Resumo

Abstract

Capítulo 1

Introdução

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 [23] 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 [6]. 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*"[1] [24].

Para a Ciência da Computação, a definição é um pouco diferente, embora possua muita semelhança com o conceito já explícito. Guarino [12] 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". Logo, uma ontologia seria 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 em o 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*". 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.

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. Atualmente, esta área é bastante pesquisada por sua aplicação com Web Semântica.

As ontologias denotam uma "especificação explícita de uma conceitualização"[11], e, uma vez construídas, permitem comunicação, compartilhamento e reúso de conhecimentos. Elas interessam várias áreas do conhecimento, tais como a ciência da computação, filosofia,

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

engenharia de dados, *data science*, lógica e linguística.

2.1 Definições preliminares

Smith [32] definiu alguns conceitos que nos auxiliam a compreender melhor o que é uma ontologia.

Representação Seria uma ideia, uma imagem, um registro, ou uma descrição que se refere, ou é sobre, ou pretende referir a alguma entidade externa à representação.

Representação Composta É uma representação constituída de sub-representações como as suas partes, da mesma maneira em que os parágrafos são feitos de sentenças e as sentenças de palavras. A menor sub-representação se chama unidade representacional, que pode ser um ícone, um símbolo, um nome, entre outros.

Representação Cognitiva É uma representação cujas unidades representacionais são ideias, pensamentos ou crenças na mente de algum sujeito pensante.

Artefato Representacional É uma representação fixa em algum meio de modo que pode servir para fazer representações cognitivas que existem em mentes separados. Exemplos: um texto, um diagrama, um mapa, uma lista, entre outros.

Porção de Realidade Termo que compreende universais únicos e particulares e suas combinações mais ou menos complexas.

Domínio É uma porção da realidade que forma o assunto de estudo de uma ciência ou tecnologia. Ex: Radiologia, História, Arqueologia, etc..

Taxonomia É um artefato representacional em um grafo na forma de árvore com os nós representando universais ou classes e as arestas sendo relações "*is-a*"ou "*part-of*", que serão tratadas mais adiante.

Ontologia É um artefato representacional que compreende uma taxonomia própria, cujas unidades representacionais são usadas para designar uma combinação de universais, classes definidas e relações entre eles.

2.2 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 [9], uma Conceitualização pode ser entendida como "uma estrutura semântica intensional que codifica as regras implícitas que determinam a estrutura de uma porção da realidade". 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 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.

Debruyne [7] fez um estudo mais sistemático sobre o que seria uma Conceitualização, baseado na pesquisa de Guarino [12]. Para facilitar o entendimento das definições que virão a seguir, é necessário entender dois conceitos vindos da Filosofia:

Contexto Intensional Ocorre quando um termo não pode ser substituído sem que a verdade sobre a sentença seja ameaçada. Por exemplo, embora saibamos que Clark Kent é o Super-Homem e também é um jornalista, a sentença a seguir é falsa, pois o Super-Homem não é um jornalista:

- Lois Lane acredita que o Super-Homem irá investigar uma notícia com ela.

Contexto Extensional Se refere à extensão de um termo, que é o conjunto de objetos que ele denota. Um contexto é extensional se alguma expressão e aparece e pode ser substituída por uma expressão e' com a mesma extensão mantendo o seu valor verdadeiro. Por exemplo:

- Violão é um instrumento musical.
- Piano é um instrumento musical.

Acima, temos que *instrumento musical* é uma extensão de *Violão* e de *Piano*.

Feito este esclarecimento, seguem as definições de Guarino.

Noção Extensional de Conceitualização É um par (D, R), onde D é um domínio e R são as relações relevantes que existem nesse domínio.

Espaço Domínio Um par (D, W), com D representando um domínio e W sendo os mundos possíveis dentro dele, ou seja, os conjuntos máximos de estados das coisas desse domínio.

Relação Conceitual Dado um espaço domínio (D, W), uma relação conceitual ρ^n de aridade n em (D, W) é definida como uma função total $\rho^n : W \rightarrow 2^{D^n}$ de W para o conjunto de todas as relações n-árias em D.

Noção Intensional de Conceitualização Uma Conceitualização para D é definida como uma tripla ordenada $C = (D, W, R)$, onde R é um conjunto de relações conceituais do espaço domínio (D, R).

Definindo uma Linguagem Lógica L com um Vocabulário V, teremos:

Interpretação Extensional de uma Linguagem Um modelo para L é definido como uma estrutura (S, I), onde $S = (D, R)$ é uma estrutura de mundo e $I : V \rightarrow D \cup R$ é uma função de interpretação atribuindo elementos de D para símbolos constantes de V e elementos de R para símbolos predicados de V.

