Ontoprolog: uma linguagem para especificação de discursos sobre ontologias

Lauro César Araujo

Doutor em Ciência da Informação, especialidade Arquitetura da Informação Pesquisador do Centro de Pesquisa em Arquitetura da Informação da Universidade de Brasília, CPAI/UnB

Mamede Lima-Marques

Doutor em Ciência da Computação pela Université Toulouse III Paul Sabatier, França, pós-doutorado no Centro de Lógica, Epistemologia e História da Ciência - UNICAMP. Professor Titular da Universidade de Brasília. Fundador e diretor do Centro de Pesquisa em Arquitetura da Informação, CPAII/UnB

http://dx.doi.org/10.1590/1981-5344/2532

Este artigo introduz a arquitetura da informação de uma linguagem formal textual para representar e obter deduções a respeito problemas reais com base em ontologias de domínio e ontologias de fundamentação. O arcabouço é desenvolvido com base em Programação em Lógica, e consiste em uma linguagem formal que utiliza o paradigma de metamodelagem para produzir ontologias heterogêneas que podem ser descritas como instâncias de ontologias de fundamentação.

Palavras-chave: *Unified Foundation Ontology; Lógica não Clássica; Programação em Lógica.*

Ontoprolog: a language to specify discourses on ontologies

This paper introduces the information architecture of a textual formal language to represent and to reason about real problems based on domain ontologies and foundational ontologies. The framework is based on Logic Programming, and consists of a formal language that uses the metamodeling paradigm to produce heterogeneous

ontologies that can be described as instances of foundation ontologies.

Keywords: *Unified Foundation Ontology; Non-Classical Logic; Logic Programming.*

Recebido em 05.09.2015 Aceito em 11.04.2017

1 Introdução

A Science of Information (SI), conforme proposta por Hofkirchner (2011, p. 372), é uma disciplina que lida com processos de informação de sistemas naturais, sociais e tecnológicos de modo mais amplo do que a tradicional Information Science (IS). Enquanto esta é uma área proveniente da Biblioteconomia e da Documentação, aquela é uma disciplina complementar e emergente que busca um entendimento mais amplo do conceito de informação. Isso inclui integração epistemológica com áreas como Lógica, Matemática, Ciência da Computação, Física, Cibernética, entre outras. Nesse contexto, Lima-Marques (2011) propõe a Architecture of Information (AI) como uma área da Science of Information que trata a informação como configurações do mundo que podem ser experienciadas por sujeitos sociais. Como efeito dessa experiência, os sujeitos formam teorias de conceituações a respeito da realidade. Quando comunicadas, as teorias são proferidas na forma de discursos. Os discursos carregam proposições que descrevem conceitos do domínio experienciado pelo sujeito. Para que os discursos possam ser utilizados no contexto social considerado pelos modelos de AI, o arquiteto da informação deve valer-se, por exemplo, de linguagens formais, sistemas lógicos, ontologias e técnicas de modelagem conceitual, que são instrumentos para descrição e raciocínio desses discursos sobre a realidade da informação.

Na esfera da Modelagem Conceitual, uma ontologia é entendida como uma teoria metafísica. Ela pode descrever um domínio, ou representar uma teoria geral abstrata, utilizada como instrumento para a construção de outras ontologias. Como um sujeito que experimenta a realidade, um arquiteto da informação deve lançar mão de diferentes instrumentos que servem como suporte à atividade de formulação de ontologias. Nesse contexto destacam-se como ferramentas as ontologias de fundamentação, como a Unified Foundational Ontology (UFO) proposta por Guizzardi (2005), que funcionam como visão e metateoria a respeito do mundo[1], bem como linguagens visuais de modelagem, como a OntoUML, proposta na mesma obra. Aliados a essas ferramentas gerais, nos últimos anos têm sido desenvolvidos instrumentos concretos baseados em software, como as soluções de model finding construídas com Alloy (BRAGA et al., 2010), que possuem a capacidade de simular instâncias de

[1] Ontologias de fundamentação cognitivamente coerentes, como a UFO, descrevem entidades categóricas abstratas da realidade, mas que são reconhecíveis e compartilhadas por diferentes sujeitos. Por isso, elas servem como ontologia de referência, quase como paradigmas a respeito da realidade.

modelos de ontologias de universais; as técnicas de verificação de padrões e de anti-padrões de modelagem apresentadas por Guizzardi (2014) e implementadas na ferramenta OntoUML Lightweight Editor (OLED) (SALES, 2014, p. 237); e representações em OWL de regras sobre ontologias baseadas na UFO, como as propostas por Guizzardi e Zamborlini (2014), entre outros. Porém, embora existam instrumentos específicos quem auxiliam em determinados aspectos da modelagem, não se encontra na literatura especializada linguagem formal textual que permita representação e raciocínio sobre ontologias de domínios criadas de forma colaborativa, a partir de ontologias de fundamentação, com suporte à integração de especificações e de identificação de inconsistências.

Diante desse contexto, o objetivo deste artigo é introduzir a proposta de uma linguagem formal textual para representar problemas reais com base em ontologias, e obter deduções a respeito de teorias de domínio baseadas em ontologias de fundamentação, a exemplo da UFO. A realização desse objetivo é alcançada com a construção de um arcabouço com base em Programação em Lógica intitulado Ontoprolog. Ontoprolog consiste em uma linguagem baseada em operadores, em um tradutor dessa linguagem para cláusulas de Horn e em um conjunto de predicados a respeito das teorias denotadas por essas cláusulas, que são base para a construção de noções de consistência que são suportadas por lógicas clássicas e modais. O escopo do artigo é apresentar os principais aspectos de arquitetura da informação da abordagem adotada. Aspectos das Lógicas Modais usadas para integração de ontologias descritas de forma colaborativa, bem como a ontologia de fundamentação UFO, serão contemplados em outras publicações, e são apenas indicados neste texto.

