ontologia

Para construir uma ontologia, é necessário escolher um domínio e o nível de generalidade que é necessário que ela atinja. Também deve-se ter em mente quem vai usá-la. Para que ela alcance o máximo de utilidade, é necessário que as perguntas que se deseja que ela responda sejam feitas antes de sua construção.

Geralmente, grandes ontologias são projetadas por equipes interdisciplinares, para que ela seja o mais correta e abrangente quanto possível. Quando se confecciona uma Ontologia, é necessário que sejam feitas algumas decisões de projeto.

Gruber fez uma proposta de critérios de *design* para ontologias com o objetivo de tornar o compartilhamento de conhecimento e interoperabilidade com programas baseados em conhecimento mais fácil (Gruber, 1995). Eles são os seguintes:

Clareza Uma ontologia deve ter uma linguagem clara e efetiva na definição de seus termos. Tal definição deve ser objetiva. Embora ela possa vir de situações sociais ou requisitos computacionais, ela deve ser independente destes contextos. Usar uma linguagem lógica, é um meio para este fim, ou seja, quando for possível fazer uma definição usando axiomas lógicos, isso deve ser feito. Onde possível, uma definição completa (com condições necessárias e suficientes) é preferível a uma definição parcial (com condições necessárias ou suficientes). Todas devem ser documentadas usando linguagem natural.

Coesão Uma ontologia deve permitir inferências consistentes com suas definições. No mínimo, os axiomas usados nas definições devem ser logicamente consistentes. A coerência também deve ser aplicada aos conceitos informais, definidos na linguagem natural da documentação e nos exemplos. Se uma sentença que pode ser inferida contradiz uma definição ou exemplo dado informalmente, a ontologia não é coesa.

Estendibilidade Uma ontologia deve ser projetada para ser capaz de antecipar o uso de conhecimento compartilhado. Ela deve oferecer uma fundação conceitual de modo que o novo conhecimento possa ser apoiado nela e que a ontologia possa ser estendida e especializada, ou seja, novos termos podem ser definidos usando o vocabulário existente, de modo que a revisão das crenças do conhecimento anterior possa ser evitada ao máximo.

Viés mínimo de codificação A Conceitualização deve ser especificada num nível de conhecimento que não dependa de uma codificação que utiliza um nível de símbolos particulares. Um viés de codificação ocorre quando as escolhas de representação são feitas por pura conveniência de notação ou de implementação. Como agentes de conhecimento podem ser implementados em diferentes sistemas e estilos de representação, isso deve ser minimizado.

Compromisso ontológico mínimo Isso faz com que ela suporte as atividades de compartilhamento de conhecimento desejadas. Quanto menos suposições sobre o mundo modelado, melhor. Isso permite que as partes comprometidas com a ontologia sejam livres para especializar e instanciar a ontologia o quanto quiserem. Já que isso é baseado no uso consistente do vocabulário, ele pode ser minimizado usando uma teoria fraca (genérica) e definindo apenas os conceitos necessários para a comunicação do conhecimento consistente com esta teoria.

É possível notar que todos esses critérios poderão não ser atendidos ao mesmo tempo, portanto, alguns *trade-offs* deverão ser feitos. Podemos ter um conflito, por exemplo, entre

os critérios **Clareza** e **Estendibilidade**, já que o máximo de clareza implica que as definições terão a sua interpretação restrita.

2.2.1 Componentes de uma ontologia

Computacionalmente falando, as ontologias possuem cinco partes. Para ilustrá-las melhor, vamos fazer uma ontologia sobre Música. Será usada uma fonte monoespaçada para as partes dela.

Classes Descrevem os conceitos de um certo domínio do discurso. São o foco das ontologias. São uma coleção de todos os particulares aos quais é possível aplicar um termo geral. Por exemplo: a classe `Cantor`.

Propriedades São atributos que descrevem características do conceito a que uma classe se refere. Em nosso exemplo, a classe `Pessoa` possui as propriedades `Nome` e `Idade`. A classe `Cantor` pode ter a propriedade `RitmoPredominante`.

Relações Ontologias são constituídas de relações hierárquicas. Por exemplo `Cantor` $\rightarrow$ `Pessoa`. A hierarquia de classes representa uma relação "*is-a*" (França, 2009). Tais relações são transitivas. Existem outras relações, como `Cantor canta Musica` e `Compositor compoe Musica`.

Restrições São os tipos das propriedades, por exemplo: enquanto para `Nome` e `RitmoPredominante` uma *string* seja suficiente, para a Idade, um inteiro já está de bom tamanho. Além disso, as restrições podem ser referentes às classes. Exemplo: a releção `canta` tem domínio `Cantor`.

Instância é um indivíduo (por exemplo, `MariahCarey`), de uma classe (como `Cantor`, `Compositor` ou `Pessoa`). Vale lembrar que uma instância não é uma subclasse.

Uma classe pode ter nomes diferentes em certo idiomas (por exemplo, `Cantor` e `Singer` seriam a mesma classe). Isso não é um problema pois não é o nome que define uma classe. Sinônimos e palavras em línguas distintas não representam classes diferentes.

Várias classes subordinadas a uma superclasse são consideradas irmãs. Elas devem ter o mesmo nível de generalidade. Por exemplo, seja uma classe `Musica`. Suas subclasses podem ser `Cancao`, `MusicaArgentina` e `MusicaBrasileira`. Tais classes são irmãs.

Existem três processos para definir as classes e a hierarquia, a seguir:

Top-down Vai das classes mais genéricas para as mais específicas. Um exemplo seria criar as classes `Musica`, `Pessoa`, entre outras.

Bottom-up Vai das classes mais específicas para as mais genéricas, agrupando as específicas já criadas. Na ontologia estudada aqui, é possível começar das músicas já conhecidas para então separá-las por ritmos.

Vai-e-Vem Define conceitos simples para generalizá-los e especificá-los. É uma combinação dos dois primeiros itens.

2.2.2 Montando a ontologia

Montar uma ontologia é um processo que segue os seguintes passos:

- Definir as classes da ontologia.
- Colocá-las em uma hierarquia taxonômica.
- Determinar suas propriedades e restrições.
- Criar uma base de conhecimento para essas classes e propriedades, ou seja, preencher a ontologia com as instâncias.
- Colocar os valores das propriedades para as instâncias.

Embora pareça ser direto, esse processo é iterativo, como afirmam Noy e McGuinness (Noy *et al.*, 2001). Uma vez feito, deve ser repetido para que haja uma adequação das classes com as instâncias colocadas, pois, por exemplo, se uma classe acabar com apenas uma subclasse, a modelagem pode ter um problema, ou a ontologia não está completa. E ainda, se uma classe possui mais de uma dúzia de subclasses, novas categorias (classes ou subclasses) podem ser necessárias.

Às vezes, uma classe possui muitas propriedades específicas e diferentes em várias de suas instâncias. Nesse caso, a inserção de uma classe deixará a ontologia mais compreensível. A nova classe fará com que a distinção das instâncias ocorra, efetivamente, evitando mal-entendidos.

Teoricamente, o processo nunca acaba. Na prática, ele é interrompido quando a ontologia fornece respostas suficientemente boas para a maior parte das consultas realizadas, ou seja, tal critério é subjetivo.

Ainda em relação às classes, algumas observações podem ser feitas. A primeira é em relação à herança múltipla. Ela acontece quando uma classe é subclasse de várias outras classes, por exemplo, a classe `PopBrasileiro`, pode pertencer à classe `Pop` e à classe `MusicaBrasileira`. Isso é aceitável, pois no mundo real, acontece várias vezes e de diversas maneiras.

As ontologias podem representar classes disjuntas. Classes disjuntas são aquelas que não possuem uma intersecção. Em nossa ontologia, `Cantor` e `Compositor` não são disjuntas. No entanto, `Cancao` e `HinoNacional` são classes disjuntas.

Para os nomes das classes, não há uma convenção específica, só há um consenso de que manter um padrão de nomenclatura é algo bom. Para fazer isso, pode-se usar `snake_case`, `camelCase` ou usar espaços na grafia.

Uma ontologia não precisa ter toda a informação existente sobre o domínio. Não é necessário especializar ou generalizar mais do que seja necessário para a aplicação. Além disso, as classes não precisam ter todas as propriedades possíveis e nem carregar todas as distinções que estão no mundo. Isso significa que a ontologia deve ser o modelo mais simples para o problema que se deseja resolver.

Algumas relações podem ter uma inversa, assim como ocorre com funções matemáticas. Uma relação que possui uma inversa pode ser `Cantor canta Musica`, com sua inversa sendo `Musica cantadaPor Cantor`, e a relação `Compositor compoe Musica` pode ter a inversa `Musica compostaPor Compositor`.

2.3 Usabilidade de uma ontologia

A criação de uma ontologia é feita por uma equipe interdisciplinar, geralmente composta por *experts* da área que se deseja cobrir e técnicos para a confecção de ontologias (vindos da área de Computação). Isso não impede a equipe de possuir profissionais de mais áreas. Uma ontologia é feita para que qualquer pessoa possa acessar suas informações.

Ontologias feitas sobre áreas de estudo (por exemplo, a Aviação) são muito mais úteis do que aquelas feitas sobre acontecimentos (como a queda do voo TAM 3054). A existência dessa última nem faz sentido, já que as ontologias servem para modelar casos gerais, não particulares. É melhor, por exemplo, construir uma ontologia sobre acidentes de avião e seus conceitos.

2.4 Problemas relacionados

Existem alguns problemas relacionados a ontologias. Os mais comuns são os problemas de modelagem e construção. Um problema muito comum é a existência, na linguagem natural, de Homônimos e Sinônimos. Deve existir um cuidado especial com eles, já que eles podem levar a uma confusão na nomenclatura.

Em relação à modelagem, o problema de não utilizar uma equipe interdisciplinar especializada pode levar a uma cobertura incompatível de conceitos, ou seja, as classes podem ficar muito distantes da realidade. Além disso, usar fontes não confiáveis na ontologia pode fazer com que os problemas que elas estavam sendo feitas para resolver, não sejam resolvidos corretamente.

Tais problemas tornam-se muito maiores quando duas ontologias são integradas. A possível integração entre ontologias é um dos motivos para elas existirem, afinal, isso pode acelerar o seu desenvolvimento. Em nosso exemplo, se já existir uma ontologia sobre Música Brasileira, poderemos absorvê-la, mas o cuidado terá de ser redobrado em relação às questões acima.

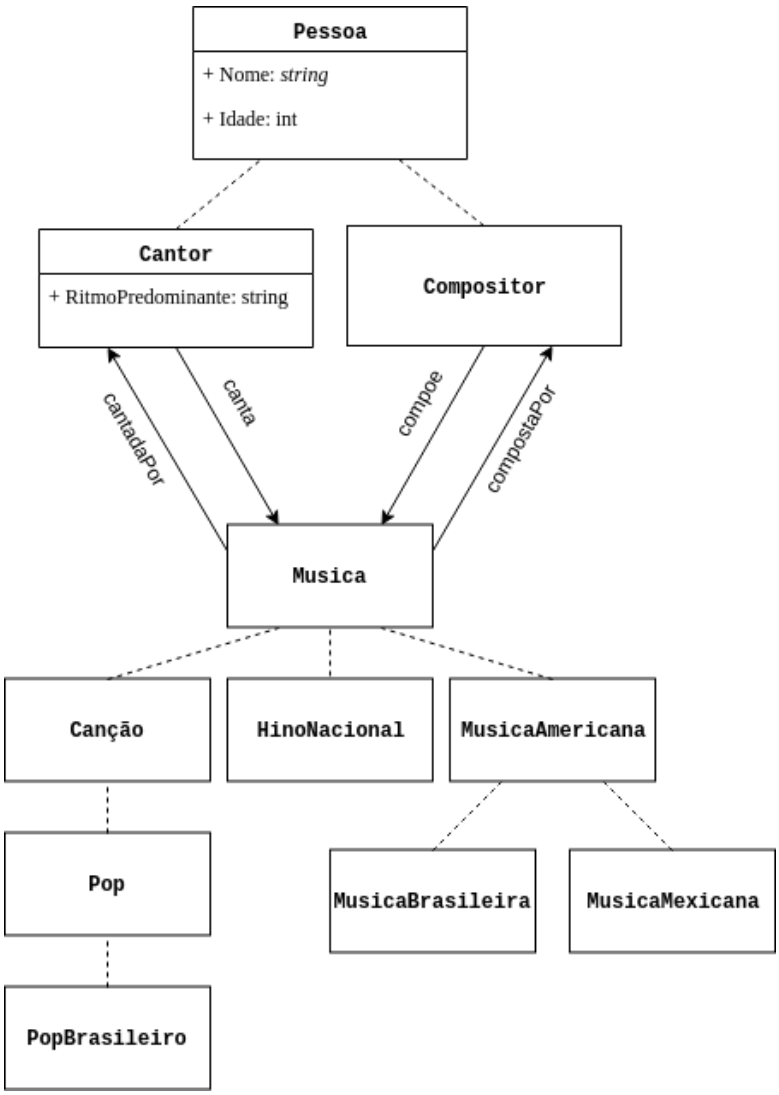


Figura 2.1: Esquema de classes de uma ontologia sobre Música. As linhas tracejadas indicam as subclasses e as setas, as relações.

Em relação à implementação, os pontos que surgem são em geral a respeito da linguagem utilizada, mas isso será tratado no Capítulo 3.

Além dos já citados, pode acontecer de chegar um novo conhecimento e a ontologia ficar inconsistente. Neste caso, terão de ser usadas algumas operações de Revisão de Crenças, foco deste trabalho, que será estudado no Capítulo 4.

A ontologia desenvolvida nesse capítulo segue na Figura 2.1

Capítulo 3

Lógicas de Descrição

A área de Inteligência Artificial é composta por diversas subáreas. Uma delas é a de Representação de Conhecimento, que cuida de construir formalismos adequados para expressar conhecimento sobre um domínio. Tal área estuda entidades inteligentes, que possuem algum tipo de conhecimento e precisam fazer inferências a partir dele.

Existe um consenso, já mencionado no Capítulo 2, de que linguagens formais são uma maneira boa de caracterizar axiomas lógicos para a modelagem de uma ontologia. O principal formalismo utilizado para representar conhecimento, com um cuidado especial para as terminologias, são as Lógicas de Descrição (doravante, LD).