Interpretação Intensional de uma Linguagem Por meio de uma estrutura (C, I), onde C é uma Conceitualização como definido no item 3, e I é uma função de interpretação como definido no item acima, teremos que uma interpretação intensional é um compromisso ontológico K para L.

Conjunto de modelos pretendidos de uma linguagem de acordo com um compromisso O conjunto $I_K(L)$ de todos os modelos pretendidos de uma linguagem L que são compatíveis com um compromisso $K = (C, I)$ são todos os modelos de L que são compatíveis com K. O compromisso garante consistência.

2.3 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 [11] 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. 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.

Coerência 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 é incoerente.

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 não poderão ser atendidos ao mesmo tempo, portanto, alguns *trade-offs* deverão ser feitos. Podemos ter um conflito, por exemplo, entre os critérios 1 e 3, já que o máximo de clareza implica que as definições terão a sua interpretação restrita.

2.3.1 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 `Cantor` possui as propriedades `Nome`, `RitmoPredominante` e `Idade`.

Relações Ontologias são constituídas de relações hierárquicas. Por exemplo `Cantor` $\rightarrow$ `Pessoa`. A hierarquia de classes representa uma relação "*is-a*"[8]. Tais relações são transitivas. Existem também as relações "*part-of*", que já estão dentro da Teoria dos Conjuntos. Uma classe é parte da outra se a compõe.

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.

Instância é um indivíduo (por exemplo, `MariahCarey`), de um conjunto universal (cantores, músicos, pessoas). Vale lembrar que uma instância não é uma subclasse.

Como já vimos acima, uma classe pode ter nomes diferentes, pois não é ele 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` e `MusicaAmericana`, e não `Cancao`, `MusicaAmericana` e `MusicaBrasileira` (a última é subclasse da penúltima).

Os conceitos definidos até aqui mostram um jeito de caracterizar ontologias, que é pelo número de termos que ela possui. Uma outra métrica é o número de axiomas lógicos que ela possui.

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 os ritmos gerais e depois aprofundar para os regionais.

Bottom-up Vai das classes mais específicas para as mais genéricas, agrupando as específicas já criadas. Na ontologia estudada aqui, seria possível começar dos ritmos regionais e depois agrupá-los por suas semelhanças.

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

2.3.2 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 [25]. 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.

Outro aspecto interessante é o de Classe Abstrata. Uma classe desse tipo não pode ter instâncias diretas. Em nossa ontologia, podemos ter classes de músicas regionais. A classe `Musica` pode ter varias subclasses, tais como `MusicaArgentina` e `MusicaBrasileira`, entre outras. Note que essas classes são abstratas, pois no mundo real, não existe mais de um tipo de Música Brasileira, como o conjunto de ritmos definido pelo senso comum, que possa pertencer ao conceito `MusicaBrasileira`.

As ontologias possuem suporte a 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 manter um padrão de nomenclatura é algo bom. Para fazer isso, pode-se usar `snake_case`, `camelCase` ou usar espaços na grafia. Além disso, deve-se prestar atenção se o nome de uma classe possui o nome de sua superclasse ou não.

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`
- `Musica cantadaPor Cantor`

2.4 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 limita a equipe de possuir profissionais de mais áreas. Uma ontologia é feita para que qualquer pessoa possa acessar suas informações.

É possível achar os usuários de uma ontologia, mas nem sempre o seu autor. O razoável é assumir que elas foram evoluindo com o passar do tempo.

Ontologias feitas sobre áreas de estudo (por exemplo, a Aviação) serão 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. Seria melhor, por exemplo, construir uma ontologia sobre acidentes de avião e seus conceitos.

2.5 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.