Conforme Almeida (2014), embora o uso de ontologias seja comum na (tradicional) Ciência da Informação, Inteligência Artificial e Ciência da Computação, o tratamento da informação por meio de Lógicas Matemáticas não é tão comum na IS. O diferencial deste artigo é a aplicação do conceito de Engenharia Lógica desenvolvido por Lima-Marques (1992, p. 7-10) para tratar problemas de informação que podem ser formalizados com base na aplicação de Lógicas, técnicas de computação e ontologias. Essa abordagem é defensável na medida que traz adequação metodológica e consistência formal aos modelos desenvolvidos no âmbito da Ciência da Informação.

2 Bases para a definição da linguagem

Parte do problema que consiste em descrever uma ontologia de determinado domínio pode ser reduzido ao problema semiótico de registrar, em alguma linguagem, a expressão da experiência do sujeito com os objetos. Posto dessa forma, a construção de modelos conceituais é também uma atividade dependente de sujeitos, em que a representação de conceitos a respeito do mundo não se trata da representação da verdade última, mas do registro de acordos entre sujeitos a respeito de entendimentos compartilhado sobre a realidade. Porém, esses acordos não são arbitrários: são fundamentados com a

realidade experimentada, por isso, possuem compromissos ontológicos. Nesse cenário, ontologias de fundamentação funcionam como referências sobre o universo de discurso e visam tornar menos imprecisos esses acordos.

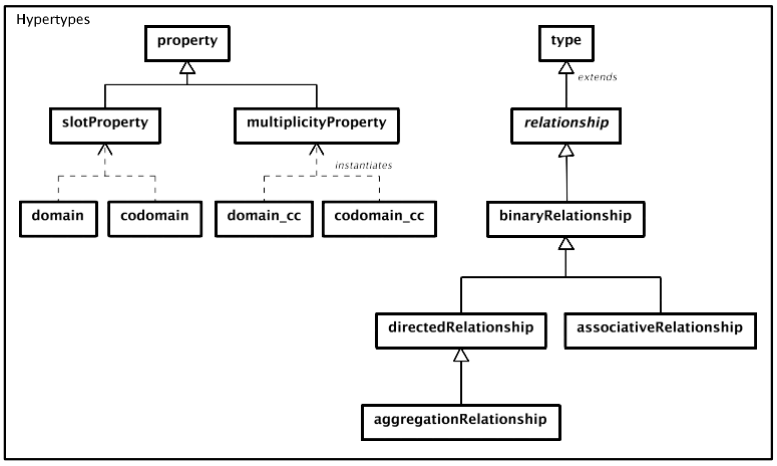
A introdução de ontologias de fundamentação, como a UFO, conduz a um modelo metateórico em que a UFO exerce o papel de metateoria sobre a qual teorias específicas sobre determinados domínios ontológicos são descritas por um sujeito. Porém, mesmo como uma teoria geral, a UFO é construída sobre uma ontologia de referência ainda mais ampla, no caso, a Ontologia de Quatro Categorias Aristotélicas (LOWE; 2006, p. 18), (GUIZZARDI; WAGNER, 2010, p. 178). Essa escolha confirma a sujeição da UFO a um paradigma de metamodelagem, em que indivíduos particulares de uma teoria são instâncias de categorias universais que capturam propriedades comuns dos indivíduos que exemplificam seus tropes. Devido a isso, é necessário destacar a existência de um paradigma ontológico fundamental, que governa a UFO e esta, por sua vez, governa as teorias específicas de domínio descritas com base nela. Esse paradigma antecede as características modais da interpretação das instâncias de uma ontologia descrita com base na UFO.

Para formalizar essa ontologia de referência fundamental, a Figura 1 ilustra em um diagrama UML a Metateoria de Hypertypes, que consiste em um arranjo lógico proposto como teoria de referência inicial para descrição de conceituações com Ontoprolog. A metateoria representa o paradigma metafísico mais elementar da linguagem. Baseada no Quadrado Aristotélico, a Metasteoria de Hypertypes reconhece a existência categórica fundamental de apenas duas entidades, os properties e os types. Instâncias de properties são tropes dependentes de instâncias de types. Um type é uma referência a uma entidade do universo do discurso. Há pelo menos quatro relações formais básicas entre properties e types. A relação de subsunção, representada na Figura pela seta aberta, que aponta para o conceito mais geral, é equivalente à noção transitiva de subconjunto, em que membros do conjunto mais específico são também membros do conjunto mais geral. A relação de instância é representada pela seta pontilhada, e aponta para o conceito que é instanciado. A relação é similar à relação de subsunção, com a diferença de que as relações de instância não são transitivas entre si, de modo que subsunções da

instância não são membros dos conjuntos instanciados. Um elemento é instância de outro porque ele é membro da categoria representada pelo universal instanciado. No âmbito de uma linguagens de metamodelagem, como a proposta neste artigo, a relação de instância exerce papel extremamente importante, porque denota a definição de níveis teóricos, que hierarquizam os diferentes (meta)modelos descritos. Desse modo, uma teoria é construída por um conjunto de instâncias de types e de properties e suas respectivas relações formais. As instâncias desses conceitos denotam um nível teórico subjugado e definido sobre essa teoria. Além das relações formais de subsunção e de instanciação, que governam tanto types como properties, há ainda dois