Essas lógicas são um subconjunto da Lógica de Primeira Ordem (a partir de agora, LPO), que por sua vez, estende a Lógica Proposicional. As LD são utilizadas por sua expressividade. Para anotar que uma `Cancao` ou um `HinoNacional` são uma `Musica`, usando a ontologia que está sendo desenvolvida neste trabalho, podem ser utilizadas as seguintes sentenças, usando as lógicas até agora citadas, na Tabela 3.1.

Lógica	**Sentença**
Proposicional	$c \vee h \rightarrow m$
de Primeira Ordem	$\forall x(\texttt{Cancao}(x) \vee \texttt{HinoNacional}(x) \rightarrow \texttt{Musica}(x))$
de Descrição ($\mathcal{ALC}$)	$\texttt{Cancao} \sqcup \texttt{HinoNacional} \sqsubseteq \texttt{Musica}$

Tabela 3.1: A mesma sentença expressa em diferentes lógicas

Pode-se observar que a notação da LD permite que o entendimento de suas sentenças seja feito usando a Teoria dos Conjuntos. A última sentença da tabela leva a entender que a "união" dos "conjuntos" `Cancao` e `HinoNacional` está contida no "conjunto" `Musica`.

No entanto, é necessário certo cuidado com essa compreensão. Os conceitos, na verdade, são mapeados a conjuntos. Esses, por sua vez, são subconjuntos de um domínio. Quem faz o mapeamento é uma função de interpretação. As definições necessárias para o entendimento desse parágrafo estão diluídas pelo capítulo.

3.1 Conceitos, papéis e indivíduos

As LDs possuem um jeito próprio de organizar o conhecimento. Elas usam três definições para retratar o que é desejado (Van Harmelen *et al.*, 2008). Cada uma delas é utilizada na construção de uma ontologia. São, a seguir:

Conceitos Também chamados de classes, representam um conjunto de indivíduos. Nas ontologias, são equivalentes às Classes.

Papéis Podem ser denotados como propriedades. Retratam as relações binárias que existem entre os indivíduos. Basicamente, são as Relações de uma ontologia. Alguns papéis podem ter uma função de caracterização, ou seja, definindo alguns atributos para o conceito. Tal função corresponde a uma Propriedade de uma Classe, em uma ontologia.

Indivíduos Representam os particulares que existem dentro de um Conceito. Para as ontologias, são as Instâncias.

Logo, uma LD é definida por uma tripla (N_C, N_R, N_I), onde N_C é um conjunto de Conceitos atômicos, N_R é um conjunto de Papéis atômicos e N_I é um conjunto de nomes de indivíduos.

Feita essa distinção, o conhecimento em uma ontologia é dividido em duas partes, com as LD. A primeira delas é a *TBox*. Ela se refere ao conhecimento terminológico, ou seja, o conhecimento intensional do domínio. Nela ficam os conceitos, as propriedades e as restrições.

A outra parte é a *ABox*, e nela fica o conhecimento assertivo, ou seja, o conhecimento extensional. Ela contém as asserções sobre as instâncias, ou seja, como eles se encaixam nas definições explícitas na *TBox*.

3.2 $\mathcal{ALC}$ e outras linguagens

As pesquisas na área de Representação de Conhecimento levaram à confecção das chamadas Linguagens Terminológicas de Representação. Uma delas é a KL-ONE de Brachman Brachman e Schmolze (1988), que possui uma semântica clara, além de separar o conhecimento assertivo e terminológico.

Tais estudos levaram ao desenvolvimento da $\mathcal{ALC}$ Schmidt-Schauß e Smolka (1991). Essa linguagem, cuja sigla significa *Attributive Concept Language with Complements*, é uma das Lógicas de Descrição mais básicas que existem, embora seja uma das mais utilizadas para as atividades que envolvem raciocínio lógico.

Ela contempla os seguintes construtores, para quaisquer conceitos `A` e `B` e qualquer papel `r`, como visto na Tabela 3.2.

Nome	**Notação**
Conceito Universal	$\top$
Conceito Vazio	$\bot$
Conceito Atômico	$\mathtt{A}$
Negação	$\neg\mathtt{A}$
União	$\mathtt{A} \sqcup \mathtt{B}$
Intersecção	$\mathtt{A} \sqcap \mathtt{B}$
Universal	$\forall\, \mathtt{r}.\mathtt{A}$
Existencial	$\exists\, \mathtt{r}.\mathtt{A}$

Tabela 3.2: Construtores da Lógica $\mathcal{ALC}$

Essa representação também permite que sejam escritos axiomas terminológicos, tais como a subsunção, denotada por $\mathtt{A} \sqsubseteq \mathtt{B}$, e a equivalência, caracterizada como $\mathtt{A} \equiv \mathtt{B}$ e que significa $\mathtt{A} \sqsubseteq \mathtt{B}$ e $\mathtt{B} \sqsubseteq \mathtt{A}$. Para o conhecimento assertivo também existem axiomas, tanto para conceitos ($\mathtt{A}(x)$), quanto para papéis ($\mathtt{r}(x, y)$).

A linguagem $\mathcal{ALC}$ permite que seja feita a negação de conceitos complexos (não-atômicos). Tal propriedade permite descobrir, por exemplo, se algum conceito é satisfatível. Usando o símbolo $\sim$ para demonstrar equivalência, teremos que:

- $\neg(\forall r.\mathtt{A}) \sim \exists r.\neg\mathtt{A}$
- $\neg(\exists r.\mathtt{A}) \sim \forall r.\neg\mathtt{A}$
- $\neg(\mathtt{A} \sqcup \mathtt{B}) \sim \neg\mathtt{A} \sqcap \neg\mathtt{B}$
- $\neg(\mathtt{A} \sqcap \mathtt{B}) \sim \neg\mathtt{A} \sqcup \neg\mathtt{B}$
- $\neg\neg\mathtt{A} \sim \mathtt{A}$

A partir de sua constituição, é possível definir três sublinguagens da $\mathcal{ALC}$ (Schmidt-Schauß e Smolka, 1991). Elas estão descritas a seguir:

- $\mathcal{ALE}$: permite a descrição de conceitos simples sem a utilização de uniões;
- $\mathcal{ALU}$: deixa apenas conceitos simples serem descritos e restringe as quantificações de papéis existenciais à forma $\exists\mathtt{r}.\top$;
- $\mathcal{AL}$: é a intersecção das sublinguagens acima.

Os nomes das sublinguagens acima são derivados da seguinte maneira: $\mathcal{ALU}$ vem da junção de $\mathcal{AL}$ com uniões, $\mathcal{ALE}$ é a $\mathcal{AL}$ acrescida de quantificadores existenciais. A própria $\mathcal{ALC}$ é uma extensão da $\mathcal{AL}$ com adição de complementos (negações) para qualquer conceito, seja ele atômico ou complexo.

Nota-se que a sigla que representa o nome de uma linguagem expressa as características que ela possui. A $\mathcal{ALC}$ é base para outras linguagens mais expressivas, que são construídas a partir de sua extensão. Algumas delas são:

- $\mathcal{S}$: linguagem $\mathcal{ALC}$ com papéis transitivos;
- $\mathcal{F}$: propriedades funcionais;
- $\mathcal{N}$: restrição numérica;
- $\mathcal{C}$: negação de conceitos complexos;
- $\mathcal{U}$: união de conceitos;
- $\mathcal{E}$: quantificação existencial completa;
- $\mathcal{Q}$: restrição numérica qualificada;
- $\mathcal{O}$: nominal;
- $\mathcal{H}$: hierarquia de papéis;
- $\mathcal{I}$: propriedades inversas;
- $\mathcal{N}$: restrições de cardinalidade;
- $^{(\mathcal{D})}$: usa propriedades de tipo de dados.

Um exemplo de linguagem com essa nomenclatura é a $\mathcal{SHIF}^{(\mathcal{D})}$, que engloba a $\mathcal{ALC}$ e, ainda, hierarquia de papéis, propriedades inversas, transitivas e funcionais e tipos de dados. Outro exemplo é a lógica $\mathcal{SHOIN}^{(\mathcal{D})}$, que envolve a $\mathcal{SHIF}^{(\mathcal{D})}$ e restrições de cardinalidade em papéis.

3.3 Interpretação e Consequência Lógica

Uma interpretação $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consiste em um conjunto domínio ($\Delta^{\mathcal{I}}$) e uma função de interpretação $\cdot^{\mathcal{I}}$ que atribui a toda descrição de conceito um subconjunto de $\Delta^{\mathcal{I}}$, seguindo as seguintes equações (Baader *et al.*, 2008):

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $\bot^{\mathcal{I}} = \emptyset$
- $(\neg \mathsf{A})^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus \mathsf{A}^{\mathcal{I}}$
- $(\mathsf{A} \sqcup \mathsf{B})^{\mathcal{I}} = \mathsf{A}^{\mathcal{I}} \cup \mathsf{B}^{\mathcal{I}}$
- $(\mathsf{A} \sqcap \mathsf{B})^{\mathcal{I}} = \mathsf{A}^{\mathcal{I}} \cap \mathsf{B}^{\mathcal{I}}$
- $(\exists \mathsf{r} \,.\, \mathsf{A})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \text{existe } b \in \Delta^{\mathcal{I}} \text{ tal que } (a, b) \in \mathsf{r}^{\mathcal{I}}\}$
- $(\forall \mathsf{r} \,.\, \mathsf{A})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \text{para todo } b \in \Delta^{\mathcal{I}}, \text{se } (a, b) \in \mathsf{r}^{\mathcal{I}}, \text{então } b \in \mathsf{A}^{\mathcal{I}}\}$

Com isso, podemos ver que uma interpretação $\mathcal{I}$ satisfaz:

- $\mathtt{A} \equiv \mathtt{B}$ se e somente se $\mathtt{A}^{\mathcal{I}} = \mathtt{B}^{\mathcal{I}}$
- $\mathtt{A} \sqsubseteq \mathtt{B}$ se e somente se $\mathtt{A}^{\mathcal{I}} \subseteq \mathtt{B}^{\mathcal{I}}$
- a *TBox* $\mathcal{T}$ se e somente se satisfaz todos os elementos de $\mathcal{T}$
- $\mathtt{A}(a)$ se e somente se $a^{\mathcal{I}} \in A^{\mathcal{I}}$
- $\mathtt{r}(a, b)$ se e somente se $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in \mathtt{r}^{\mathcal{I}}$
- a *ABox* $\mathcal{A}$ se e somente se satisfaz todos os elementos de $\mathcal{A}$

Uma Base de Conhecimento $\mathcal{ALC}$ é um par $\Sigma = (\mathcal{T}, \mathcal{A})$ onde $\mathcal{T}$ é uma *TBox* e $\mathcal{A}$, uma *TBox*. Logo, pode ser dito que uma interpretação $\mathcal{I}$ é um modelo de Σ se satisfaz $\mathcal{T}$ e $\mathcal{A}$. Uma base de conhecimento Σ é satisfatível se admite um modelo.

Com isso, podemos definir Consequência Lógica como $\Sigma \models \phi$ se todo modelo de Σ é um modelo de ϕ. Na ontologia deste texto, isso pode ser aplicado da seguinte maneira, sendo:

- $\Sigma = (\mathcal{T}, \mathcal{A})$, onde:
  - $\mathcal{T}$ é a *TBox*: $\mathtt{HinoNacional} \sqsubseteq \mathtt{Musica}$
  - $\mathcal{A}$ é a *ABox*: `HinoNacional(hinoNacionalBrasileiro)`

É possível observar que $\Sigma \models$ `Musica(hinoNacionalBrasileiro)` será uma consequência lógica.

3.4 Equivalências com a Lógica de Primeira Ordem

As LD, como definido anteriormente, são um subconjunto das LPO. Portanto, toda e qualquer expressão expressa nessa linguagem terá um equivalente em LPO.

Tal relação se estende até às definições que ela utiliza em sua constituição. Os conceitos, papéis e indivíduos acima citados, correspondem, respectivamente, a predicados unários, predicados binários e constantes em Lógica de Primeira Ordem.

Com essa elucidação, é possível elaborar uma tabela que mostra as principais equivalências entre LD e LPO, com intenção de usá-la para uma eventual tradução entre as lógicas. Definindo t_x como a interpretação em x de uma sentença, já que é necessária uma variável livre para a tradução, teremos o resultado da Tabela 3.3.

Conceito	**LD**	**Tradução**	**LPO**
Classe	$\mathtt{A}$	$t_x(\mathtt{A})$	$A(x)$
União	$\mathtt{A} \sqcup \mathtt{B}$	$t_x(\mathtt{A} \sqcup \mathtt{B})$	$A(x) \vee B(x)$
Intersecção	$\mathtt{A} \sqcap \mathtt{B}$	$t_x(\mathtt{A} \sqcap \mathtt{B})$	$A(x) \wedge B(x)$
Universal	$\forall\, \mathtt{r}.\mathtt{A}$	$t_x(\forall \mathtt{r}.\mathtt{A})$	$\forall y(r(x,y) \rightarrow t_y(A))$
Existencial	$\exists\, \mathtt{r}.\mathtt{A}$	$t_x(\exists \mathtt{r}.\mathtt{A})$	$\exists y(r(x,y) \wedge t_y(A))$
Subsunção	$\mathtt{A} \sqsubseteq \mathtt{B}$	$t_x(\mathtt{A} \sqsubseteq \mathtt{B})$	$\forall x(t_x(A) \rightarrow t_x(B))$
Equivalência	$\mathtt{A} \equiv \mathtt{B}$	$t_x(\mathtt{A} \equiv \mathtt{B})$	$\forall x(t_x(A) \longleftrightarrow t_x(B))$

Tabela 3.3: Tabela para tradução entre LD e LPO

Usando a ontologia de música como exemplo, pode-se ver como essa tradução é aplicada:

1. $\texttt{Pop} \sqsubseteq \texttt{Cancao}$ vira $\forall x(\texttt{Pop}(x) \rightarrow \texttt{Cancao}(x))$.

2. $\texttt{Cancao} \sqcup \texttt{HinoNacional} \sqsubseteq \texttt{Musica}$ é traduzida como $\forall x(\texttt{Cancao}(x) \vee \texttt{HinoNacional}(x) \rightarrow \texttt{Musica}(x))$.

3. $\forall\, \texttt{canta.Musica} \sqsubseteq \texttt{Cantor}$ corresponde a $\forall x(\forall y(\texttt{canta}(x,y) \rightarrow \texttt{Musica}(y)) \rightarrow \texttt{Cantor}(x))$.