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 construída nesse capítulo fica assim:

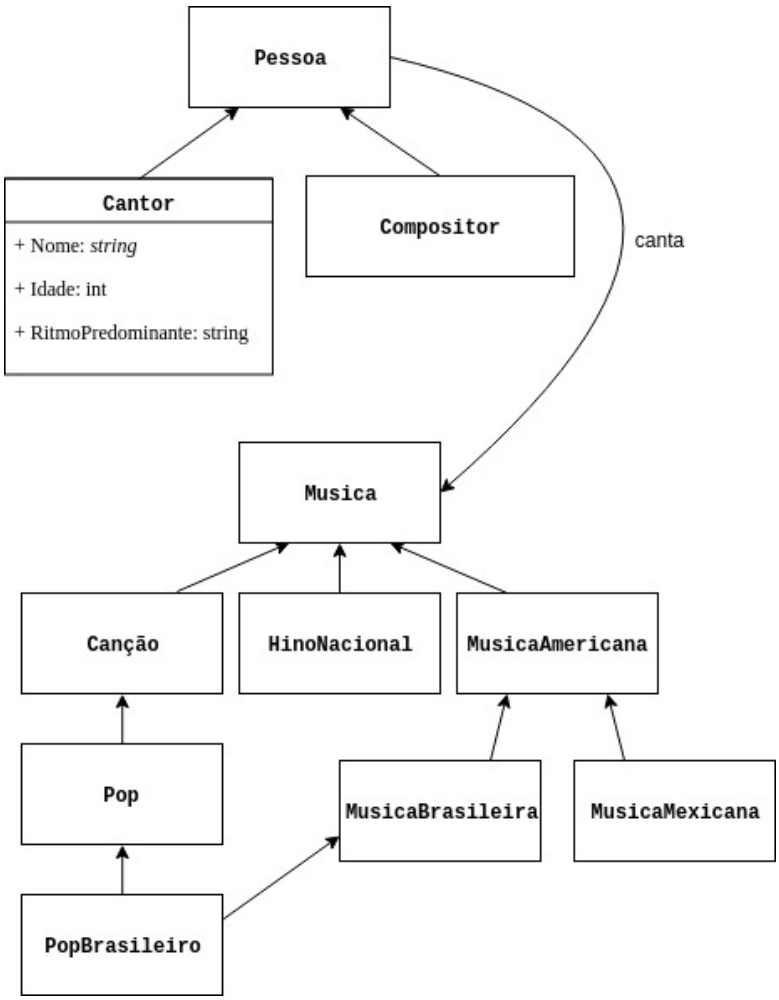


Figura 2.1: Esquema de classes de uma ontologia sobre Música

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:

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 LD possuem um jeito próprio de organizar o conhecimento. Elas usam três definições para retratar o que é desejado [22]. 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, que possui uma semântica clara, além de separar o conhecimento assertivo e terminológico.

Tais estudos levaram ao desenvolvimento da $\mathcal{ALC}$. 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`:

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	$\forall\, \mathtt{r.A}$
Existencial	$\exists\, \mathtt{r.A}$

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 $\mathcal{ALC}$ [31], que são 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.

Com essa nomenclatura, pode-se dar duas linguagens como exemplo[28]. Uma delas 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.

Uma outra 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[3]:

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

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*: `HinoNacional` $\sqsubseteq$ `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:

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	$\forall\, \mathtt{r.A}$	$t_x(\forall \mathtt{r.A})$	$\forall y(r(x,y) \rightarrow t_y(A))$
Existencial	$\exists\, \mathtt{r.A}$	$t_x(\exists \mathtt{r.A})$	$\exists y(r(x,y) \wedge t_y(A))$
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. $\mathtt{Pop} \sqsubseteq \mathtt{Cancao}$ vira $\forall x(\mathtt{Pop}(x) \rightarrow \mathtt{Cancao}(x))$.
2. $\mathtt{Cancao} \sqcup \mathtt{HinoNacional} \sqsubseteq \mathtt{Musica}$ é traduzida como $\forall x(\mathtt{Cancao}(x) \vee \mathtt{HinoNacional}(x) \rightarrow \mathtt{Musica}(x))$.
3. $\forall\, \mathtt{canta.Musica} \sqsubseteq \mathtt{Cantor}$ corresponde a $\forall x(\forall y(\mathtt{canta}(x,y) \rightarrow \mathtt{Musica}(y)) \rightarrow \mathtt{Cantor}(x))$.
4. $\exists\, \mathtt{canta.Pop} \sqsubseteq \mathtt{Cantor} \sqcap \mathtt{Compositor}$ equivale a $\forall x(\exists y(\mathtt{canta}(x,y) \wedge \mathtt{Pop}(y)) \rightarrow \mathtt{Cantor}(x) \wedge \mathtt{Compositor}(x))$.

3.5 Consistência

Após uma Base de Conhecimento ser feita, é necessário que ela seja consistente, ou seja, ela não pode se contradizer. A inconsistência acontece quando a inferência de um conceito e sua negação, ao mesmo tempo, é possível.