tipos de relações formais que relacionam diretamente types e properties. A relação de associação de propriedade relaciona um property a um type, no sentido que denote que instâncias de determinado conceito possuem instâncias de determinada propriedade. Por fim, a relação de atribuição de valor indica a relação específica entre instâncias de properties, instâncias de types e o valor atribuído a essa relação. Segundo a Figura 1, esse valor pode ser de dois tipos: slotProperty ou multiplicityProperty. Cada uma possui duas instâncias específicas. As instâncias da primeira indicam que as propriedades podem ser de domínio (domain) ou de contra-domínio (codomain) cujos valores representam um type de algum nível da teoria. Isso indica que as propriedades são relações entre types. Já as instâncias de multiplicityProperty indicam a existência de um tipo específico de type que se refere a restrições de multiplicidade sobre os domínios e contra-domínios. Seguindo a Figura 1, do lado dos types a relação de subsunção denota a visão básica de mundo em que alguns types são relações (relationship) e que algumas relações são binárias (binaryRelationship). Dessas relações binárias se entende ainda dois tipos mais específicos, no caso, as relações direcionadas (directedRelationship), que são relações funcionais, e as associativas (associativeRelationship). Por fim, as relações direcionais também podem indicar um tipo mais específico de conceito, no caso, as relações de agregação (aggregationRelationship), em que se pode descrever conceitos como uma composição de outros.

Figura 1 – Metateoria de *Hypertypes*

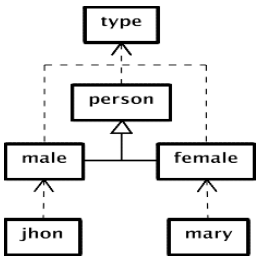


Fonte: os autores

Essa hierarquia básica de conceitos descreve o vocabulário e as distinções lógicas mais fundamentais sobre as quais ontologias mais específicas são construídas. Além das quatro relações formais já apresentadas (subsunção, instanciação, associação e propriedade e atribuição de valor), uma quinta relação formal relaciona qualquer elemento da Metateoria de Hypertypes, ou de suas instâncias, a entidades ditas extra-ontológicas. Tratam-se de meta-propriedades que indicam restrições ou características lógicas às ontologias. Por exemplo, na Figura 1 a tipografia em itálico do conceito relationship indica que este

denota uma entidade abstrata. Dessa forma, há uma relação entre o conceito relationship com uma meta-propriedade, digamos, abstract, que indica, por exemplo, que aquele conceito não pode ser instanciado diretamente. O domínio das metapropriedades de uma teoria Ontoprolog é aberto, porque elas devem ser definidas de acordo com objetivos específicos da modelagem. As meta-propriedades são entendidas como elementos fora da teoria de domínio, e são utilizadas como elementos lógicos da linguagem. Restrições e regras específicas de inferências podem ser definidas com base nessas relações de meta-propriedade.

Figura 2 – Exemplo de ontologia de domínio



Fonte: os autores

Embora não reconheça muitas categorias ontológicas, discursos a respeito da realidade podem ser descritos com base nessa ontologia fundamental. Por exemplo, imaginemos a situação em que se perceba na realidade a existência de duas pessoas, Jhon (`jhon`) e Mary (`mary`). Do ponto de vista metodológico, uma primeira tentativa de modelagem conceitual é identificar um conceito universal que lhes dê identidade enquanto indivíduos distintos de outros da realidade, mas que possa ser geral o suficiente para ambos serem instância desse universal. Portanto, digamos, ambos são pessoas (`person`). Porém, do ponto de vista de universais, ambos são de sexo diferentes, digamos, masculino (`male`) e feminino (`female`). Dessa forma, pode-se dizer que Jhon instancia male e Mary instancia `female`. Já `male`, `female` e `person`, todos eles, instanciam a entidade `type` da Metateoria de Hypertypes. A Figura 2 ilustra por meio de diagramas UML essa descrição.

Ainda que as relações formais de subsunção, instanciação, metateorias etc., representem apenas parte das dimensões tratadas pela linguagem Ontoprolog, elas são basilares para características importantes dos resultados obtidos. As seções seguintes descrevem aspectos técnicos da definição formal da linguagem proposta.

2.1 A escolha da Programação em Lógica

A linguagem Ontoprolog é concebida com base nos fundamentos ontológicos da UFO e construída sobre um arcabouço lógico da Programação em Lógica. A Programação em Lógica é escolhida devido às inúmeras possibilidades teóricas de tratamento lógico da informação, como a disponibilidade de diferentes abordagens de programação por restrições (FRÜHWIRTH, 1998; SNEYERS et al., 2010) e de lógicas não clássicas, como lógicas não monotônicas[2], lógicas multi-modais presentes no MProlog (NGUYEN, 2009), Molog (FARIÑAS DEL CERRO, 1986), TIM (BALBIANI; HERZIG; LIMA-MARQUES, 1991) e TARSKI (ALLIOT; HERZIG; LIMA-MARQUES, 1992), lógicas paraconsistentes (RODRIGUES, 2010), entre outras. Além do amplo arcabouço teórico disponível, aspectos de metaprogramação, que permitem introspecção da base de dados de runtime e acesso às estruturas de dados, tabelas e ligação de variáveis tanto em nível de programação como de meta-programação de forma nativa, tornam as implementações de Programação em Lógica, conhecidas como Prolog, adequadas aos propósitos deste trabalho. Com base em Christiansen (2002), dentre as características de Prolog que nos faz optar pela ferramenta no âmbito deste trabalho, destacam-se:

a) Gupta (1998) defende que as cláusulas de Horn usadas na Programação em Lógica proveem arcabouço adequado para especificar a semântica denotacional de linguagens e inferir propriedades de programas, especialmente se comparado a Teoria de Conjuntos e Cálculo Lambda;

b) a linguagem base de Prolog é padronizada pela série de normas ISO/IEC 13211-1:1995, o que garante a interoperabilidade de programas entre diferentes implementações;

há diferentes implementações que enriquecem aspectos específicos da linguagem sem, no entanto, abrir mão do suporte à especificação padrão ISO. Dentre essas implementações, destacam-se os softwares livres SWI-Prolog[3] e XSB[4] [5], além do sistema com licença comercial e acadêmica SICStus Prolog[6]. Todos provêm, inclusive, integrações com linguagens usadas corporativamente, como Java;

por fim, de forma muito enfática, Kowalski (1974), um dos célebres criadores de Prolog, defende que a linguagem é adequada não apenas para representar outros tipos de lógicas, mas é ela em si uma lógica que permite que "conhecimentos" sejam expressos diretamente em forma clausal:

[2] A incorporação de controles como *cut* – simbolizado por ! – representam aspectos operacionais incorporados à semântica dos programas. Isso justifica a descrição comum de Prolog como "Lógica + Controle": ele engloba tanto aspectos lógicos denotacionais como comportamentos operacionais que extrapolam as características clássicas. Além disso, a "hipótese do mundo fechado" (REITER, 1978) implica em comportamentos não-monotônicos não presentes originalmente em lógicas com comportamento clássico.