4. $\exists\, \texttt{canta.Pop} \sqsubseteq \texttt{Cantor} \sqcap \texttt{Compositor}$ equivale a $\forall x(\exists y(\texttt{canta}(x,y) \wedge \texttt{Pop}(y)) \rightarrow \texttt{Cantor}(x) \wedge \texttt{Compositor}(x))$.

3.5 Consistência

Após uma Base de Conhecimento ser criada, é necessário que ela seja consistente, ou seja, ela não pode se contradizer. A inconsistência acontece quando a inferência de uma asserção e sua negação, ao mesmo tempo, é possível (por exemplo, $\texttt{A}(a)$ e $\neg\texttt{A}(a)$).

Para verificar tal propriedade, podem ser aplicadas as regras de *tableau* para Lógicas de Descrição. Com elas, é possível determinar de maneira algorítmica a consistência de uma Base de Conhecimento. São elas:

Regra E (ou $\sqcap$-*rule*) Se $(\texttt{A} \sqcap \texttt{B})(a)$ está em $\mathcal{A}$, mas $\texttt{A}(a)$ e $\texttt{B}(a)$ não estão os dois em $\mathcal{A}$, então a *ABox* $\mathcal{A}$ será acrescida de $\texttt{A}(a)$ e $\texttt{B}(a)$.

Regra OU (ou $\sqcup$-*rule*) Quando $(\texttt{A} \sqcup \texttt{B})(a)$ está em $\mathcal{A}$, mas nem $\texttt{A}(a)$ nem $\texttt{B}(a)$ está em $\mathcal{A}$, então à *ABox* $\mathcal{A}$ será incluído $\texttt{A}(a)$ ou $\texttt{B}(a)$.

Regra EXISTE (ou $\exists$-*rule*) Se $(\exists \texttt{r.A})(a)$ está em $\mathcal{A}$, e não há nenhum indivíduo c tal que $\texttt{r}(a,c)$ e $\texttt{A}(c)$ estejam em $\mathcal{A}$, então tanto $\texttt{r}(a,b)$ quanto $\texttt{A}(b)$ serão adicionados a $\mathcal{A}$, onde b é um novo indivíduo.

Regra PARA TODO (ou $\forall$-*rule*) Quando se $(\forall \texttt{r.A})(a)$ e $\texttt{r}(a,b)$ estão em $\mathcal{A}$, mas $\texttt{A}(b)$ não está, esse último será colocado em $\mathcal{A}$.

Para descobrir se uma fórmula é insatisfatível, basta aplicar essas regras na sentença lógica. Se, no final, um *Clash* (conflito) for alcançado, isto é, quando temos uma asserção e sua negação na *ABox*, a sentença é insatisfatível.

Segue, na figura 3.1, um exemplo para aplicar essas regras (Koubarakis, 2010):

1	$(\forall \texttt{hasChild}.\texttt{Male} \sqcap \exists \texttt{hasChild}.\neg\texttt{Male})(\texttt{a})$	given
2	$(\forall \texttt{hasChild}.\texttt{Male})(\texttt{a})$	1, $\sqcap$-rule
3	$(\exists \texttt{hasChild}.\neg\texttt{Male})(\texttt{a})$	1, $\sqcap$-rule
4	$\texttt{hasChild}(\texttt{a},\texttt{b})$	3, $\exists$-rule
5	$(\neg\texttt{Male})(\texttt{b})$	3, $\exists$-rule
6	$\texttt{Male}(\texttt{b})$	2, 4, $\forall$-rule
7	**Clash**	5, 6

Figura 3.1: Exemplo de aplicação das regras de *Tableau*

Note que nesse exemplo não há uso da $\sqcup$-*rule*. Quando ela é usada, são criados dois ramos. Para que a fórmula seja insatisfazível, os dois devem resultar em choque (*Clash*).

Para verificar que sentenças no estilo $\texttt{C} \sqsubseteq \texttt{D}$ são válidas, deve ser verificada a satisfazibilidade de $\texttt{C} \sqcap \neg\texttt{D}$. Se essa última for insatisfatível, a primeira é consequência lógica da base de conhecimento.

Capítulo 4

Revisão de Crenças

A área de pesquisa que estuda as alterações em estados de crenças é conhecida como Revisão de Crenças. Na campo das ontologias, essa área trata o problema da inconsistência. Como escrito anteriormente, uma ontologia (ou um sistema de crenças) fica inconsistente quando alguma informação incompatível com as crenças estabelecidas até o momento é incorporada.

Para entender os estudos dessa área, seja o seguinte exemplo do assunto da ontologia descrita neste trabalho, assumindo que ela possui os fragmentos de conhecimento abaixo em alguma linguagem formal de representação:

1. *Toda música estadunidense pertence ao subgênero Country.*
2. *Todos os cantores estadunidenses cantam apenas músicas estadunidenses.*
3. *Lady Gaga é uma cantora estadunidense.*
4. *Lady Gaga canta a música "Bad Romance".*
5. *Os subgêneros Country e Pop são disjuntos.*

Com esse conjunto de informações, é possível inferir que a música "*Bad Romance*" pertence ao subgênero *Country*. No entanto, suponha que a seguinte informação chegue até o agente:

A música "Bad Romance" pertence ao subgênero Pop.

Como se pode ver, a nova informação entra em choque direto com a inferência realizada e, se absorvida, deixará a ontologia inconsistente. Para isso, é necessário fazer uso de alguma operação estudada na área.

De uma forma geral, esse campo da Inteligência Artificial estuda qualquer alteração dos estados epistêmicos, desde a simples adição de algum novo conhecimento que não entra em conflito com o que já está na ontologia. Além disso, ele lida também com a remoção segura

de alguma informação. Para que uma remoção seja considerada segura, é necessário apagar, além da informação propriamente dita, aquelas que a implicam.

O estado epistêmico de um agente nada mais é do que o conjunto de tudo aquilo em que ele acredita e como as suas crenças se relacionam num certo instante. Pode-se entender um estado epistêmico também como uma representação idealizada do estado cognitivo de um agente em determinado momento, como explicou Gärdenfors (Gärdenfors, 1988).

Uma alteração do estado epistêmico seria, portanto, uma revisão que acontece quando o agente recebe uma nova informação que possivelmente entra em choque com as informações que ele possui no estado atual. Essa revisão deve manter as crenças antigas ao máximo, fazendo assim, uma mudança mínima (Resina, 2010).

O paradigma AGM, que será usado neste trabalho, recebe este nome por causa das iniciais dos autores do artigo considerado o pontapé inicial desta área de pesquisa (Alchourrón *et al.*, 1985). Nela, os estados de crenças são representados por conjuntos logicamente fechados de sentenças, ou seja, conjuntos K tais $K = \mathrm{Cn}(K)$, onde $\mathrm{Cn}(K)$ representa o conjunto de todas as consequências lógicas de K.

Quando $K = \mathrm{Cn}(K)$, diz-se que há um equilíbrio do estado epistêmico. Com K sendo logicamente fechado, se $K \models \psi$, sendo ψ uma sentença qualquer, tem-se que $\psi \in K$. O uso de conjuntos logicamente fechados aumenta a eficiência do reparo (Hansson, 1991), já que não é necessário realizar inferências sobre os axiomas.

As sentenças que serão trabalhadas são de uma lógica (L, Cn), tal que L é uma linguagem fechada em relação aos conectivos lógicos $\wedge$, $\vee$, $\rightarrow$ e $\neg$ e que satisfaz (Ribeiro, 2010):

Tarskianicidade A lógica é monotônica (todas as consequências dedutíveis continuam assim mesmo após a adição de alguma sentença), idempotente e satisfaz inclusão;

Dedução $\alpha \in \mathrm{Cn}(K \cup \{\beta\})$ se e somente se $\beta \rightarrow \alpha \in \mathrm{Cn}(K)$, onde α e β são sentenças lógicas;

Compacidade se $\alpha \in \mathrm{Cn}(K)$, então existe $K' \subseteq K$ finito tal que $\alpha \in \mathrm{Cn}(K')$;

Supraclassicalidade Toda consequência da lógica (L, Cn) é também uma consequência da lógica proposicional. Vale observar que essa propriedade não serve para Lógicas de Descrição.

4.1 Tratamento da informação

Antes de ver como é possível tratar as inconsistências causadas pela entrada de alguma informação nova, é necessário observar como uma sentença α qualquer é tratada em algum conjuntos de crenças K.

Existem três tipo de tratamento, descritos abaixo:

1. α é aceita pelo conjunto de crenças. Isso pode acontecer de duas maneiras diferentes:

   (a) α é aceita explicitamente. Quando isso acontece, temos que $\alpha \in K$;

   (b) α é aceita implicitamente. Esse caso ocorre quando $\alpha \in \text{Cn}(K) \setminus K$, para Bases de Crenças.

2. α é rejeitada. Aqui, temos que $\neg\alpha \in K$ ou que $\neg\alpha \in \text{Cn}(K) \setminus K$, para Bases de Crenças.

3. α é indeterminada. Deste modo, K não conhece a sentença α, assim, $\neg\alpha \notin \text{Cn}(K)$ e $\alpha \notin \text{Cn}(K)$.

Existem também algumas questões metodológicas que precisam ser resolvidas, ou pelo menos observadas antes de alguma Revisão de Crenças. Gärdenfors definiu algumas delas (Gärdenfors, 2003).

A primeira é em relação à representação das crenças na base de dados. Vale notar que, como definido no Capítulo 2, é necessário o uso de alguma linguagem formal de representação. A maioria das bases de dados trabalha com fatos e regras como formas primitivas de informação. O mecanismo escolhido para a Revisão deve levar em conta o formalismo escolhido para a representação.

Uma outra questão se preocupa com os elementos explicitamente representados na base de crenças e os que podem ser derivados desses. Existem bases que dão algum *status* especial aos elementos explícitos, e outras que dão a mesma importância para todos.

A última questão é referente a qual retração fazer, ou seja, qual edição fazer na base de dados. A lógica, propriamente dita, não é suficiente para definir quais são os melhores elementos para serem removidos ou mantidos. Uma das ideias referentes a isso é que a quantidade informação perdida durante o reparo seja a mínima possível.

Nas bases de dados que dão diferentes prioridades para as suas crenças, pode-se remover as que possuem menor prioridade, em detrimento daquelas com maior importância. Tudo isso depende de como o sistema de crenças está estruturado.

4.2 Operações clássicas de Revisão de Crenças

Existem três operações clássicas de Revisão de Crenças (Gärdenfors, 1992). São elas a Expansão, a Revisão e a Contração. Apenas a Expansão é definida diretamente, enquanto as duas últimas são definidas por postulados.

Para as definições que serão feitas será usada uma lógica L, que é baseada na Lógica de Primeira Ordem ou Proposicional. As variáveis, que são as sentenças lógicas, serão representadas pelo alfabeto grego. Os conjuntos de crenças, com letras latinas maiúsculas.

4.2.1 Expansão

A única operação definida diretamente recebe a notação $K + \alpha$. Ela é também a mais simples, já que representa a chegada de algum conhecimento à base sem que os anteriores sofram modificações. Além disso, as crenças que podem ser inferidas também são adicionadas.

O conjunto resultante é, portanto, formado pelo fecho lógico da união do conjunto inicial com a informação nova, o que significa que $K + \alpha = \mathrm{Cn}(K \cup \alpha)$. Vale ressaltar que o operação não exige que o conjunto permaneça consistente.

A informação α não era necessariamente indeterminada. Pode ser que ela já estivesse na base. Depois da operação de Expansão, de qualquer modo, α fica aceito.

4.2.2 Contração

Esta operação é o contrário da anterior. Em vez de uma nova informação ser adicionada à base de conhecimento, deseja-se que algum conhecimento seja removido. A operação é representada por $K - \alpha$. No entanto, as referências à Expansão acabam por aí.

Às vezes, retirar α de K não é suficiente. Nesses casos, é necessário desistir de algumas crenças do conjunto. As sentenças que são abandonadas são aquelas que implicam α. O critério da mudança mínima deve ser aplicado aqui, removendo o mínimo de sentenças possível, ou as que possuem prioridade menor.

A informação α, que era aceita, agora é indeterminada, já que foi abandonada. Agora, o conjunto resultante $K - \alpha$ não possui α explicitamente nem implicitamente.

Os postulados de racionalidade que regem a Contração seguem abaixo, com uma sucinta explicação sobre o seu significado.

Fecho $K - \alpha = \mathrm{Cn}(K - \alpha)$
: Após a Contração, o conjunto resultante é logicamente fechado.

Inclusão $K - \alpha \subseteq K$
: A operação não permite que novas sentenças sejam adicionadas ao conjunto.

Vacuidade Se $\alpha \notin K$, então $K - \alpha = K$
: Quando a fórmula que se deseja contrair não está no conjunto, o mesmo é inalterado.

Sucesso Se $\alpha \notin \mathrm{Cn}(\varnothing)$, então $\alpha \notin K - \alpha$
: A não ser que α seja uma tautologia, ela não estará contida no conjunto resultante da operação.

Equivalência Se $\mathrm{Cn}(\alpha) = \mathrm{Cn}(\beta)$, então, $K - \alpha = K - \beta$
: Se duas fórmulas possuem consequências lógicas iguais (são logicamente equivalentes), a Contração por uma terá o mesmo resultado do que a Contração pela outra.

Recuperação $K \subseteq (K - \alpha) + \alpha$

A operação de Contração é desfeita pela Expansão.

Intersecção conjuntiva $(K - \alpha) \cap (K - \beta) \subseteq K - (\alpha \wedge \beta)$

As crenças que não foram descartadas na Contração por α ou por β serão mantidas na Contração por $\alpha \wedge \beta$.

Inclusão conjuntiva Se $\alpha \notin K - (\alpha \wedge \beta)$ então, $K - (\alpha \wedge \beta) \subseteq K - \alpha$

Se a sentença α é removida na Contração por $(\alpha \wedge \beta)$, então tudo o que seria removido na Contração por α apenas será removido.

Os seis primeiros postulados são os chamados postulados básicos. Os últimos dois são os postulados suplementares.

Ainda para a Contração existem dois construtores especiais definidos, que serão discutidos à frente.

4.2.3 Revisão

A Revisão, assim como a Expansão, tem a ver com adicionar alguma crença α ao conjunto K. O conjunto resultante é indicado por $K * \alpha$. A nova informação α era rejeitada ou indeterminada e passa a ser aceita, ou vice-versa.