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 $(\mathtt{A} \sqcap \mathtt{B})(a)$ está em $\mathcal{A}$, mas $\mathtt{A}(a)$ e $\mathtt{B}(a)$ não estão os dois em $\mathcal{A}$, então a *ABox* $\mathcal{A}$ será acrescida de $\mathtt{A}(a)$ e $\mathtt{B}(a)$.

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

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

Regra PARA TODO (ou $\forall$-*rule*) Quando se $(\forall \mathtt{r}.\mathtt{A})(a)$ e $\mathtt{r}(a, b)$ estão em $\mathcal{A}$, mas $\mathtt{A}(b)$ 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* (choque) for alcançado, isto é, quando temos um conceito e sua negação na *ABox*, a sentença é insatisfatível.

Koubarakis [21] deu um exemplo para aplicar essas regras:

1	$(\forall \mathtt{hasChild}.\mathtt{Male} \sqcap \exists \mathtt{hasChild}.\neg \mathtt{Male})(\mathtt{a})$	given
2	$(\forall \mathtt{hasChild}.\mathtt{Male})(\mathtt{a})$	1, $\sqcap$-rule
3	$(\exists \mathtt{hasChild}.\neg \mathtt{Male})(\mathtt{a})$	1, $\sqcap$-rule
4	$\mathtt{hasChild}(\mathtt{a}, \mathtt{b})$	3, $\exists$-rule
5	$(\neg \mathtt{Male})(\mathtt{b})$	3, $\exists$-rule
6	$\mathtt{Male}(\mathtt{b})$	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 $\mathtt{C} \sqsubseteq \mathtt{D}$ são válidas, deve ser verificada a satisfazibilidade de $\mathtt{C} \sqcap \neg \mathtt{D}$. 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 [13].

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 [28].

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 [2]. Nela, os estados de crenças são representados por conjuntos logicamente fechados de sentenças, ou seja, conjuntos K tais $K = \text{Cn}(K)$, onde $\text{Cn}(K)$ representa o conjunto de todas as consequências lógicas de K.

Quando $K = \text{Cn}(K)$, diz-se que há um equilíbrio dos estados epistêmicos. Com K sendo logicamente fechado, se $K \models \psi$, sendo ψ uma sentença qualquer, tem-se que $\psi \in K$. Um sistema de crenças em equilíbrio epistêmico é chamado de base de crenças. O uso de bases de crenças aumenta a eficiência do reparo [17].

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 [29]:

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

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

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

Supraclassicalidade Toda consequência da lógica (L, Cn) é também uma consequência da lógica proposicional.

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 $\alpha \in \text{Cn}(K) \setminus K$.
2. α é rejeitada. Aqui, temos que $\neg\alpha \in K$.
3. α é indeterminada. Deste modo, K não conhece a sentença α, assim, $\neg\alpha \notin K$ e $\alpha \notin K$.

Existem também algumas questões metodológicas que precisam ser resolvidas, ou pelo menos observadas antes de alguma Revisão de Crenças. Gärdenfors [15] definiu algumas delas.

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 estruturada.

4.2 Operações clássicas de Revisão de Crenças

Existem três operações clássicas de Revisão de Crenças [14]. 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. 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 era indeterminada. Depois da operação de Expansão, tal conhecimento passa a ser 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 caracterizada por $K - \alpha$. No entanto, as analogias com a 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 $K - \alpha = \text{Cn}(K - \alpha)$
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 $\alpha \notin \text{Cn}(\varnothing)$, então $\alpha \notin K - \alpha$
A não ser que α seja uma tautologia, ela não estará contida no conjunto resultante da operação.

Equivalência Se $\text{Cn}(\alpha) = \text{Cn}(\beta)$, 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 [22], 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 $K * \alpha = \text{Cn}(K * \alpha)$
Após a Revisão, o fecho lógico do conjunto é mantido.

Sucesso $\alpha \in \text{Cn}(K * \alpha)$
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 $K * \alpha = L \leftrightarrow \neg\alpha \in \text{Cn}(\varnothing)$

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

Equivalência Se $\text{Cn}(\alpha) = \text{Cn}(\beta)$, 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, $(K * \alpha) + \beta = K * (\alpha \wedge \beta)$

4.2.4 Relações entre as operações

A Revisão e a Contração podem ser construídas uma em função da outra, como escreveu Gärdenfors [13]. 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. Essa é a fórmula da Revisão interna. Foi proposta, em 1993 [19], a fórmula da Revisão externa, que é $K * \alpha = (K + \alpha) - \neg\alpha$. Ela significa que primeiro o conhecimento é adicionado à base e, posteriormente, é removido o que o contradiz. Observando as duas fórmulas, pode-se notar que não importa a ordem das operações, desde que seja feita a Contração pela negação e a Expansão pela afirmação. De qualquer jeito, o resultado será o mesmo do que o da Revisão.