[3] Disponível em: <http://www.swi-prolog.org/>. Acesso em: 2 jul 2015

[4] Disponível em: <http://xsb.sourceforge.net/>. Acesso em: 2 jul 2015

[5] Uma característica marcante do XSB é capacidade de lidar naturalmente com *tabling* e *memoization* de resultados (SWIFT; WARREN, 2012).

[6] Disponível em: <https://sicstus.sics.se/>. Acesso em: 2 jul 2015

> Methods for transforming arbitrary first-order sentences into clausal form are described in Nilsson's book [. . .]. It is our thesis, however, that clausal form defines a natural and useful language in its own right, taht thoughts can conveniently be expressed directly in clausal form, and that literal translation from another language, such as full predicate logic, often distorts the original thought (KOWALSKI, 1974, p. 569).

É devido a essa visão de Kowalski (1974) que no decorrer deste artigo usa-se preferencialmente a notação clausal, conforme apresentado por Lloyd (1993), em detrimento de outros tipos de notação, como as tradicionais notações de Primeira Ordem.

3 Sintaxe da linguagem

A sintaxe da linguagem de Ontoprolog é concebida de modo a equilibrar dois requisitos de natureza diferentes: um pragmático e outro técnico. O requisito pragmático estabelece que a linguagem deve ser precisa, concisa e passível de ser lida em voz alta de modo mais natural possível. O requisito técnico impõe que ela deve ser definida com base na composição de operadores de Prolog. Esse segundo requisito é devido também a um aspecto pragmático: as sentenças de Ontoprolog são concebidas para sintaticamente serem reconhecidas como sentenças de um programa padrão em Lógica, de modo que possam ser incorporadas em qualquer programa Prolog. Já o primeiro requisito pragmático é devido ao fato de que a linguagem de Ontoprolog deve ser utilizada em sessões reais de modelagem conceitual, em que geralmente um grupo de pessoas participam na construção de acordos a respeito do universo do discurso de modo colaborativo. Por isso, ela deve otimizar a leitura natural.

A escolha de definição com base na combinação de operadores de Prolog permite que extensões da linguagem sejam realizadas de modo relativamente simples. Por isso, concebe-se inicialmente uma sintaxe base, capaz de expressar constructos primários da linguagem. Essa sintaxe base é apresentada no Código 1 por meio de regras EBNF (Extended Backus-Baur Form). A notação concreta é baseada no padrão proposto em W3C (2008):

Código 1 – Definição EBNF da sintaxe base de Ontoprolog

```
ONTOLOG_SPECIFICATION ::= SENTENCE+

SENTENCE ::= ( DISJOINT | INSTANCE | PARTIAL_SUBSUMPTION | COMPLETE_SUBSUMPTION |
PROPERTY_ASSOCIATION | PROPERTY_ASSIGNMENT | SUBSETS | REDEFINES | POWER_TYPE | META ) '.'

DISJOINT ::= 'disjoint' '[' ATOM (',' ATOM)* ']'

INSTANCE ::= ( ATOM | 'disjoint' '[' ATOM (',' ATOM )* ']' ) '::' ( INSTANTIABLE_ENTITY | '['
ATOM (',' ATOM )* ']' )

PARTIAL_SUBSUMPTION ::= ( ATOM | 'disjoint'? '[' ATOM ( ',' ATOM )* ']' ) ( '::'
INSTANTIABLE_ENTITY )? ( 'extends' | 'extend' ) ( ATOM | '[' ATOM ( ',' ATOM )* ']' )

COMPLETE_SUBSUMPTION ::= ( ATOM | ('disjoint'? '[' ATOM ( ',' ATOM)* ']' ) ) ( '::'
INSTANTIABLE_ENTITY )? 'cover' ATOM

PROPERTY_ASSOCIATION ::= 'property' ( ATOM | '[' ATOM (',' ATOM )* ']' ) 'on' (
ENTITY_WITH_PROPERTY | '[' ENTITY_WITH_PROPERTY (',' ENTITY_WITH_PROPERTY )* ']' )

PROPERTY_ASSIGNMENT ::= PROPERTY_TYPE 'at' ENTITY_IN_RELATION ':=' VALUE

VALUE ::= ANY_PROLOG_TERM

SUBSETS ::= PROPERTY_TYPE 'at' ENTITY_IN_RELATION 'subsets' PROPERTY_TYPE 'at'
ENTITY_IN_RELATION

REDEFINES ::= PROPERTY_TYPE 'at' ENTITY_IN_RELATION 'redefines' PROPERTY_TYPE 'at'
ENTITY_IN_RELATION

POWER_TYPE ::= 'powertype' ( ENTITY_PT | '[' ENTITY_PT (',' ENTITY_PT )* ']' ) 'classifying' (
ATOM | '[' ATOM ( ',' ATOM )* ']')

META ::= 'meta' ( META_PROPERTY | '[' META_PROPERTY (',' META_PROPERTY)* ']' ) 'on' ( ( ATOM
'at' )? ( ( 'extensions' | 'transitive'? 'direct'? 'instances' ) 'of' )? )? ( ENTITY_WITH_META
| '[' ENTITY_WITH_META (',' ENTITY_WITH_META )* ']' )

META_PROPERTY ::= ANY_PROLOG_TERM

INSTANTIABLE_ENTITY ::= ATOM

PROPERTY_TYPE ::= ATOM

ENTITY_IN_RELATION ::= ATOM

ENTITY_WITH_PROPERTY ::= ATOM

ENTITY_WITH_META ::= ENTITY_WITH_PROPERTY

ENTITY_PT ::= ENTITY_WITH_PROPERTY | ATOM 'at' ATOM

ATOM ::= [a-z][a-zA-Z0-9]* | "'" [^']* "'"
```