Como na operação anterior, o conjunto anterior precisa se mostrar consistente. É necessário apagar o mínimo possível das fórmulas de K. De mesmo modo que a Contração, ela não é definida diretamente, mas sim por oito postulados, os seis primeiros, básicos, e os dois últimos, suplementares. São eles:

Fecho $K * \alpha = \mathrm{Cn}(K * \alpha)$

Após a Revisão, o fecho lógico do conjunto é mantido.

Sucesso $\alpha \in \mathrm{Cn}(K * \alpha)$

A operação garante que o conjunto resultante contenha a nova informação.

Inclusão $K * \alpha \subseteq K + \alpha$

A Revisão não adiciona nada a mais do que a Expansão adicionaria no conjunto.

Preservação Se $\neg\alpha \notin K$, então $K + \alpha \subseteq K * \alpha$

Quando a fórmula por que se deseja realizar a Revisão não está no conjunto, é garantido que tudo que seria acrescentado na Expansão será adicionado na Revisão. Juntando esse postulado com o **Inclusão**, temos que, sendo a Revisão feita por uma fórmula logicamente consistente com K, o mesmo resultado seria alcançado por uma Expansão, ou seja, $K + \alpha = K * \alpha$.

Consistência $K * \alpha = L$ se e somente se $\neg\alpha \in \mathrm{Cn}(\varnothing)$

A operação de Revisão gera um conjunto consistente, e vice-versa. Neste postulado, L representa um conjunto consistente qualquer.

Equivalência Se $\mathrm{Cn}(\alpha) = \mathrm{Cn}(\beta)$, então, $K * \alpha = K * \beta$

Se duas fórmulas possuem consequências lógicas iguais (são logicamente equivalentes), a Revisão por uma terá o mesmo resultado do que a Revisão pela outra.

Conjunção $K * (\alpha \wedge \beta) \subseteq (K * \alpha) + \beta$

O resultado de revisar pela conjunção de duas fórmulas está contido naquele obtido pela Revisão por uma fórmula e posterior Expansão pela outra.

Vacuidade Se $\neg\beta \notin K * \alpha$ então, $(K * \alpha) + \beta \subseteq K * (\alpha \wedge \beta)$

Se a negação de β não pertence ao resultado da Revisão por α, então a Revisão por α e posterior Expansão por β resulta em um conjunto que está contido no que é obtido pela Revisão pela conjunção dessas fórmulas. Unindo com **Conjunção**, se a Revisão $K * \alpha$ for feita não resultando em $\neg\beta$, e então a Expansão por β for realizada, o mesmo resultado seria alcançado pela Revisão de K por $(\alpha \wedge \beta)$, ou seja, $(K * \alpha) + \beta = K * (\alpha \wedge \beta)$

Existem alguns construtores para a operação da Revisão. Um deles será discutido mais à frente.

4.2.4 Relações entre operações

A Revisão e a Contração podem ser construídas uma em função da outra, como escreveu Gärdenfors (Gärdenfors, 1988). Isso pode ser alcançado por duas relações:

Identidade de Levi $K * \alpha = (K - \neg\alpha) + \alpha$

Ela mostra que a Revisão por uma fórmula α resulta no mesmo conjunto que a Contração por $\neg\alpha$ e consecutiva Expansão por α, para conjuntos logicamente fechados.

Identidade de Harper $K - \alpha = (K * \neg\alpha) \cap K$

A Contração por uma fórmula pode ser obtida pela intersecção entre o conjunto resultante após uma Revisão pela negação da fórmula e o conjunto inicial.

4.3 Pseudocontração

Além das operações básicas, outras foram desenvolvidas. Pode acontecer de uma contração não ser a melhor operação a ser aplicada numa base de crenças, por exemplo. Considere o conjunto $K = \{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n\}$. A aplicação da contração resultará em $K - \alpha_i = \varnothing$,

por **Sucesso** e **Inclusão**. Esse não é o resultado da mesma operação sob o conjunto $K' = \{\alpha_1, \alpha_2, ..., \alpha_n\}$, embora $\mathrm{Cn}(K) = \mathrm{Cn}(K')$.

O que pode ser feito para evitar esse tipo de resultado é substituir o postulado da inclusão por um mais fraco (Hansson, 1989):

Inclusão lógica $\mathrm{Cn}(K - \alpha) \subseteq \mathrm{Cn}(K)$

Com isso, a operação executada será chamada de Pseudocontração (Hansson, 1993b). Para esta operação, existe um construtor que será descrito neste trabalho.

4.4 Construtores para as operações

Como mencionado anteriormente, existem alguns construtores para as operações. Para a Expansão isso não é necessário, já que para essa operação, basta adicionar um axioma, sem se preocupar com a consistência. Esses construtores são espécies de funções, que tentam passar uma "receita" de como fazer essa operação.

4.4.1 Contração *Kernel*

Para Teorias

A contração *Kernel* precisa de algumas definições para ser discutida. Ela encontra conjuntos minimais que implicam a informação α que se deseja contrair, e então remove pelo menos um elemento de cada um desses conjuntos, para que α deixe de pertencer ao (ou deixe de ser consequência lógica do) conjunto (Hansson, 1994).

Ela é definida da seguinte forma:

$$K -_\sigma \alpha = K \setminus \sigma(K \perp\!\!\!\perp \alpha),$$

onde $-_\sigma$ é a operação chamada de Contração *Kernel*, σ, uma função de incisão, e $\perp\!\!\!\perp$, a representação de um conjunto-*Kernel*, definidas abaixo:

Conjunto-*Kernel* Sendo K um conjunto que está contido na lógica L e $\alpha \in L$, o conjunto-*Kernel*, denotado por $K \perp\!\!\!\perp \alpha$ é o conjunto de todos os subconjuntos minimais de K que implicam α. Formalmente, para quaisquer conjuntos I e J, J pertence a $K \perp\!\!\!\perp \alpha$ se e somente se:

- $J \subseteq K$;
- $\alpha \in \mathrm{Cn}(J)$;
- para todo $I \subsetneq J$, vale que $\alpha \notin \mathrm{Cn}(I)$.

Função de incisão É uma função σ, para um conjunto de crenças K, que seleciona no mínimo uma fórmula de cada elemento de um conjunto-*Kernel*, ou seja, para todo $\alpha \in L$:

- $\sigma(K \perp\!\!\!\perp \alpha) \subseteq \bigcup(K \perp\!\!\!\perp \alpha)$;
- se $J \neq \varnothing$, então $J \cap \sigma(K \perp\!\!\!\perp \alpha) \neq \varnothing$, para todo $J \in K \perp\!\!\!\perp \alpha$

Com isso, pode-se concluir que a função de incisão escolhe, dentre os subconjuntos maximais de K tais que $K \models \alpha$, os "piores" de cada, de acordo com algum critério arbitrário. As remoções desses escolhidos do conjunto fornecem o resultado da operação. A função de incisão é baseada em hierarquia. Um operador $-_\sigma$ para uma base B qualquer é uma Contração se e somente se satisfaz os oito postulados de racionalidade dessa operação.

Para Bases de Crenças

Existe uma generalização para Bases de Crenças para a Contração *Kernel*. A construção é a mesma, mas as propriedades mudam. Um operador $-_\sigma$ para uma base B qualquer é uma Contração *Kernel* se e somente se satisfaz os seguintes postulados:

Inclusão $B -_\sigma \alpha \subseteq B$
A operação não permite que novas sentenças sejam adicionadas ao conjunto, resultando em um conjunto com o mesmo número de sentenças ou menos.

Sucesso Se $\alpha \notin \mathrm{Cn}(\varnothing)$, então $\alpha \notin \mathrm{Cn}(B -_\sigma \alpha)$
A não ser que α seja uma tautologia, ela não será consequência do no conjunto resultante da operação.

Core-retainment Se $\beta \in B \setminus (B -_\sigma \alpha)$, então existe B' tal que $B' \subseteq B$, onde:

- $\alpha \notin \mathrm{Cn}(B')$
- $\alpha \in \mathrm{Cn}(B' \cup \{\beta\})$

Uniformidade Se, para todo $B' \subseteq B$, valer que $\alpha \in \mathrm{Cn}(B')$, se e somente se $\beta \in \mathrm{Cn}(B')$, então $B -_\sigma \alpha = B -_\sigma \beta$
Se em todos os subconjuntos de B que implicam α também for implicado β e vice-versa, a Contração por qualquer uma dessas fórmulas terá o mesmo resultado.

4.4.2 Contração *Partial Meet*

Para Teorias

Essa construção quer atingir o critério da mudança mínima. Tal critério é atingido por meio da obtenção de conjuntos maximais onde a sentença α não é consequência lógica. A

operação fornecerá um resultado baseado em um conjunto de conjuntos maximais, chamado de conjunto-resíduo (Alchourrón *et al.*, 1985).

A **Contração *Partial Meet*** para Teorias é definida da seguinte forma:

$$K -_\gamma \alpha = \bigcap \gamma(K \perp \alpha),$$

onde $-_\gamma$ é a operação chamada de Contração *Partial Meet*, γ é uma função de seleção, e $\perp$ é a representação de um conjunto-resíduo, definidas abaixo:

Conjunto-Resíduo Definido como $T \perp \alpha$, onde T é um conjunto que está contido na lógica L e $\alpha \in L$. Ele é o conjunto de todos os subconjuntos maximais de T que não implicam α. Formalmente, para quaisquer conjuntos R e S, o conjunto S pertence a $T \perp \alpha$ se e somente se:

- $S \subseteq T$;
- $\alpha \notin \mathrm{Cn}(S)$;
- se $S \subsetneq R \subseteq T$, então $\alpha \in \mathrm{Cn}(R)$, para todo R.

Função de seleção É uma função γ que seleciona algumas crenças de um conjunto T, ou seja, $\forall \alpha \in L$:

- se $T \perp \alpha \neq \varnothing$, então $\varnothing \neq \gamma(T \perp \alpha) \subseteq T \perp \alpha$;
- caso contrário, $\gamma(T \perp \alpha) = \{T\}$.

É possível entender, portanto, que o conjunto resultante da Contração *Partial Meet* é a intersecção dos subconjuntos maximais que não implicam α escolhidos pela função de seleção γ e são as crenças em que o agente acredita de forma mais arraigada, ou seja, as crenças que possuem mais implicantes. A função de seleção é baseada em relações transitivas. Um operador $-_\gamma$ para uma base B qualquer é uma Contração se e somente se satisfaz os oito postulados de racionalidade dessa operação.

Para Bases de Crenças

A operação definida acima pode ser generalizada para bases de crenças. A construção é a mesma, mas as propriedades mudam. Uma operação $-_\gamma$ para uma base B é uma Contração *Partial Meet* se e só se satisfaz os seguintes postulados (Hansson, 1993a):

Inclusão $B -_\gamma \alpha \subseteq B$

Semelhante ao da Contração *Kernel*.

Sucesso Se $\alpha \notin \mathrm{Cn}(\varnothing)$, então $\alpha \notin \mathrm{Cn}(B -_\gamma \alpha)$

Funciona como no construtor anterior.

Relevância Se $\beta \in B \setminus (B -_\gamma \alpha)$ então existe B' tal que $B -_\gamma \alpha \subseteq B' \subseteq B$, onde:

- $\alpha \notin \mathrm{Cn}(B')$
- $\alpha \in \mathrm{Cn}(B' \cup \{\beta\})$

Se β for removido na Contração $B -_\gamma \alpha$, então, de alguma maneira, β tem relação com o fato que $B \models \alpha$. Esse postulado assegura que elementos que não possuem bons motivos para a remoção fiquem na base de crenças. É uma versão mais forte do postulado ***Core-retainment*** apresentado acima.

Uniformidade Se, para todo $B' \subseteq B$, valer que $\alpha \in \mathrm{Cn}(B')$, se e somente se $\beta \in \mathrm{Cn}(B')$, então $B -_\gamma \alpha = B -_\gamma \beta$
Análogo ao da Contração *Kernel*.

4.4.3 Revisão *Kernel*

A Revisão *Kernel* tem aspectos em comum com a Contração *Kernel*, como pode-se imaginar. Ela é definida da seguinte maneira, para um conjunto K e uma sentença α (Ribeiro e Wassermann, 2008):

$$K *_\sigma \alpha = K \setminus \sigma(K \downarrow\downarrow \alpha) \,,$$

onde $*_\sigma$ é a operação chamada de Revisão *Kernel*, σ é uma função de incisão, e $\downarrow\downarrow$ é a representação de um conjunto-*kernel* para a revisão, como definido abaixo:

Conjunto-*Kernel* para a revisão Definido como $K \downarrow\downarrow \alpha$, onde K é um conjunto contido na lógica L e $\alpha \in L$. Ele é o conjunto de todos os subconjuntos minimais de K que são inconsistentes com α. Formalmente, para quaisquer conjuntos I e J, J pertence a $K \downarrow\downarrow \alpha$ se e somente se:

- $J \subseteq K$;
- $\alpha \in \mathrm{Cn}(J)$;
- para todo $I \subsetneq J$, vale que $\alpha \notin \mathrm{Cn}(I)$.

Sua construção só é possível porque falta **Supraclassicalidade** para as Lógicas de Descrição.

Essa operação satisfaz os seguintes postulados:

Inclusão $K *_\sigma \alpha \subseteq K + \alpha$
Se α for consistente com K, a Revisão será, na verdade, uma Expansão. Caso contrário, algumas sentenças. Esse postulado impede a adição de sentenças irrelevantes para o conjunto.

Sucesso $\alpha \in K *_\sigma \alpha$

Também chamado de sucesso forte, obriga que α esteja no conjunto gerado pela operação.

Consistência fraca Se α é consistente/coerente, então $K *_\sigma \alpha$ será consistente/coerente. Se uma base de conhecimento é inconsistente, então existe algum conceito nela que é necessariamente vazio. Essa noção é chamada de **Coerência**.

Core-retainment Se $\beta \in K \setminus K *_\sigma \alpha$, então existe $K' \subseteq K \cup \{\alpha\}$ tal que K' é consistente/coerente e $K' \cup \{\beta\}$ não é consistente/coerente.

Pré-expansão $K + \alpha *_\sigma \alpha = K *_\sigma \alpha$

A Revisão tem o mesmo resultado que uma Expansão e posterior Revisão pela mesma fórmula.

4.4.4 Pseudocontração SRW

Uma proposta de pseudocontração foi proposta em 2015 (Santos *et al.*, 2015). Ela utiliza um operador de consequência lógica Cn*, tarskiano. Tal operação é chamada de Pseudocontração SRW.