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 Construtores para a Contração

Existem dois construtores especiais para a Contração. Esses construtores são espécies de funções, que tentam passar uma "receita" de como fazer essa operação. Usando a Identidade de Levi, é possível fazê-los para a Revisão, também.

4.3.1 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 [5].

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:

- $S \subseteq T$;
- $\alpha \notin \text{Cn}(S)$;
- se $S \subsetneq R \subseteq T$, então $\alpha \in \text{Cn}(R)$, 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. Fazendo uma conexão com os postulados da Contração, uma operação $-$ satisfaz os seis postulados básicos se e somente se $-$ é uma Contração *Partial Meet*.

Para Bases de Crenças

A operação definida acima pode ser generalizada para bases de crenças. Segundo Hansson [19], uma operação $-_\gamma$ para uma base B é uma Contração *Partial Meet* se e só se satisfaz os seguintes postulados:

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

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

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

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.

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

4.3.2 Contração *Kernel*

Para Teorias

Assim como a Contração *Partial Meet*, a *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. [5].

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:

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

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.

Para Bases de Crenças

Assim como generalização feita para a Contração *Partial Meet* de teorias para bases de crenças, existe uma para a Contração *Kernel*. Um operador $-_\sigma$ para uma base B qualquer é uma Contração *Kernel* se e somente se satisfaz os seguintes postulados:

Inclusão $B -_\sigma \alpha \subseteq B$
Semelhante ao da Contração *Partial Meet*.

Sucesso Se $\alpha \notin \text{Cn}(\varnothing)$, então $\alpha \notin \text{Cn}(B -_\sigma \alpha)$
Funciona como no construtor anterior.

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

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

É uma versão mais fraca do postulado **Relevância** apresentado acima.

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

4.4 Pseudocontração

Uma contração *partial meet* pode ser aplicada a uma base de crenças e não resultar no conjunto desejado. Se a aplicarmos no conjunto $K = \{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n\}$, o resultado será $K - \alpha_i = \varnothing$, por **inclusão**. Esse não é o resultado da mesma operação sob o conjunto $K' = \{\alpha_1, \alpha_2, ..., \alpha_n\}$, embora Cn($K$) = Cn($K'$).

O que pode ser feito para evitar esse tipo de resultado é substituir o postulado da inclusão por um mais fraco [16]:

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

Com isso, a operação executada será chamada de pseudocontração [18].

4.4.1 Pseudocontração SRW

Uma proposta de pseudocontração foi proposta em 2015 [30]. 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 [20].

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*"[36], no início dos anos 2000.

Os primeiros rascunhos foram publicados em julho de 2002, e em fevereiro de 2004 [37], a OWL (*Web Ontology Language*) tornou-se o padrão recomendado pela W3C para o processamento de ontologias.

Em outubro de 2007 [35], 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. Ela foi recomendada oficialmente pela W3C em dezembro de 2012 [33].

Abaixo estão algumas características das duas versões da OWL.

5.1.1 OWL Web Ontology Language

A primeira versão possui três sublinguagens [27], cada uma com 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 $\mathcal{SHIF}^{(\mathcal{D})}$ [10].
- 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 [34]. 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*

Figura 5.1: O logotipo do 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 alguns programadores da Universidade de Manchester [26]. Sua primeira versão foi lançada em 1999, e a atual é de 2017.

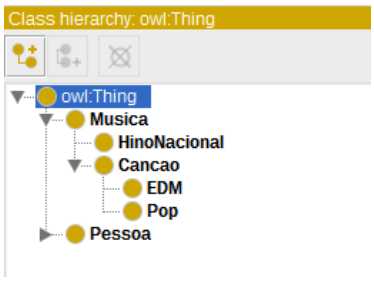
Seguindo os princípios do *Software* Livre, ele é um arcabouço gratuito e de código aberto, feito para a construção de sistemas inteligentes, com o uso ou não de uma interface gráfica.

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.