Uma instância válida da estrutura gramatical apresentada no Código 1 é chamada de Especificação ou Especificação Ontoprolog. Conforme a definição apresentada, uma Especificação (`ONTOPROLOG_SPECIFICATION`) baseada na sintaxe de Ontoprolog é um conjunto não vazio de sentenças (`SENTENCE`). As sentenças podem ser de diferentes categorias sintáticas, mas sempre terminam com um ponto final. A primeira delas, `DISJOINT`, é definida com o operador disjoint seguido de uma lista de átomos separados por vírgulas entre colchetes. O mesmo tipo de raciocínio é válido para as demais categorias sintáticas, que são definidas conforme as regras EBNF. A definição dos símbolos terminais que denotam entidades externas à linguagem é realizada pela expressão regular contida em `ATOM` e em `ANY_PROLOG_TERM`, sendo que este último refere-se a termos de Prolog. As

categorias sintáticas principais são descritas na Tabela 1. Já a semântica formal de parte dessas categorias é apresentada na seção 4 e seção 5.

Tabela 1 – Descrição das categorias sintáticas das sentenças de Ontoprolog

Tipo de sentença	*Descrição*
DISJOINT	*Representa uma lista de conceitos que qualquer dois não podem figurar conjuntamente como supertipos de subsunções ou metatipos de instâncias*
INSTANCE	*Denota a relação de instância estabelecida entre entidades*
PARTIAL SUBSUMPTION	*Estabelece a relação de subsunção entre entidades*
COMPLETE_SUBSUMPTION	Estabelece a relação de subsunção entre entidades com informação adicional de que as subsunção apresentada são todas as possíveis para as classes mais gerais
PROPERTY_ASSOCIATION	*Define a relação de associação de propriedade*
PROPERTY ASSIGNMENT	*Define a relação de atribuição de valor de propriedade*
SUBSETS	*Denota a relação de subconjunto de relação ontológica entre entidades*
REDEFINES	*Denota a relação de redefinição de relação ontológica entre entidades*
POWER TYPE	*Define a relação de power type entre entidades*
META	*Atribui meta-propriedades a entidades*

Fonte: Os autores.

O Código 2 contém a definição em Prolog de operadores que implementam a sintaxe definida no Código 1. A prioridade e associatividade dos operadores são apresentadas como de praxe, por meio do predicado `op/3`. O uso de operadores de Prolog na implementação da sintaxe permite representação e acesso direto às árvores sintáticas, de modo que a análise léxica de uma linguagem construída internamente em um Programa em Lógica é realiza de forma automática, e acessível em termos de metaprogramação. Desse modo, é possível construir sentenças da linguagem que são igualmente sentenças de um programa Prolog comum, o que inclui a capacidade de se realizar unificação das sentenças com base na árvore sintática da composição de operadores.

Código 2 – Operadores Prolog para implementação das regras EBNF

```
:- op(646, xfy, [on, classifying, :=]).
:- op(645,  fy, [meta, powertype, property]).
:- op(644, yfx, [redefines, subsets]).
:- op(643, xfy, [at, value]).
:- op(630, yfx, [extends, extend, cover]).
:- op(480, xfy, [::]).
:- op(477, yfx, [of]).
:- op(475,  yf, [instance, instances]).
:- op(450,  fy, [disjoint]).
:- op(390, xfx, [..]).
:- op(379,  fy, [direct, transitive]).
```

A atribuição de significado aos constructos sintáticos criados como instâncias das definições gramaticais apresentadas no Código 1, e descritas na Tabela 1, é realizadas por meio de traduções entre essa

sintaxe e um conjunto de cláusulas de Horn, estas apresentados na seção 4. O mecanismo de tradução da sintaxe na semântica é apresentado na seção 5.

4 Estrutura semântica

A semântica de uma Especificação Ontoprolog é denominada Teoria, e é dada por meio da estrutura OT = (C, G, H, R), tal que:

a) C é um conjunto finito e não vazio de entidades que guardam relação ontológica com o mundo;

b) G é um conjunto finito e possivelmente vazio de entidades lógicas que não exigem compromisso ontológico com o universo descrito, mas fazem parte do universo de objetos meta-teóricos a respeito da realidade;

c) H é um conjunto finito e não vazio de instâncias de relações entre objetos C ∪ G, representadas por um conjunto de símbolos predicativos primitivos. As relações são apresentadas na forma das cláusulas de Horn descritas na Tabela 2 escritas como mnemônicos. As cláusulas de Horn de H são chamadas de Relações semânticas;

d) R é um conjunto finito e não vazio de regras positivas e negativas definidas sobre relações de H. Essas regras regulam a noção de consequência lógica obtida a partir das relações semânticas.

As cláusulas de Horn descritas na Tabela 2 denotam os fatos concretos a respeito da realidade e são produzidas com base nas Especificações por meio de Traduções (seção 5). Tratam-se de cláusulas unitárias, em que o corpo é vazio (LLOYD, 1993). Com base nessas cláusulas unitárias, em R define-se um conjunto de predicados que denotam os fechos transitivos e relações derivadas desses fatos base. Esses predicados são cláusulas definidas em que o consequente representa um predicado a respeito da Teoria e o antecedente é baseado exclusivamente em relações básicas de R ou em outros predicados baseados em R. Por questão de limitação de espaço, não se apresenta neste artigo todas as definições predicativas contidas em uma teoria Ontoprolog, mas indica-se o exemplo da cláusula definida extensionof_irreflexive/2 (Definição formal 1), que demonstra como essas definições são construídas.