Ela é definida da seguinte forma:

$$K -_* \alpha = \bigcap \gamma(\text{Cn*}(K) \perp \alpha),$$

onde $-_\gamma$ é a operação chamada de Pseudocontração SRW, γ é uma função de seleção, e $\perp$ é a representação de um conjunto-resíduo. Essa operação satisfaz os seguintes postulados:

Inclusão* $K -_* \alpha \subseteq \text{Cn*}(K)$

Relevância* Se $\beta \in \text{Cn*}(K) \setminus (K -_* \alpha)$ então existe K' tal que $K -_* \alpha \subseteq K' \subseteq \text{Cn*}(K')$, mas $\alpha \in \text{Cn}(K' \cup \{\beta\})$

Uniformidade* Se, para todo $K' \subseteq \text{Cn*}(K)$, valer que $\alpha \in \text{Cn*}(K')$, se e somente se $\beta \in \text{Cn}(K')$, então $B -_* \alpha = K -_* \beta$

Sucesso Se $\alpha \notin \text{Cn}(\varnothing)$, então $\alpha \notin \text{Cn}(K - \alpha)$

Nota-se que o postulado do sucesso é o mesmo da contração *Partial Meet*. Isso ocorre porque é necessário que a fórmula pseudocontraída não seja classicamente implicada pelo conjunto resultante da operação. Os demais postulados são versões enfraquecidas dos postulados da Contração *Partial Meet*.

Capítulo 5

Ferramentas Computacionais

O interesse no estudo de Desenvolvimento de Ontologias, Lógicas de Descrição e Revisão de Crenças, deve-se, em parte, às aplicações computacionais que tais áreas possuem. Neste capítulo, algumas delas serão abordadas.

5.1 OWL

O final da década de 1990 foi a época em que surgiu a ideia de Web Semântica. Esse conceito, já consolidado hoje, seria uma parte da *World Wide Web* onde é possível o trabalho cooperativo entre as máquinas e os seres humanos. Ele ocorre a partir dos significados que são dados aos conteúdos disponíveis na rede, para que haja compreensão por ambos os tipos de agentes (Herman, 2001).

A fim de que alguma Linguagem de Representação fosse criada, a W3C (sigla para Consórcio para a *World Wide Web*) fundou o "*Web Ontology Working Group*", no início dos anos 2000 (W3C, 2001).

Os primeiros rascunhos foram publicados em julho de 2002, e em fevereiro de 2004, a OWL (*Web Ontology Language*) tornou-se o padrão recomendado pela W3C para o processamento de ontologias (W3C, 2004).

Em outubro de 2007, um novo grupo foi concebido para adicionar alguns recursos à linguagem. Dois anos depois, a W3C fez o lançamento da OWL 2, que seria compatível com editores e raciocinadores semânticos (W3C, 2007). Ela foi recomendada oficialmente pela W3C em dezembro de 2012 (W3C, 2012b).

Abaixo estão algumas características das duas versões da OWL.

5.1.1 OWL Web Ontology Language

A primeira versão possui três sublinguagens (Bechhofer *et al.*, 2004). Cada uma possui suas particularidades:

- *Lite*: Sublinguagem mais simples, feita para hierarquias de classificação com restringimentos simples. Suporta restrições de cardinalidade simples (apenas 0 ou 1). Foi demonstrado que OWL-Lite é equivalente a $\mathcal{SHIF}^{(\mathcal{D})}$ (Grau *et al.*, 2008).

- DL: É equivalente a $\mathcal{SHOIN}^{(\mathcal{D})}$. Oferece expressividade máxima evitando alguns problemas de decidibilidade. Inclui todos os construtores da OWL. Possui mais complexidade formal do que a sublinguagem *Lite*.

- *Full*: Versão mais completa, oferecendo expressividade máxima e a liberdade sintática do RDF, só que sem garantias computacionais. Nessa sublinguagem, pode acontecer de uma Classe ser tratada como uma coleção de indivíduos ou como um indivíduo, o que acarreta problemas de decidibilidade. Não corresponde a nenhuma lógica de descrição.

5.1.2 OWL 2 Web Ontology Language

A segunda versão também possui sublinguagens (W3C, 2012a). Elas são:

- EL: Corresponde à lógica $\mathcal{EL}$++. Funciona bem para ontologias com muitas classes e/ou propriedades. Permite a existência de algoritmos de inferência polinimiais.

- QL: Baseada na lógica de descrição DL-*Lite*, foi feita para aplicações em que o volume de instâncias é muito grande e que a consulta de dados é a tarefa mais realizada. Tais consultas podem até ser implementadas usando sistemas de bancos de dados convencionais.

- RL: Criada para ontologias que precisam de raciocínio escalável, ou seja, que são possivelmente grandes em número de instâncias, mas que precisam de uma sublinguagem que mantenha um bom poder expressivo. É baseada em uma lógica de descrição chamada DLP.

- DL: Assim como no lançamento anterior, oferece bastante expressividade, sem esbarrar em problemas de execução por causa da indecibilidade. Ela pode ser mapeada para a LD $\mathcal{SROIQ}^{(\mathcal{D})}$.

- *Full*: Análoga à da primeira versão. Muito expressiva, porém, indecidível.

5.2 *Protégé*

O *Protégé* é um editor semântico, compatível com a OWL 2. Ele foi desenvolvido pelo Centro de Pesquisa para Informática Biomédica da Universidade de Stanford, em colaboração com

Figura 5.1: O logotipo do *Protégé*.

alguns programadores da Universidade de Manchester [1]. Sua primeira versão foi lançada em 1999, e a atual é de 2017. Seu logotipo está exibido na Figura 5.1

Seguindo os princípios do *Software* Livre, ele é um arcabouço gratuito e de código aberto, feito para a construção de sistemas inteligentes, com o uso ou não de uma interface gráfica.

Assim como o ambiente de desenvolvimento integrado Eclipse, o *Protégé* é bastante flexível porque é possível desenvolver uma grande variedade de *plug-ins* para serem acoplados a ele.

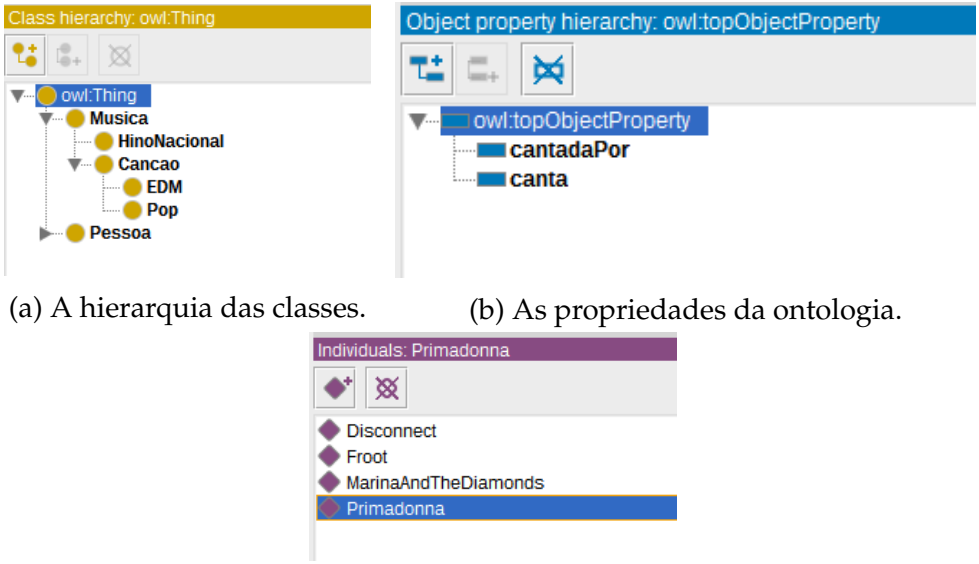
Para exemplificar alguns recursos do *Protégé*, será usada, como exemplo, parte da ontologia criada neste estudo:

- Os conceitos:
  - `Musica` $\equiv$ `Cancao` $\sqcup$ `HinoNacional`;
  - `Pop` $\sqsubseteq$ `Cancao`;
  - `EDM` $\sqsubseteq$ `Cancao`;
  - `Pessoa` $\equiv$ `Cantor` $\sqcup$ `Compositor`.
- As propriedades:
  - **`canta`**, com domínio `Cantor` e contradomínio `Musica`;
  - **`cantadaPor`**, inversa da propriedade acima.
- As instâncias e asserções:
  - `MarinaAndTheDiamonds`, instância da classe `Cantor`;
  - `Primadonna`, instância da classe `Pop`;
  - `MarinaAndTheDiamonds` **`canta`** `Primadonna`;
  - `Froot`, instância da classe `Pop`;
  - `MarinaAndTheDiamonds` **`canta`** `Froot`;
  - `Disconnect`, instância da classe `EDM`;

[1] https://protege.stanford.edu/about.php

– `MarinaAndTheDiamonds` **`canta`** `Disconnect`.

No *Protégé*, a ontologia fica como na Figura 5.2.



(a) A hierarquia das classes.

(b) As propriedades da ontologia.

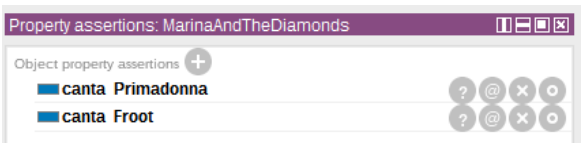
(c) As instâncias definidas.

Figura 5.2: A representação da ontologia do *Protégé*

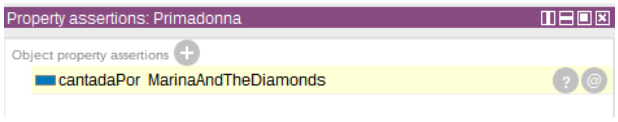
5.2.1 Raciocinadores

Um aspecto interessante sobre esse arcabouço é o uso de *plug-ins* de *raciocínio*, como o HermiT [2]. A partir deles, é possível fazer inferências a partir dos axiomas lógicos que foram criados.

No fragmento da ontologia, temos que `MarinaAndTheDiamonds` **`canta`** `Primadonna`. É possível inferir que `Primadonna` **`cantadaPor`** `MarinaAndTheDiamonds`. O *Protégé* consegue fazer isso com o auxílio do raciocinador, como exibido na Figura 5.3.



(a) Músicas que `MarinaAndTheDiamonds` canta.



(b) Com o raciocinador, é mostrado quem canta Primadonna. O destaque indica inferência.

Figura 5.3: A asserção feita na construção da ontologia e a inferência feita.

5.2.2 Buscas

A partir de uma ontologia e de uma base de dados, é possível fazer buscas neste arcabouço, utilizando, entre várias linguagens de busca, o SPARQL.

[2] http://www.hermit-reasoner.com/

SPARQL é um acrônimo recursivo para SPARQL *Protocol and RDF Query Language* (Beckett, 2011). Ele tem uma sintaxe que lembra a do SQL, e com seu uso, é possível tratar a ontologia como uma base de dados.

Vale notar que as buscas funcionam apenas com a terminologia e as asserções gravadas no arquivo da ontologia. Para que as buscas funcionem sobre as inferências, é necessário exportá-las para um novo arquivo.

No exemplo utilizado, temos que `MarinaAndTheDiamonds` canta música de dois ritmos diferentes. Para descobrir quais são as músicas `Pop` que ela canta, podemos rodar uma consulta, como na mostrada na Figura 5.4.

```
SELECT ?musica
WHERE { onto:MarinaAndTheDiamonds onto:canta ?musica.
        ?musica a onto:Pop. }
```

(a) Fragmento da consulta para a pergunta. Note que o prefixo `onto` refere-se a definições dessa ontologia.

musica
Primadonna
Froot

(b) A consulta gera uma tabela com as músicas do ritmo `Pop` que `MarinaAndTheDiamonds` canta.

Figura 5.4: A consulta feita e seu resultado.

Capítulo 6

Implementação do *plug-in*

O *plug-in* construído para este trabalho é, na verdade, uma reunião de implementações realizadas em trabalhos anteriores. As operações cobertas pelo *plug-in* e suas respectivas construções anteriores são:

Contração O *plug-in* possui os dois construtores descritos neste trabalho, as Contrações *Partial Meet* e a *Kernel*. Ambas foram baseadas em implementações de algoritmos feitas na tese de doutorado de Cóbe (Cóbe, 2014).

Revisão *Kernel* Para essa operação, os códigos feitos por Resina para o seu trabalho de mestrado (Resina, 2010) foram reconstruídos, com o auxílio dos estudos de Ribeiro (Ribeiro e Wassermann, 2008).

Pseudocontração SRW Essa operação foi completamente absorvida ao *plug-in* do código feito por Matos, em seu trabalho de conclusão de curso. (Matos e Wassermann, 2016)

Todo o código-fonte está disponível no GitHub, assim como os arquivos compilados e instruções de compilação e execução.

O programa consiste em uma interface com o usuário pelo terminal. Ele recebe os parâmetros pela linha de comando. A saída é um arquivo OWL com uma ontologia que pode ser aberto no *Protégé*. As informações de entrada e saída podem ser acessadas no README do projeto.

6.1 Desenvolvimento

O projeto foi inteiramente feito no IntelliJ versão 2018.2 [1]. Esse *software* não é gratuito, mas oferece uma versão de uso para estudantes. As dependências foram gerenciadas pelo *Apache Maven* 3.5.2 [2].

[1] https://www.jetbrains.com/idea/
[2] https://maven.apache.org/

Para as lógicas internas, são necessários a *OWL API* 5.1.6 [3], usada para todo o trabalho com os axiomas e o *HermiT Reasoner* [4]1.3.8, um motor de inferências. Para facilitar a entrada dos parâmetros, foi utilizado o *JCommander* 1.58 [5].

6.2 Algoritmo *BlackBox*

A implementação das quatro operações tem a sua parte principal em um algoritmo *BlackBox*, baseados nos estudos de Resina, onde tiveram suas corretudes provadas (Resina, 2010). Ele é chamado assim pois não precisa de um motor de inferência. É necessário apenas decidir se uma base de conhecimento implica certo axioma, ou se a base é consistente. Ele possui algumas variações, para o cálculo do conjunto-resíduo para a Contração *Partial Meet*, do conjunto-*kernel* para a Contração *Kernel* e do conjunto-*kernel* da Revisão *Kernel*.

Para efeitos de otimização, foi colocado um limite tanto nas filas utilizadas nos algoritmos abaixo quanto nos conjuntos que serão retornados. Tais limites podem ser definidos pelo usuário, opcionalmente.