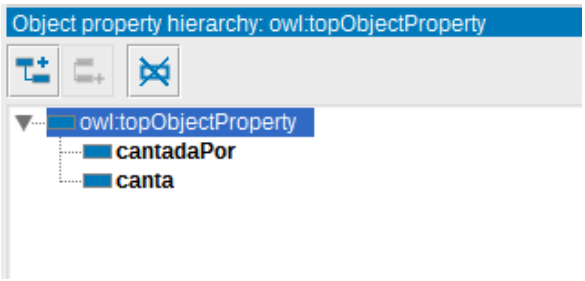
Para exemplificar alguns recursos do Protégé, será usada, como exemplo, parte da ontologia criada neste estudo:

- 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`;
  - `MarinaAndTheDiamonds` **`canta`** `Disconnect`.

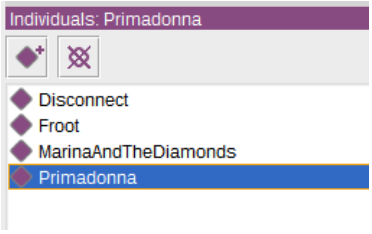
No *Protégé*, a ontologia fica assim:



(a) A hierarquia das classes.



(b) As propriedades da ontologia.



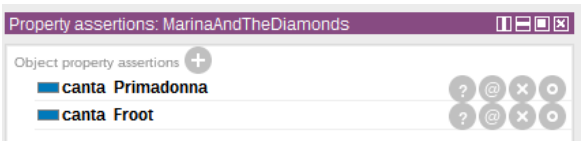
(c) As instâncias definidas.

Figura 5.2: A representação da ontologia do *Protégé*

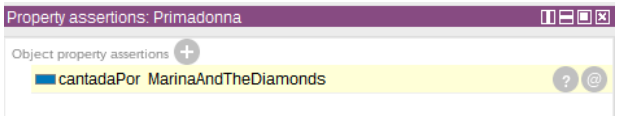
5.2.1 Raciocinadores

Um aspecto interessante sobre esse arcabouço é o uso de *plug-ins* de *raciocínio*, como o HermiT [1]. 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:



(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.