Definição formal 1 (extensionof_irreflexive/2). Trata-se do fecho transitivo da relação deo/2 que denota que se um conceito c ∈ C é uma subsunção de s1 ∈ C e s1 é uma subsunção de s2, então c é uma subsunção de s2, e s2 é um super tipo de c e de s1. A relação é definida do seguinte modo. Para uma variável T ∈ C e uma variável S ∈ C que denota um super tipo de T , temos que:

extensionof_irreflexive(*T*, *S*) ← *deo*(*T*, *S*).
extensionof_irreflexive(*T*, *S*) ← *deo*(*T*, *X*), *extensionof_irreflexive*(*X*, *S*).

Por definição, extensionof_irreflexive/2 é uma relação recursiva ascendente, transitiva e assimétrica, mas não reflexiva. Isso decorre do fato de extensionof_irreflexive/2 ser definida sobre deo/2, que é irreflexiva e assimétrica naturalmente.

4.1 Uma noção de consistência de modelos conceituais

Em uma cláusula de Horn, mais precisamente em uma cláusula definida, o consequente é formado precisamente por apenas um literal positivo (átomo). Por isso, não é possível diretamente escrever axiomas negativos – por exemplo, que denotem a expressão "não é o caso que ocorre que ϕ"– em uma teoria descrita em um arcabouço de Programação em Lógica, como é o caso de Ontoprolog. Por isso, axiomas negativos de Ontoprolog são escritos como regras positivas na forma h ← ϕ, em que h é o consequente da implicação, chamado de cabeça, que consiste precisamente de um átomo, e ϕ é um esquema de conjunções na forma $\psi 1 \wedge \ldots \wedge \psi_n$ para n > 0, em que ψ_i são literais que são possivelmente negativos. O conjunto de todas as cláusulas com o mesmo símbolo predicativo p na cabeça é chamado de definição de p. Portanto, uma regra p pode ser definida como um conjunto disjunto de literais.

No caso das traduções de regras negativas em regras positivas, a cabeça da regra denota uma situação inconsistente do programa – no caso, da Teoria. A conjunção das negações das cabeças de regras negativas traduzidas como regras positivas – doravante identificados pelo acrônimo RNP – cujo método Resolução implicam em vazio (∅), ou seja, a conjunção das negações das RNP que resultam em True, indicam que uma Teoria é consistente com todas as regras que a regem. Ou, dito de outra forma, ∅ indica que não há contra-exemplos que desautorizem a conclusão de que o programa seja consistente com as regras. Por exemplo, considere o axioma da Equação 1 escrito em Lógica de Primeira Ordem, que denotaria a noção de que não é o caso que exista um C que é extensão de si mesmo, ou seja, que não existe o caso reflexivo de extensionof_irreflexive/2 (Definição formal 1).

$$\forall C \; \neg \textit{extensionof\_irreflexive}(C, C) \tag{1}$$

Uma RNP equivalente à Equação 1 poderia ser escrita conforme a Equação 2, em que o consequente *extensionof_reflexive/*1 denota o caso em que acontece de existir na Teoria um *C* que torna a relação reflexiva *extensionof_irreflexive/*2 verdadeira.

$$\forall C \; (\textit{extensionof\_reflexive}(C) \leftarrow \textit{extensionof\_irreflexive}(C, C)) \tag{2}$$

Eliminando parênteses e quantificadores, conforme proposto por Lloyd (1993), a Equação 3 apresenta a forma final simplificada de uma RNP.

$$extensionof\_reflexive(C) \leftarrow extensionof\_irreflexive(C, C) \quad (3)$$

Nesse caso, um programador poderia definir um átomo *consistent/0*, conforme a Equação 4[7], de modo que *consistent/0* denote o caso em que não existe um *X* tal que *extensionof_reflexive*(*X*) seja o caso:

$$consistent \leftarrow \neg extensionof\_reflexive(\_) \quad (4)$$

Para um número *n* de RNP ϕ, *consistent/*0 seria definido como a Equação 5.

$$consistent \leftarrow \neg\phi 1 \wedge \ldots \wedge \neg\phi n \quad (5)$$

Um modo de tornar a solução mais geral, para predicados de qualquer aridade, é encapsular as RNP em um conjunto de cláusulas definidas de uma única cabeça, agrupadas por suas disjunções. Com esse objetivo, define-se o predicado ngrule/n (de negative global rule), em que n ≥ 1 é a aridade de ngrule, conforme a Definição 1:

Definição 1 (Regra de validação de teoria clássica)**.** A *regra de validação clássica* é um conjunto de disjunções na forma $ngrule(R_1, V_{11}, \ldots, V_{1m}) \vee \ldots \vee ngrule(R_n, V_{n1}, \ldots, V_{nz})$ em que R_i é uma constante que denota uma RNP e $V_{i1}, \ldots, V_{im}$ e $V_{n1}, \ldots, V_{iz}$ são variáveis que qualificam R_i, para $n \geq 0$, $m \geq 0$, $z \geq 0$ e i variando entre 1 e n.

—

Com essa definição, o termo predicativo ngrule possui aridade variável, por exemplo, a, e é identificado como ngrule/a, para a ≥ 1. Além disso, como ngrule/n possui aridade variável, e como não se está num arcabouço de ordem superior, a definição de consistent/0 deve ser feita para cada forma possível de ngrule/n. Com isso em tela, consistent/0 poderia ser definido conforme a Equação 6, para o caso de ngrule/2 e ngrule/3.

$$consistent \leftarrow \neg ngrule(\_, \_), \neg ngrule(\_, \_, \_) \quad (6)$$

[7] Na fórmula Equação 4, o símbolo _ (sublinhado) é uma variável anônima, ou seja, trata-se de uma variável existencialmente quantificada.