6.2.1 *BlackBox* para o conjunto-resíduo

Para o cálculo do conjunto-resíduo, utilizado na contração *Partial Meet* e na Pseudocontração SRW, foi utilizada a implementação adaptada por Matos da implementação original de Cóbe (Matos e Wassermann, 2016). Ela consiste em duas funções:

REMAINDERBLACKBOX É uma função que recebe um conjunto de axiomas B, uma sentença A e um conjunto de axiomas X, tal que $X \nvdash A$, e constrói um elemento do conjunto resíduo $(B \perp A)$, que contém todos os elementos de X. O conjunto devolvido X' é tal que $X' \subseteq X \in B \perp A$. O método começa com X e acrescenta todos os axiomas de B que não façam o conjunto resultante implicar A. De acordo como o laço principal é implementado, resultados diferentes podem ser alcançados. No entanto, isso não é um problema, porque a função que chama esta só pede um item do conjunto-resíduo. O seu código segue abaixo:

```
function REMAINDERBLACKBOX(B, A, X)
    X' ← X
    for all β ∈ B \ X do
        if X' ∪ {β} ⊬ A then
            X' ← X' ∪ {β}
        end if
```

[3] http://owlcs.github.io/owlapi/
[4] http://www.hermit-reasoner.com/
[5] http://jcommander.org

```
    end for
    return X'
end function
```

REMAINDERSET Usando a função acima, ela constrói o conjunto-resíduo. Seja X um conjunto, tal que $X \nvdash A$, inicialmente vazio. Implicitamente, uma árvore é construída. Sua raiz é um elemento do conjunto-resíduo obtido a partir de X, e para cada axioma s fora desse conjunto, se $X \cup \{s\} \nvdash A$, o algoritmo cria um nó filho na árvore com um conjunto-resíduo obtido a partir de $X' = X \cup \{s\}$. Como se deseja apenas gerar o conjunto, a árvore não é construída por completo. Ao invés disso, os elementos são criados como se a árvore fosse percorrida por uma busca em largura, por isso o uso da fila. O seu código segue abaixo:

```
function REMAINDERSET(B, A)
    fila ← fila vazia
    S ← REMAINDERBLACKBOX(B, A, ∅)
    remainder ← {S}
    for all s ∈ B \ S do
        coloque s em fila
    end for
    while fila não está vazia do
        Hn ← o próximo de fila
        if Hn ⊬ A then
            S ← REMAINDERBLACKBOX(B, A, Hn)
            remainder ← remainder ∪ {S}
            for all s ∈ B \ S do
                coloque Hn ∪ {s} em fila
            end for
        end if
    end while
    return remainder
end function
```

6.2.2 *BlackBox* para o conjunto-*kernel* para a contração

Para o cálculo do conjunto-*kernel*, usado na contração *Kernel*, foi utilizada uma adaptação implementação do código de Cóbe (Matos e Wassermann, 2016). Ela consiste em duas funções:

KERNELBLACKBOX É uma função que recebe um conjunto de axiomas B e uma sentença A. Ela constrói um conjunto minimal de B que implica A. Essa função é uma adaptação

da estudada por Resina (Resina, 2010), e se divide em duas partes: expansão, onde são adicionam todos os axiomas de B, caso $B \vdash A$, a um conjunto B', definido previamente como vazio; e encolhimento, onde cada elemento β de B' é analisado individualmente. Caso $B' \setminus \{\beta\}$ implique A, β é apagado de B'. Logo, o conjunto devolvido é um conjunto minimal de B que implica A, portanto, um elemento do Kernel. Seu código é o seguinte:

```
function KernelBlackBox(B, A)
    B' ← ∅
    if B ⊢ A then
        B' ← B
    end if
    for all β ∈ B' do
        if B' \ {β} ⊢ A then
            B' ← B' ∪ {β}
        end if
    end for
end function
```

KernelSet É uma função que recebe os mesmos parâmetros citados acima, e utiliza a função supracitada. Ela começa com um elemento do Kernel. Assim como o construtor do conjunto-resíduo, ele constrói uma árvore, fazendo percorrendo em largura. Para cada elemento Hn da fila, o algoritmo o remove de B e, se B ainda implica A, computa-se o menor subconjunto S de $B \setminus Hn$ que implica A, e então, tem-se em S um novo elemento do Kernel. Depois disso, B é restaurado. Seu código está abaixo:

```
function KernelSet(B, A)
    if B ⊬ A then
        return ∅
    end if
    fila ← fila vazia
    S ← KernelBlackBox(B, A)
    kernel ← {S}
    for all s ∈ S do
        coloque s em fila
    end for
    while fila não está vazia do
        Hn ← o próximo de fila
        B ← B \ Hn
        if B ⊢ A then
            S ← KernelBlackBox(B, A)
```

```
            kernel ← kernel ∪ {S}
            for all s ∈ S do
                coloque Hn ∪ {s} em fila
            end for
        end if
        B ← B ∪ Hn
    end while
    return kernel
end function
```

6.2.3 *BlackBox* para o conjunto-*kernel* para a revisão

O cálculo do conjunto-*kernel* da Revisão *Kernel* é feito a partir de uma construção feita em 2008 (Ribeiro e Wassermann, 2008). Assim como os outros, consiste em duas funções:

REVISIONKERNELBLACKBOX É uma função que recebe o resultado de uma outra operação: a Expansão. Não é importante, no início, se o conjunto de entrada é consistente. Seu funcionamento é semelhante com os BLACKBOX anteriores. Possui uma parte de expansão e outra de encolhimento. A única diferença aqui, é que em vez de verificar se $B' \vdash \alpha$ (B' um conjunto inicialmente vazio), checa-se se B' é consistente/coerente. A função devolve um conjunto minimal inconsistente. O algoritmo segue abaixo:

```
function REVISIONKERNELBLACKBOX(B + α)
    B' ← ∅
    for all β ∈ B + α do
        B' ← B' ∪ {β}
        if B' é inconsistente/incoerente then
            Pare
        end if
    end for
    for all ϵ ∈ B' do
        if B' \ {ϵ} é inconsistente/incoerente then
            B' ← B' \ {ϵ}
        end if
    end for
    return B'
end function
```

REVISIONKERNELSET Essa função também recebe o resultado da Expansão $B + \alpha$. Diferentemente do conjunto resíduo, aqui a árvore é construída em profundidade. Como todos os algoritmos recursivos, possui um caso base e um geral. O caso base é quando

o conjunto de entrada é consistente, e não há nada a ser feito. Para o caso geral, ela chama a função definida acima, e para todos os elementos do conjunto definido abaixo como S, ele remove um dos elementos e tenta achar o conjunto de subconjuntos minimais inconsistentes para essa nova base. Seu código está abaixo:

function REVISIONKERNELSET($B + \alpha$)
  if $B + \alpha$ é consistente/coerente **then**
    return $\varnothing$
  end if
  $S \leftarrow$ REVISIONKERNELBLACKBOX($B + \alpha$)
  $B' \leftarrow B' \cup \{S\}$
  for all $s \in S$ **do**
    $B' \leftarrow B' \cup$ REVISIONKERNELSET($B + \alpha \setminus \{\beta\}$)
  end for
  return B'
end function

6.3 Funções de seleção e incisão

Duas das operações precisam de uma função de seleção γ. A implementação é flexível, a partir de uma interface. A função que está codificada aqui, foi trazida de um trabalho anterior (Matos e Wassermann, 2016), e devolve apenas um elemento do conjunto-resíduo.

As outras duas operações necessitam de uma função de incisão σ. A implementação foi feita da mesma maneira, só que aqui existem dois métodos diferentes para essa função. O primeiro é parecida com a da função γ, ou seja, devolve apenas um elemento do conjunto-*kernel*. Já o segundo devolve a união dos elementos do conjunto-*kernel*. Essa última é a implementação que vai no padrão do *plug-in*.

6.4 Exemplos da execução

São feitos aqui alguns exemplos simples para ilustrar a funcionalidade do *plug-in*. Todos os exemplos estão relacionados à ontologia discutida neste trabalho.

6.4.1 Contração *Kernel*

Para a contração *Kernel*, será usado um exemplo análogo. As classes estudadas aqui são `Cancao`, `Pop` e `SynthPop`. Os axiomas da ontologia são os listados abaixo, e sua representação está na Figura 6.1.

- SynthPop $\sqsubseteq$ Pop
- Pop $\sqsubseteq$ Cancao

Suponha que o subgênero SynthPop evoluiu e está distante do Pop, mas ainda mantém o nome. Portanto, não se deseja mais que ele seja uma subclasse de Pop.

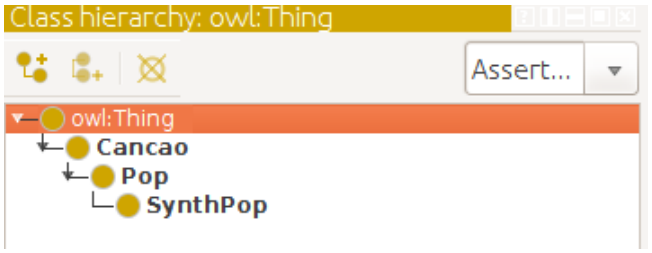


Figura 6.1: As classes da ontologia de gêneros musicais.

Com o comando abaixo, a operação é realizada. O resultado está na Figura 6.2.

```
java -jar ontologyrepair-1.0.0-SNAPSHOT.jar -c --core-retainment -i cancao.owl
-o output.owl -f "SynthPop SubClassOf Pop"
```

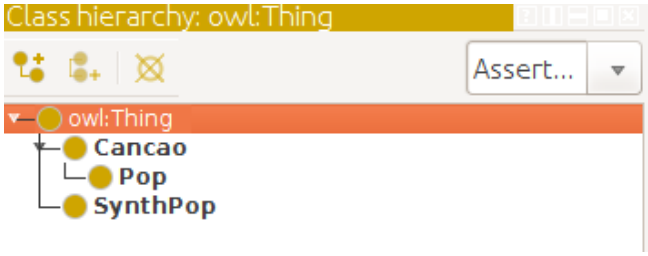


Figura 6.2: O resultado da operação Contração *Kernel*.

6.4.2 Contração *Partial Meet*

Para a contração *Partial Meet*, o exemplo utilizado tem as classes Musica, Cancao e HinoNacional, e os seguintes axiomas:

- HinoNacional $\sqsubseteq$ Cancao
- Cancao $\sqsubseteq$ Musica

Sua representação fica como na Figura 6.3.

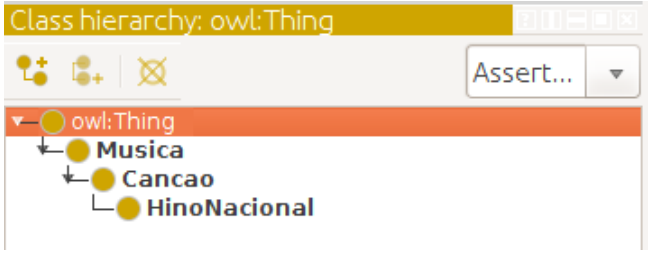


Figura 6.3: A estrutura da ontologia de tipos de música.

Como definido anteriormente, na verdade, `Cancao` e `HinoNacional` são classes na mesma hierarquia, portanto, precisamos remover `HinoNacional` ⊑ `Cancao`. A operação é realizada com o seguinte comando:

```
java -jar ontologyrepair-1.0.0-SNAPSHOT.jar -c --relevance -i musica.owl
-o output.owl -f "HinoNacional SubClassOf Cancao"
```

Após a sua execução, acontece o desejado, mas aparentemente, esse não é o melhor resultado, como pode ser visto na Figura 6.4.

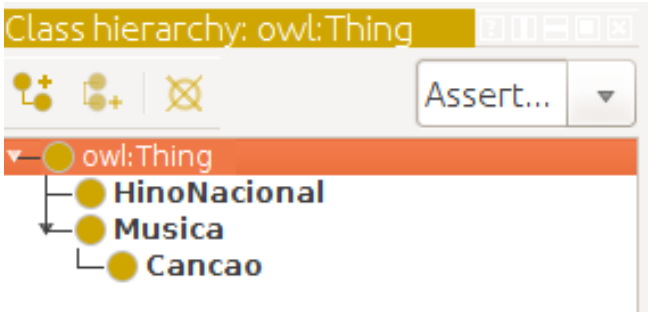


Figura 6.4: O resultado da operação Contração *Partial Meet*.

6.4.3 Revisão *Kernel*

O exemplo usado para a Revisão *Kernel* será simples também. Sejam um fragmento da ontologia de música os axiomas abaixo e representação como na Figura 6.5.

- `FunkAmericano` ⊑ `Cancao`
- `FunkBrasileiro` ⊑ `FunkAmericano`

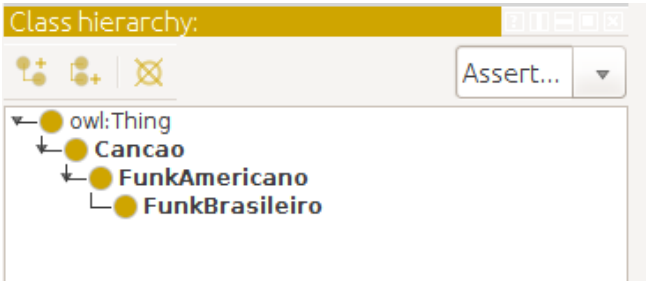


Figura 6.5: A estrutura da outra ontologia de gêneros musicais.

No entanto, hoje já se classifica o Funk Brasileiro como distinto do Funk Americano. Logo, vamos adicionar à nossa base que os dois subgêneros são disjuntos.

Com o comando abaixo, acontece a operação:

```
java -jar ontologyrepair-1.0.0-SNAPSHOT.jar -r -i funk.owl -o output.owl
-f "FunkBrasileiro DisjointWith FunkAmericano"
```

O resultado é visto abaixo. Note que a propriedade de disjunção está no produto final. Ambos os produtos podem ser observados na Figura 6.6.

Figura 6.6: O resultado da operação Revisão *Kernel* e a presença da disjunção, na classe `FunkAmericano`, respectivamente.

6.4.4 Pseudocontração SRW

Para a Pseudocontração SRW, serão consideradas as classes `Cantor` e `Compositor` da ontologia musical, e o seguinte axioma: `Compositor` $\sqsubseteq$ `Cantor`, com a representação exibida na Figura 6.7.