SPARQL é um acrônimo recursivo para SPARQL *Protocol and RDF Query Language* [4]. 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:

```
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, a contração *Partial Meet* e a *Kernel*. Ambas foram baseadas em implementações de algoritmos feitas na tese de doutorado de Cóbe [5].

Revisão Para essa operação, os códigos feitos por Resina para o seu trabalho de mestrado [28] foram reconstruídos.

Pseudocontração SRW Essa operação foi completamente absorvida ao *plug-in* do código feito por Matos, em seu trabalho de conclusão de curso. [22]

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].

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].

[1] https://www.jetbrains.com/idea/
[2] https://maven.apache.org/
[3] http://owlcs.github.io/owlapi/
[4] http://www.hermit-reasoner.com/
[5] http://jcommander.org

6.2 Algoritmo *BlackBox*

A implementação das três operações tem a sua parte principal em um algoritmo *BlackBox*, baseados nos estudos de Resina [28], onde tiveram suas corretudes provadas. Ele é chamado assim pois não precisa de um motor de inferência. É necessário apenas decidir se uma base de conhecimento implica certo axioma. Ele possui algumas variações, para o cálculo do conjunto-resíduo, do conjunto-*kernel* para a contração e do conjunto-*kernel* da revisão.

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.

As funções de seleção e de incisão utilizadas têm sua implementação bem simplificada, devolvendo qualquer elemento (apenas um) do conjunto-resíduo ou do conjunto-*kernel*.

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 [22]. 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
    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 implementação do código de Cóbe [22]. 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 [28], 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

6.3 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.

6.3.1 Pseudocontração SRW

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

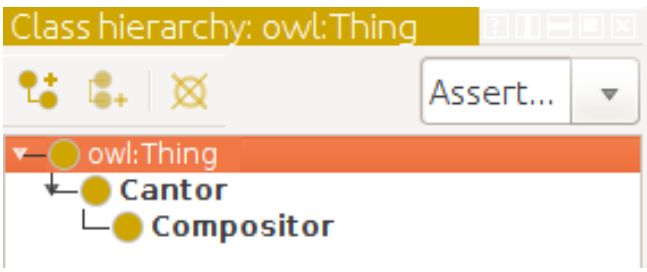


Figura 6.1: A estrutura da ontologia.

Ela possui um indivíduo, `markRonson`, pertencente à classe `Compositor`.

Figura 6.2: 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`.

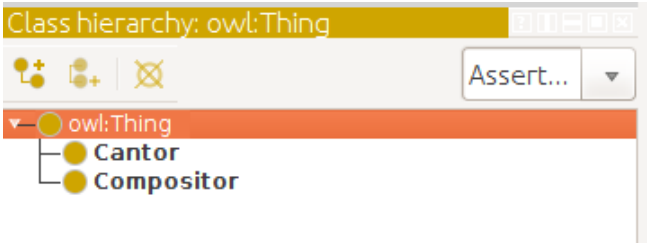


Figura 6.3: O resultado da operação.

6.3.2 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`

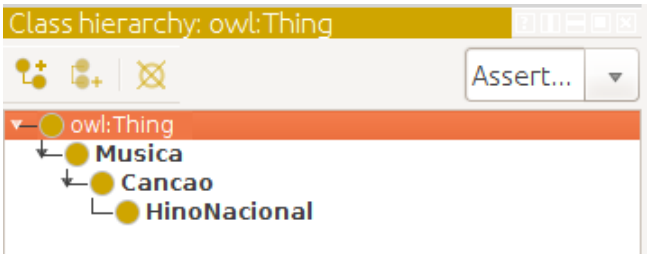


Figura 6.4: A estrutura da ontologia.

Como definido anteriormente, na verdade, `Cancao` e `HinoNacional` são classes na mesma hierarquia. 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.

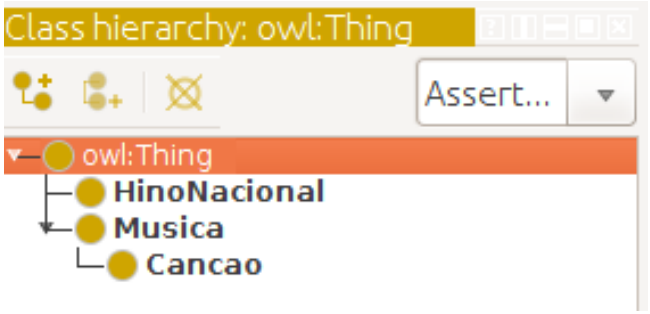


Figura 6.5: O resultado da operação.

6.3.3 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:

- `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`.

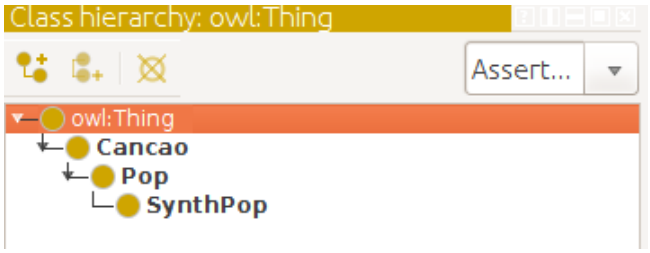


Figura 6.6: As classes da ontologia.

Com o comando abaixo, a operação é realizada.

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

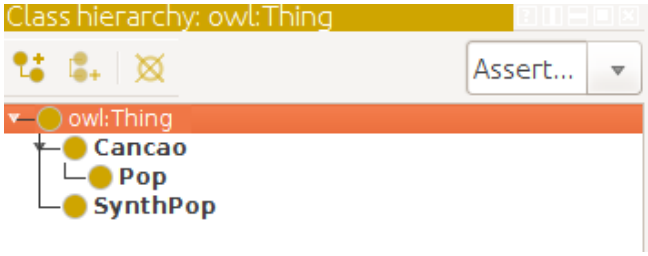


Figura 6.7: O resultado da operação.

6.3.4 Revisão

Capítulo 7

Análise de desempenhos

Agradecimentos

Referências Bibliográficas

[1] *Hermit owl reasoner*. Online; accessed 13-julho-2018.

[2] C. E. ALCHOURRÓN, P. GÄRDENFORS, AND D. MAKINSON, *On the logic of theory change: Partial meet contraction and revision functions*, The journal of symbolic logic, 50 (1985), pp. 510–530.

[3] F. BAADER, I. HORROCKS, AND U. SATTLER, *Description Logics*, in Handbook of Knowledge Representation, F. van Harmelen, V. Lifschitz, and B. Porter, eds., Elsevier, 2008, ch. 3, pp. 135–180.

[4] D. BECKETT, *What does sparql stand for?* Online; accessed 12-julho-2018.

[5] R. M. D. O. COBE, *Integração entre múltiplas ontologias: reúso e gerência de conflitos*, PhD thesis, Universidade de São Paulo, 2014.

[6] I. DAHLBERG, *Teoria do conceito*, Ciência da informação, 7 (1978).