O que, de forma geral, para um *n* qualquer, a definição seria conforme a Equação 7.

$$consistent \leftarrow \neg ngrule(\_, \_), \neg ngrule(\_, \_, \_), \ldots, \neg ngrule(\_, \_, \ldots, \_) \quad (7)$$

Dessa forma, o exemplo da Equação 3 poderia ser reescrito conforme a Equação 8.

$$ngrule(extensionof\_reflexive, C) \leftarrow extensionof\_irreflexive(C, C) \quad (8)$$

Pelo fato de as regras de validade serem escritas de forma positiva, mas denotarem condições negativas, uma Teoria é considerada consistente quando nenhuma das regras sucede, ou seja, quando nenhuma delas é verdadeira, ou ainda, quando nenhuma delas "é o caso". Dito de outra forma, uma Teoria ψ é dita consistente em relação à disjunção das regras de validade Δ quando ψ, Δ |= ∅, para ∅ como um conjunto vazio de contra-modelos (contra-exemplos) que desautorizem a conclusão de que a teoria ψ seja inconsistente.

Uma implementação de Ontoprolog deve conter um conjunto dessas regras de validade (na forma de RNP) que regem as relações possíveis dos fatos descritos por meio das cláusulas definidas bases da Tabela 2. Um exemplo concreto de uma dessas regras é apresentado pela Definição formal 2 (Regras para dd/1), que rege a relação semântica dd/1, referente aos conceitos definidos como disjuntos. Um detalhamento exaustivo dessas regras para o caso concreto de Ontoprolog são apresentadas em Araujo (2015).

Tabela 2 – Cláusulas de Horn usadas na semântica de especificações Ontoprolog

Mnemônico da relação	Nome e interpretação	domínio
entity(*e*)	*entity*: entidade ontológica do universo de discurso, sobre os quais se atribui predicados. A variável *e* denota uma constante atribuída a cada conceito da ontologia no paradigma *Unique-Name Assumption*, ou seja, cada conceito distinto é representado por um e exatamente um *e*, e cada *e* distinto representa um e exatamente um conceito da ontologia, de modo que não existam e_1 e e_2 que representem o mesmo conceito, nem um conceito representado por dois *e* distintos	$e \in C$
dio(*i*, *m*)	*direct instance of*: instância direta, em que *i* é instância de *m*. Formalmente, entende-se que *i* é membro do conjunto *m*	$(i \cup m) \in C$
deo(*t*, *s*)	*direct extension of*: subsunção direta, em que *s* é o tipo superior de *t*	$(t \cup s) \in C$
dd([c_1, ... , c_n])	*direct disjuntion*: disjunção direta, em que os conceitos [c_1, ..., c_n] são disjuntos no sentido de que não é o caso que aconteça	$c_i \in C$ em que $i \in$ {1,

	um $c \in C$ ser instância de c_a e de c_b, ou instância de algum c_{a1} e de algum c_{b1} que estendam transitiva, reflexiva e respectivamente c_a e c_b, para $c_a \neq c_b$ e $c_{a1} \neq c_{b1}$,sendo $\{c_a, c_b\} \subseteq \{c_1, ..., c_n\}$	..., n}
dcomplete(c_t, [c_1, ..., c_n])	*direct complete subsumption*: subsunção direta completa, em que [c_1, ..., c_n] são exaustivamente todos os conceitos subordinados (ou que estendem) c_t, ou seja, não é o caso que exista um c_m que participa da relação *deo*(c_m, c_t) e que $c_m \notin \{c_1, ..., c_n\}$ se ocorre *dcomplete*(c_t, {c_1, ..., c_n}); cada ocorrência de *dcomplete*/2 denota uma partição conceitual das subsunções de c_t	($c_t \cup c_i$) $\in C$ em que $i \in \{1, ..., n\}$
pt(*t*,*w*)	*power type*: em que *w* é um *power type* de *t*. A ideia de *powertype* denota que se pode inferir que instâncias de *t* estendem *w*	($t \cup w$) $\in C$
dmo(*m*, *c*)	*direct meta-property on*: meta-propriedade direta de, em que é uma meta-propriedade atribuída à *c*	$m \in G$ e $c \in C$
dpo(*p*, *c*)	*direct property on*: propriedade direta em, em que *p* é uma propriedade em *c*	($p \cup c$) $\in C$
dpv(*at*(*p*, *c*), *v*)	*direct property value on*: valor da propriedade, em que *v* é o valor da propriedade *p* em *c*	($p \cup c$) $\in C$ e $v \in (C \cup G)$
dso(*s*, *t*)	*direct subset of*: subconjunto direto de, em que *s* é a entidade que denota o subconjunto do conjunto denotado por *t*	($s \cup t$) $\in C$
dro(*r*, *t*)	*direct redefinition of*: redefinição direta de, em que *r* é a entidade que denota a redefinição do conjunto denotado por *t*	($r \cup t$) $\in C$
defeasible(*r*)	*defeasible relation*: *r* é uma relação semântica retratável. Utilizada para definição de regras padrão (*default rules*), retratáveis na presença de fatos concretos	$r \in R$

Fonte: Os autores.