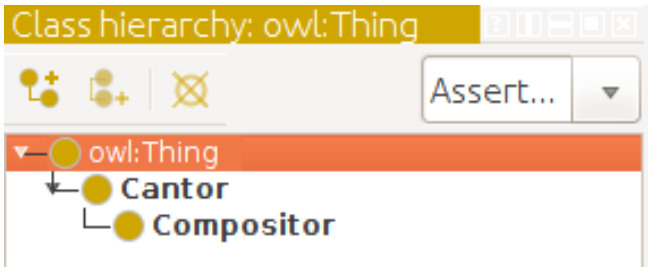


Figura 6.7: A estrutura da ontologia sobre ocupações na área de música.

Ela possui um indivíduo, `markRonson`, pertencente à classe `Compositor`, como se mostra na Figura 6.8.

Figura 6.8: A instância da ontologia.

Tudo isso implica que `markRonson` também é um `Cantor`, o que está errado.

A operação pode ser feita com o seguinte comando:

```
java -jar ontologyrepair-1.0.0-SNAPSHOT.jar -srw -i pessoa.owl -o output.owl
-f "markRonson Type: Cantor"
```

Depois da operação, temos que `Compositor` deixa de ser subclasse de `Cantor`, e então, `markRonson` não pertence mais à classe `Cantor`, como se observa na Figura 6.9.

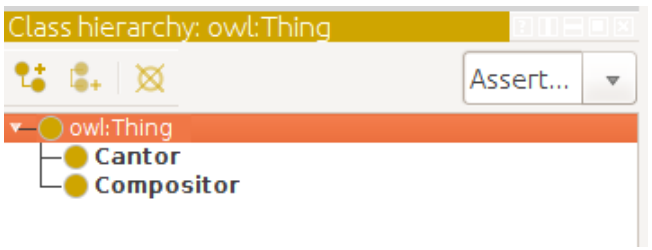


Figura 6.9: O resultado da operação Pseudocontração SRW.

Capítulo 7

Análise de Desempenhos

PARA entender como o *plug-in* se comporta em situações diferentes, foram feitos testes. Ele se dividem em dois tipos: os de performance, que visam verificar em quanto tempo o *plug-in* realiza as quatro operações implementadas; e os de comparação, que foram executados para ver os diferentes resultados que as operações podem fornecer, para condições semelhantes.

7.1 Testes de Performance

Para os testes de performance, foi realizada uma pequena alteração no *plug-in*. Ao se escrever o argumento `-t`, o programa realiza a operação desejada 1000 vezes, e devolve, além do arquivo de saída, a média de tempo de execução em milissegundos e um intervalo de confiança de 95% para essa média.

Foram utilizados quatro arquivos diferentes, todos extraídos do repositório [1] do aluno Raphael M. Cóbe, feitos para seu doutorado. Os arquivos foram os seguintes:

- `SmallKernelWith5classes.owl`: que como o nome já diz, possui 5 classes. Além disso, ele possui 18 triplas, que são usadas para fornecer significado básico para os termos da linguagem W3C (2012b). Exemplo: `Cantor canta Musica`;
- `SmallKernelWith10classes.owl`: com 10 classes e 33 triplas;
- `SmallKernelWith20classes.owl`: que possui 20 classes e 63 triplas;
- `SmallKernelWith30Classes.owl`: que tem 30 classes e 93 triplas.

As medidas número de classes e número de triplas podem ser consideradas métricas para uma ontologia.

Para cada um desses quatro arquivos, foram executadas as quatro operações. Para a Contração *Kernel*, Contração *Partial Meet* e para a Pseudocontração, a fórmula trabalhada

[1] https://github.com/raphaelmcobe/ontology-debug-and-repair

foi `"A SubClassOf B1"`. Para todas as operações, apenas esse axioma foi removido, sem nenhuma alteração a mais na base, sendo esse um resultado satisfatório.

Para a Revisão Kernel, a fórmula usada foi `"B1 DisjointWith C"`. Em todos os arquivos utilizados, todas as outras classes foram apagadas e apenas essas duas permaneceram como classes irmãs.

7.1.1 Contração *Kernel*

Para a contração *Kernel* foram utilizados os seguintes argumentos:

- `-t -c --core-retainment -i SmallKernelWith5classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -c --core-retainment -i SmallKernelWith10classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -c --core-retainment -i SmallKernelWith20classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -c --core-retainment -i SmallKernelWith30classes.owl -o output.owl -f "A SubClassOf B1"`

Foi gerado um gráfico, na Figura 7.1, que mostra a evolução do tempo de execução médio. Estão tabulados na Tabela 7.1, além das médias, os intervalos de confiança.

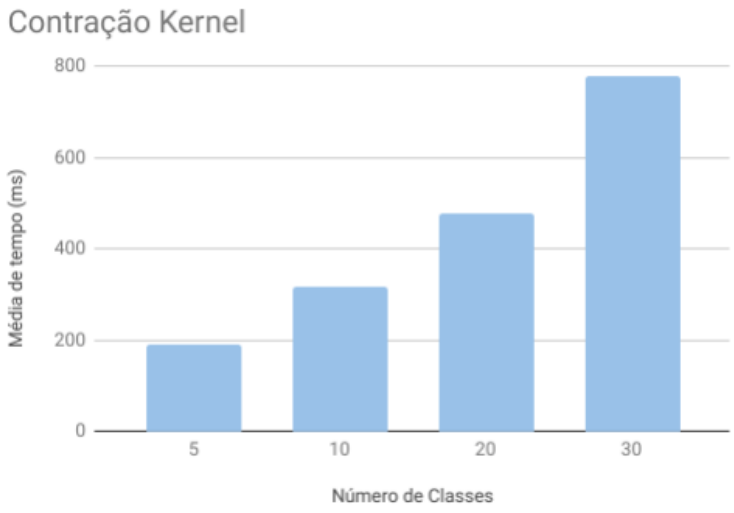


Figura 7.1: A evolução do tempo de execução da Contração *Kernel* comparada com o número de classes da ontologia.

Nº de classes	Tempo médio de execução (ms)	Intervalo de Confiança
5	190,238	[181,7, 198,8]
10	317,126	[311,5, 322,8]
20	476,483	[472,0, 481,0]
30	777,018	[764,8, 789,2]

Tabela 7.1: Resultados da performance da Contração *Kernel* com arquivos de tamanhos diferentes

7.1.2 Contração *Partial Meet*

Para a contração *Partial Meet* foram utilizados os seguintes argumentos:

- `-t -c --relevance -i SmallKernelWith5classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -c --relevance -i SmallKernelWith10classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -c --relevance -i SmallKernelWith20classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -c --relevance -i SmallKernelWith30classes.owl -o output.owl -f "A SubClassOf B1"`

Como na subseção anterior, foi gerado um gráfico, na Figura 7.2, que mostra a evolução do tempo de execução médio. Na tabela 7.2 estão presentes as médias e os intervalos de confiança.

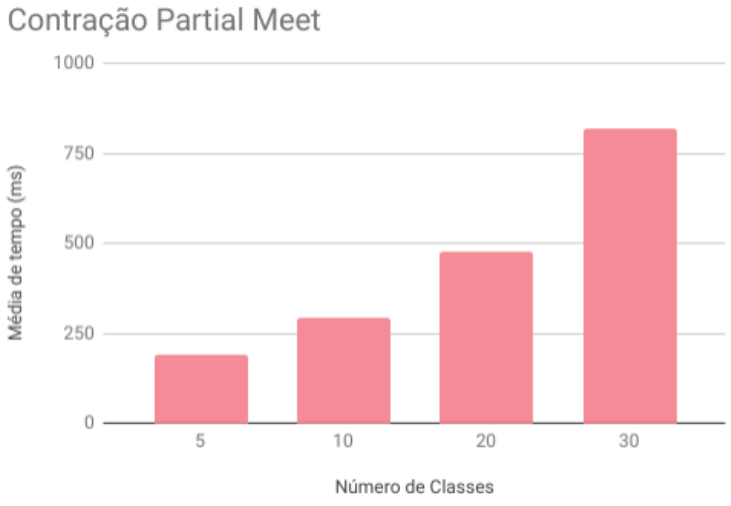


Figura 7.2: A evolução do tempo de execução da Contração *Partial Meet* comparada com o número de classes da ontologia.

Nº de classes	Tempo médio de execução	Intervalo de Confiança
5	190,890	[186,8, 195,0]
10	294,866	[290,7, 299,0]
20	476,362	[471,3, 481,4]
30	818,999	[809,1, 828,9]

Tabela 7.2: Resultados da performance da Contração *Partial Meet* com arquivos de tamanhos diferentes.

7.1.3 Pseudocontração SRW

Para a Pseudocontração SRW foram utilizados os seguintes argumentos:

- `-t -srw -i SmallKernelWith5classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -srw -i SmallKernelWith10classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -srw -i SmallKernelWith20classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -srw -i SmallKernelWith30classes.owl -o output.owl -f "A SubClassOf B1"`

Há um gráfico, na Figura 7.3, que mostra a evolução do tempo de execução médio. Na tabela 7.3 as médias e os intervalos de confiança podem ser vistos.

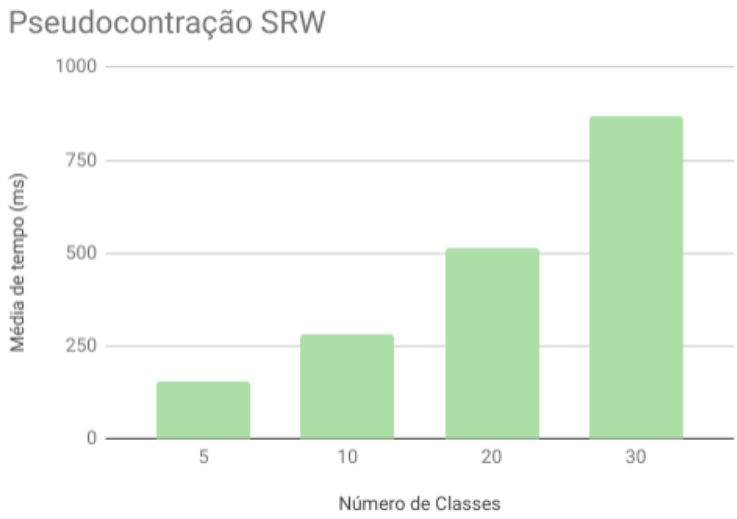


Figura 7.3: A evolução do tempo de execução da Pseudocontração SRW comparada com o número de classes da ontologia.

Nº de classes	Tempo médio de execução	Intervalo de Confiança
5	156,114	[153,6, 158,6]
10	282,843	[279,4, 286,3]
20	511,838	[504,8, 518,9]
30	867,095	[851,0, 883,2]

Tabela 7.3: Resultados da performance da Pseudocontração SRW com arquivos de tamanhos diferentes.

7.1.4 Revisão *Kernel*

Para a Revisão *Kernel* foram utilizados os seguintes argumentos:

- `-t -r -i SmallKernelWith5classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -r -i SmallKernelWith10classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -r -i SmallKernelWith20classes.owl -o output.owl -f "A SubClassOf B1"`
- `-t -r -i SmallKernelWith30classes.owl -o output.owl -f "A SubClassOf B1"`

Na Figura 7.4 pode ser vista a evolução do tempo de execução médio. Na tabela 7.4 estão presentes as médias e os intervalos de confiança.

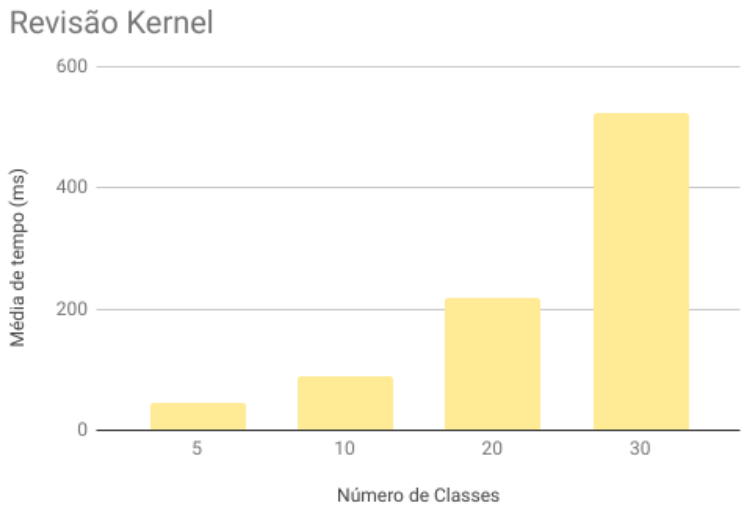


Figura 7.4: A evolução do tempo de execução da Revisão *Kernel* comparada com o número de classes da ontologia.

Nº de classes	Tempo médio de execução	Intervalo de Confiança
5	44,146	[42,2, 46,1]
10	87,953	[85,4, 90,5]
20	218,807	[215,1, 222,5]
30	522,843	[516,1, 529,6]

Tabela 7.4: Resultados da performance da Revisão *Kernel* com arquivos de tamanhos diferentes.

7.2 Comparação

Esses testes são, na verdade, a execução de diferentes operações sobre fórmulas próximas. O objetivo é verificar se existem diferenças nos resultados que elas fornecem. Os exemplos utilizados são bem curtos, afim de que se possa rapidamente enxergar as discrepâncias.

7.2.1 Comparação das subclasses de Pessoa

Para essa comparação, o exemplo da subseção 6.4.4 será revisitado, sem nenhuma alteração. Os resultados obtidos foram bem interessantes.

A execução da Contração *Kernel*, com a mesma fórmula, forneceu na resposta a mesma estrutura de classes da entrada, como pode ser visto na Figura 7.5. Foi mantido o axioma `Compositor(markRonson)`, como pode se observar na Figura 7.6. A classe `Cantor` ficou sem a instância, algo exibido pela Figura 7.7.

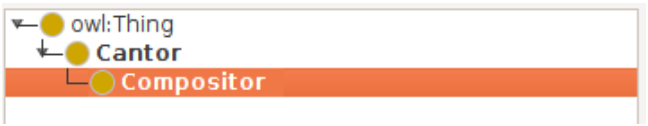


Figura 7.5: A estrutura de classes foi mantida.