[7] C. DEBRUYNE, *The relation between a framework for collaborative ontology engineering and nicola guarino's terminology and ideas in"formal ontology and information systems"*., in VaSCo@ WebSci, 2013, pp. 34–44.

[8] P. C. FRANÇA, *Conceitos, classes e/ou universais: com o que é que se constrói uma ontologia?*, Linguamática, 1 (2009), pp. 105–121.

[9] P. GIARETTA AND N. GUARINO, *Ontologies and knowledge bases towards a terminological clarification*, Towards very large knowledge bases: knowledge building & knowledge sharing, 25 (1995), pp. 307–317.

[10] B. C. GRAU, I. HORROCKS, B. MOTIK, B. PARSIA, P. PATEL-SCHNEIDER, AND U. SATTLER, *Owl 2: The next step for owl*, Web Semantics: Science, Services and Agents on the World Wide Web, 6 (2008), pp. 309–322.

[11] T. R. GRUBER, *Toward principles for the design of ontologies used for knowledge sharing*, International journal of human-computer studies, 43 (1995), pp. 907–928.

[12] N. GUARINO, *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, vol. 46, IOS press, 1998.

[13] P. GÄRDENFORS, *Knowledge in flux: Modeling the dynamics of epistemic states.*, The MIT press, 1988.

[14] ——, *Belief revision: A vade-mecum*, in International Workshop on Meta-Programming in Logic, Springer, 1992, pp. 1–10.

[15] ——, *Belief revision*, vol. 29, Cambridge University Press, 2003.

[16] S. HANSSON, *New operators for theory change*, Theoria, 55 (1989), pp. 114–132.

[17] S. O. HANSSON, *Belief contraction without recovery*, Studia logica, 50 (1991), pp. 251–260.

[18] ——, *Changes of disjunctively closed bases*, Journal of Logic, Language and Information, 2 (1993), pp. 255–284.

[19] ——, *Reversing the levi identity*, Journal of Philosophical Logic, 22 (1993), pp. 637–669.

[20] I. HERMAN, *W3c semantic web frequently asked questions*. Online; accessed 12-julho-2018.

[21] M. KOUBARAKIS, *Tableau proof techniques for dls*. Online; accessed 30-Junho-2018.

[22] V. B. MATOS AND R. WASSERMANN, *Implementação de um módulo de pseudocontração em revisão de crenças para o editor de ontologias protégé*, (2016).

[23] MERRIAM-WEBSTER ONLINE, *Merriam-Webster Online Dictionary*.

[24] M. NICKLES, A. PEASE, A. C. SCHALLEY, AND D. ZAEFFERER, *Ontologies across disciplines*, Ontolinguistics-How Ontological Status Shapes the Linguistic Coding of Concepts, (2007), pp. 23–67.

[25] N. F. NOY, D. L. MCGUINNESS, ET AL., *Ontology development 101: A guide to creating your first ontology*, 2001.

[26] U. OF STANFORD, *Protégé*. Online; accessed 12-julho-2018.

[27] M. RAY, *Owl sub-languages: Lite, dl and full*. Online; accessed 12-julho-2018.

[28] F. M. X. RESINA, *Revisão de crenças em lógica de descrição: Um plug-in para o protégé*, (2010).

[29] M. M. RIBEIRO, *Revisao de crenças em lógicas de descrição e em outras lógicas não clássicas*, PhD thesis, Universidade de São Paulo, 2010.

[30] Y. D. SANTOS, M. M. RIBEIRO, AND R. WASSERMANN, *Between belief bases and belief sets: partial meet contraction*, in Proceedings of the 2015 International Conference on Defeasible and Ampliative Reasoning-Volume 1423, CEUR-WS. org, 2015, pp. 50–56.

[31] M. SCHMIDT-SCHAUSS AND G. SMOLKA, *Attributive concept descriptions with complements*, Artificial intelligence, 48 (1991), pp. 1–26.

[32] B. SMITH, W. KUSNIERCZYK, D. SCHOBER, AND W. CEUSTERS, *Towards a reference terminology for ontology research and development in the biomedical domain*, in KR-MED, vol. 2006, 2006, pp. 57–66.

[33] W3C, *Owl 2 web ontology language*. Online; accessed 12-julho-2018.

[34] ——, *Owl 2 web ontology language profiles (second edition)*. Online; accessed 12-julho-2018.

[35] ——, *Owl working group*. Online; accessed 12-julho-2018.

[36] ——, *Web-ontology (webont) working group (closed)*. Online; accessed 12-julho-2018.

[37] ——, *World wide web consortium issues rdf and owl recommendations*. Online; accessed 12-julho-2018.