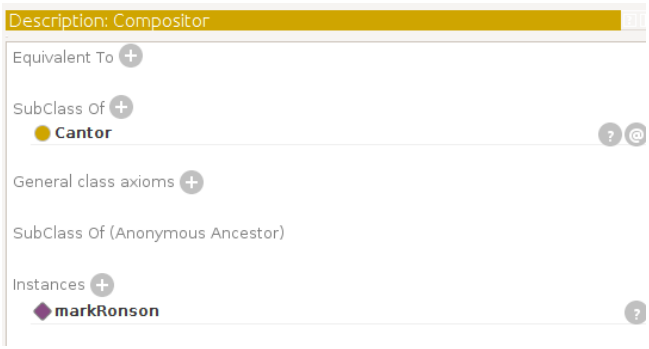


Figura 7.6: A instância `markRonson` continua na classe `Compositor`.

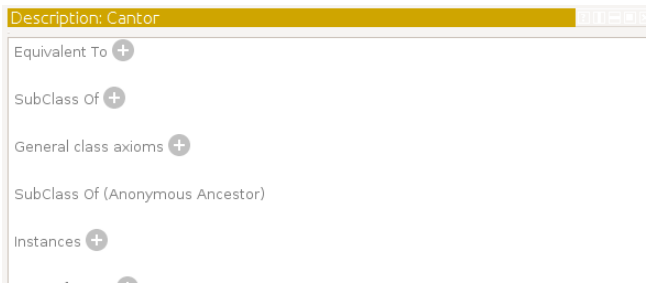


Figura 7.7: A classe `Cantor` fica sem nenhuma instância.

A Contração *Partial Meet*, também pela mesma fórmula, tira a instância `markRonson` de ambas as classes, mas mantém a estrutura das classes. A Pseudocontração SRW, como já é sabido, também tira a instância das duas classes e altera a hierarquia das classes.

A Revisão foi feita pela fórmula `"Cantor DisjointWith Compositor"`. O resultado obtido foi idêntico ao que a Pseudocontração SRW forneceu.

7.2.2 Comparação de classes em `HinoNacional`

Para essa comparação, foi feito um novo exemplo, com os seguintes axiomas: `MusicaBelica` ⊑ `HinoNacional` e `Marcha` ⊑ `MusicaBelica`. Sua hierarquia pode ser observada na Figura 7.8.

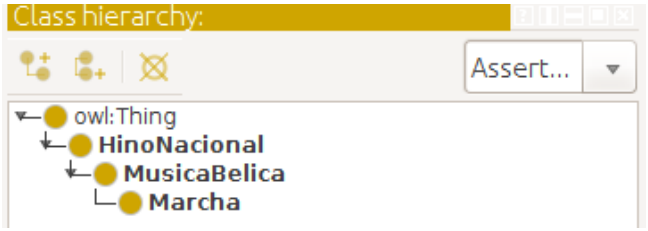


Figura 7.8: A hierarquia de classes da ontologia.

As fórmulas utilizadas para a Contração *Kernel*, Contração *Partial Meet* e Pseudocontração SRW foram a mesma `"Marcha SubClassOf HinoNacional"`. As operações devolveram resultados diferentes.

Tanto a Contração *Kernel* quanto a Pseudocontração SRW forneceram o mesmo resultado, deixando todas as classes como irmãs, como exibido na figura 7.9.

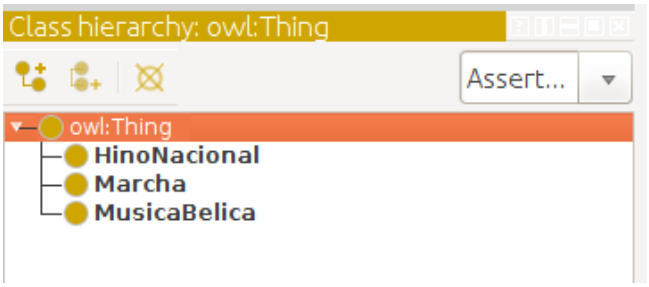


Figura 7.9: A estrutura de classes após as operações Contração *Kernel* e Pseudocontração SRW.

Já a Contração *Partial Meet* fornece um resultado mais brando, mantendo o axioma `Marcha` ⊑ `MusicaBelica`. Esse resultado parece ser mais promissor do que o anterior. Na Figura 7.10, é possível observar como fica a hierarquia.

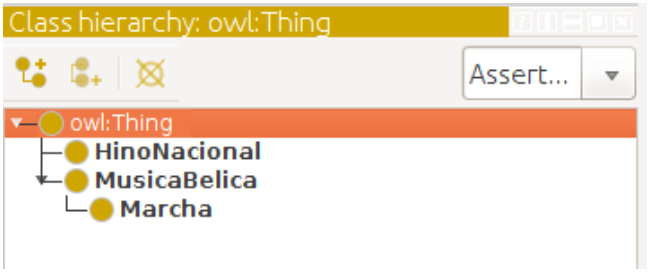


Figura 7.10: Hierarquia das classes após a Contração *Partial Meet*.

No outro oposto, temos a Revisão *Kernel*. Seu resultado é o mais radical. A fórmula utilizada foi `"Marcha DisjointWith HinoNacional"`. Pode-se observar, na Figura 7.11, que

esse axioma está presente na ontologia. No entanto, os outros dois axiomas não estão mais na ontologia, e a classe `MusicaBelica` também não existe mais na estrutura. Algo semelhante havia ocorrido nos testes de performance.

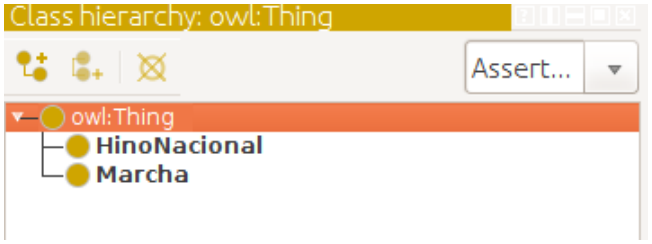


Figura 7.11: A estrutura da ontologia, radicalmente alterada após a Revisão *Kernel*.

Agradecimentos e parte subjetiva

ANTES de tudo, gostaria de retribuir àqueles que tornaram a existência desse trabalho possível. Primeiramente, agradeço a Alan Turing, considerado o pai da Ciência da Computação e a Ada Lovelace, conhecida como a primeira programadora da história.

Sou grato também a todos os professores do Instituto de Matemática e Estatística, em especial os do Departamento Ciência da Computação com os quais tive aula, alguns citados aqui: A. Fujita; A. Mandel; A. Melo; C. Ferreira; D. Batista; E. Birgin; J. Barrera; J. de Pina Jr.; J. Ferreira; M. Gubitoso; P. Feofiloff; W. Mascarenhas e Y. Kohayakawa.

Gostaria de dar destaque para aqueles que ajudaram na minha ênfase em Inteligência Artificial: F. Kon; L. Barros; N. Hirata e R. Wassermann, a orientadora deste trabalho.

Reconheço também a ajuda dos alunos de pós-graduação em Computação do IME-USP: F. Resina e V. Matos, que me deram grande suporte durante a execução deste.

Agradeço muito à minha família. Sem a assistência que eles me forneceram durante toda a minha vida, eu não teria chegado até aqui.

Este trabalho não é parecido com nada que eu já fiz. Foi a primeira vez que eu fiz um estudo teórico mais profundo sobre uma área tão específica. As leituras começaram com assuntos que eu já tinha noções, e foram evoluindo para matérias cujo meu conhecimento era quase nulo. A parte teórica me surpreendeu, pois foi a que tive mais prazer em realizar.

A parte prática também me deixou pasmo, mas por outro motivo. A princípio, achei que teria problemas com a parte teórica, mas aconteceu o contrário. Mesmo trabalhando com ferramentas já conhecidas, tive dificuldade em conseguir fazer o projeto.

Em suma, acredito que saio outra pessoa depois da entrega deste trabalho. Aprendi muito durante sua execução, não apenas sobre o seu tema, mas tudo relacionado, como pesquisa, leitura, escrita e desenvolvimento. Além disso, existe a experiência relacionada ao meu ritmo de trabalho, que com certeza irei me lembrar para sempre.

Referências Bibliográficas

Alchourrón *et al.***(1985)** Carlos E Alchourrón, Peter Gärdenfors e David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The journal of symbolic logic*, 50(2):510–530. Citado na pág.

Baader *et al.***(2008)** Franz Baader, Ian Horrocks e Ulrike Sattler. Description Logics. Em Frank van Harmelen, Vladimir Lifschitz e Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 3, páginas 135–180. Elsevier. URL download/2007/BaHS07a.pdf. Citado na pág.

Bechhofer *et al.***(2004)** Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, Lynn Andrea Stein *et al.* Owl web ontology language reference, 2004. Citado na pág.

Beckett(2011) Dave Beckett. What does sparql stand for?, 2011. URL http://lists.w3.org/Archives/Public/semantic-web/2011Oct/0041.html. Online; accessed 12-julho-2018. Citado na pág.

Brachman e Schmolze(1988) Ronald J Brachman e James G Schmolze. An overview of the kl-one knowledge representation system. Em *Readings in Artificial Intelligence and Databases*, páginas 207–230. Elsevier. Citado na pág.

Cóbe(2014) Raphael Mendes de Oliveira Cóbe. *Integração entre múltiplas ontologias: reúso e gerência de conflitos*. Tese de Doutorado, Universidade de São Paulo. Citado na pág.

Dahlberg(1978) Ingetraut Dahlberg. Teoria do conceito. *Ciência da informação*, 7(2). Citado na pág.

França(2009) Patrícia Cunha França. Conceitos, classes e/ou universais: com o que é que se constrói uma ontologia? *Linguamática*, 1(1):105–121. Citado na pág.

Giaretta e Guarino(1995) Pierdaniele Giaretta e N Guarino. Ontologies and knowledge bases towards a terminological clarification. *Towards very large knowledge bases: knowledge building & knowledge sharing*, 25(32):307–317. Citado na pág.

Grau *et al.***(2008)** Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider e Ulrike Sattler. Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322. Citado na pág.

Gruber(1995) Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International journal of human-computer studies*, 43(5-6):907–928. Citado na pág.

Guarino(1998) Nicola Guarino. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, volume 46. IOS press. Citado na pág.

Gärdenfors(1988) Peter Gärdenfors. *Knowledge in flux: Modeling the dynamics of epistemic states.* The MIT press. Citado na pág.

Gärdenfors(2003) Peter Gärdenfors. *Belief revision*, volume 29. Cambridge University Press. Citado na pág.

Gärdenfors(1992) Peter Gärdenfors. Belief revision: A vade-mecum. Em *International Workshop on Meta-Programming in Logic*, páginas 1–10. Springer. Citado na pág.

Hansson(1991) Sven Ove Hansson. Belief contraction without recovery. *Studia logica*, 50(2): 251–260. Citado na pág.

Hansson(1993a) Sven Ove Hansson. Reversing the levi identity. *Journal of Philosophical Logic*, 22(6):637–669. Citado na pág.

Hansson(1993b) Sven Ove Hansson. Changes of disjunctively closed bases. *Journal of Logic, Language and Information*, 2(4):255–284. Citado na pág.

Hansson(1994) Sven Ove Hansson. Kernel contraction. *The Journal of Symbolic Logic*, 59(3): 845–859. Citado na pág.

Hansson(1989) Svenove Hansson. New operators for theory change. *Theoria*, 55(2):114–132. Citado na pág.

Herman(2001) Ivan Herman. W3c semantic web frequently asked questions, 2001. URL https://www.w3.org/2001/sw/SW-FAQ. Online; accessed 12-julho-2018. Citado na pág.

Koubarakis(2010) Manolis Koubarakis. Tableau proof techniques for dls, 2010. URL http://cgi.di.uoa.gr/~pms509/lectures/tableaux-techniques1spp.pdf. Online; accessed 30-Junho-2018. Citado na pág.

Matos e Wassermann(2016) Vinícius Bitencourt Matos e Renata Wassermann. Implementação de um módulo de pseudocontração em revisão de crenças para o editor de ontologias protégé. Citado na pág.

Nickles *et al.***(2007)** Matthias Nickles, Adam Pease, Andrea C Schalley e Dietmar Zaefferer. Ontologies across disciplines. *Ontolinguistics-How Ontological Status Shapes the Linguistic Coding of Concepts*, páginas 23–67. Citado na pág.

Noy *et al.***(2001)** Natalya F Noy, Deborah L McGuinness *et al.* Ontology development 101: A guide to creating your first ontology, 2001. Citado na pág.

Resina(2010) Fillipe Manoel Xavier Resina. Revisão de crenças em lógica de descrição: Um plug-in para o protégé. Citado na pág.

Ribeiro e Wassermann(2008) Márcio Moretto Ribeiro e Renata Wassermann. The ontology reviser plug-in for protégé. Em *WONTO*. Citado na pág.

Ribeiro(2010) Márcio Moretto Ribeiro. *Revisao de crenças em lógicas de descrição e em outras lógicas não clássicas*. Tese de Doutorado, Universidade de São Paulo. Citado na pág.

Russel e Norvig(1995) Stuart J. Russel e Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Inc. Citado na pág.

Santos *et al.***(2015)** Yuri D Santos, Marcio M Ribeiro e Renata Wassermann. Between belief bases and belief sets: partial meet contraction. Em *Proceedings of the 2015 International Conference on Defeasible and Ampliative Reasoning-Volume 1423*, páginas 50–56. CEUR-WS. org. Citado na pág.

Schmidt-Schauß e Smolka(1991) Manfred Schmidt-Schauß e Gert Smolka. Attributive concept descriptions with complements. *Artificial intelligence*, 48(1):1–26. Citado na pág.

Van Harmelen *et al.***(2008)** Frank Van Harmelen, Vladimir Lifschitz e Bruce Porter. *Handbook of knowledge representation*, volume 1. Elsevier. Citado na pág.

W3C(2001) W3C. Web-ontology (webont) working group (closed), 2001. URL https://www.w3.org/2001/sw/WebOnt/. Online; accessed 12-julho-2018. Citado na pág.

W3C(2012a) W3C. Owl 2 web ontology language profiles (second edition), 2012a. URL https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/. Online; accessed 12-julho-2018. Citado na pág.

W3C(2007) W3C. Owl working group, 2007. URL https://www.w3.org/2007/OWL/wiki/OWL_Working_Group. Online; accessed 12-julho-2018. Citado na pág.

W3C(2012b) W3C. Owl 2 web ontology language, 2012b. URL https://www.w3.org/TR/owl-syntax/. Online; accessed 12-julho-2018. Citado na pág.

W3C(2004) W3C. World wide web consortium issues rdf and owl recommendations, 2004. URL https://www.w3.org/2004/01/sws-pressrelease. Online; accessed 12-julho-2018. Citado na pág.