Definição formal 2 (Regras para dd/1)**.** dd/1 impõe restrições sobre as relações dio/2 e deo/2, de modo que nenhuma entidade pode instanciar ou estender simultaneamente dois tipos disjuntos. As seguintes regras definem a relação dd/1:

a) seja I a lista de entidades instanciadas por E. Nenhum par (i1, i2) ∈ I pode ser disjunto:

ngrule(*dio_disjoint_types*) ← *setof*(*IT*, *dio*(*E*, *IT*), *I*), *disjoint_types*(*I*)

b) seja I a lista de entidades estendidas por E. Nenhum par (i1, i2) ∈ I pode ser disjunto:

ngrule(*deo_disjoint_types*) ← *setof*(*IT*, *deo*(*E*, *IT*), *I*), *disjoint_types*(*I*)

—

5 Semântica de tradução

Nesta seção descreve-se a tradução das sentenças de Ontoprolog em cláusulas com conotação semântica presentes na Tabela 2. A tradução é definida por indução sobre a estrutura da sintaxe da linguagem: por meio de regras de produção, atribui-se semântica a cada possibilidade de combinação dos constructos sintáticos. Desse modo, a tradução apresentada é um tipo de semântica denotacional, criada com base em Nielson e Nielson (2007, p. 91), no sentido de que "há uma cláusula semântica para categoria sintática básica" e que "para cada método de construção de elementos compostos (na categoria sintática) há uma cláusula semântica definida em termos da função semântica aplicada aos constituintes imediatos dos elementos compostos." A tradução da sintaxe de Ontoprolog é definida com base em uma Função filtro, também chamada de função de interpretação (GUPTA, 1998, p. 152), conforme segue:

Definição 2 (Função filtro)**.** A função filtro, representada por $\Rightarrow$[8], é uma função semântica (NIELSON; NIELSON, 2007, p. 91) que, dado um contexto Δ, mapeia uma sentença ou um fragmento de sentença sintática α em um conjunto finito e não vazio β, que consiste em ao menos um caso de: zero ou mais relações semânticas; e zero ou mais sentenças ou fragmentos sintáticos diferentes de α; ou uma constante $\perp$, que denota que a função não é definida para a entrada α. Desse modo, a função filtro é uma função total. A função $\Rightarrow$ é definida por casos, em que cada caso α é uma sentença ou um fragmento de sentença delimitados por $[\![$ e $]\!]$ obtidos por indução sobre a estrutura das sentenças da gramática contida no Código 1. Uma sentença é uma instância dessa gramática. Um fragmento de sentença consiste em concatenações de relações semânticas com partes de sentenças sintáticas. Por exemplo, `a :: b` é uma sentença, uma vez que é uma instância completa da EBNF descrita no Código 1. Já $dio([\![a]\!]_t,[\![b]\!]_t)$ é um fragmento de sentença, porque representa um estágio intermediário entre uma relação semântica e uma sentença. Por sua vez, dio(a, b) é uma relação semântica, porque representa o estado final de uma relação desse tipo, uma vez que não contém parte delimitada por $[\![$ e $]\!]$.

—

A definição de todos os casos α de $\Rightarrow$ é a definição completa da semântica por tradução de Ontoprolog. A totalidade de casos α é a combinação possível de instâncias válidas da gramática EBNF definida no Código 1. Desse modo, considerando completa a definição de casos de $\Rightarrow$ em relação às definições sintáticas da linguagem, toda especificação composta por sentenças bem formadas baseadas nas regras EBNF denota uma Teoria de Ontoprolog, que por sua vez pode ser consistente ou inconsistente com as RNP que regem aquela Teoria.

Um algoritmo Σ que traduz um conjunto de sentenças sintáticas (instâncias do Código 1) e produz um conjunto de relações semânticas (cláusulas de H, conforme seção 4) é um algoritmo de ponto fixo que aplica $\Rightarrow$ recursivamente a cada sentença ou fragmento de sentença α do

[8] Lida como "denota".

conjunto de entrada e a cada sentença ou fragmento de sentença da imagem β da função. A cada iteração n do algoritmo (ou seja, a cada aplicação da função), o conjunto Δ é incrementado com as relações semânticas obtidas da imagem β da iteração anterior. O algoritmo é aplicado recursivamente até que uma determinada produção β_n seja a mesma de β_{n+1}, ou em algum β_i ocorra um $\perp$, caso em que toda a árvore de aplicações do algoritmo denota um erro.

Para exemplificar como a semântica por tradução é definida, ilustra-se um caso referente à gramática de descrição da relação de instância, denotada pelo operador :: (INSTANCE). A Figura 3 ilustra o diagrama rail road das regras gramaticais em questão.

No caso, o símbolo :: é preferido ao invés de uma palavra na língua inglesa por ser este símbolo comum na literatura da UFO e de UML, de modo que é natural para os especialistas e pode igualmente ser oralizado como "instancia".

A construção padrão é a que denota que uma única entidade P é instância de um único type T:

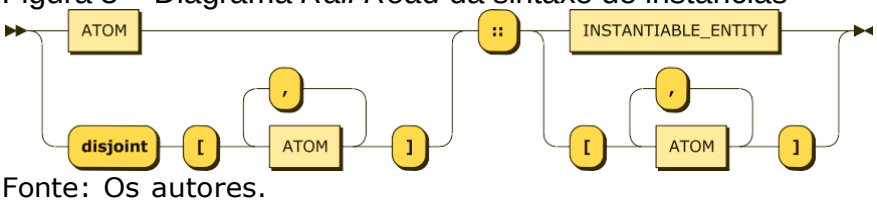
$$[\![P :: T]\!]$$
$$\Rightarrow$$
$$dio([\![P]\!]_t, [\![T]\!]_t)$$
$$[\![[P,T]]\!]_{entity}$$
$$\text{se } P \neq T$$

Outro caso é o que denota que um conjunto de conceitos são, todos, instância de um único type T :

$$[\![[P_1, \ldots, P_n] :: T]\!]$$
$$\Rightarrow$$
$$[\![P_1 :: T]\!]$$
$$\vdots$$
$$[\![P_n :: T]\!]$$

Ou então, quando um único P é instância de múltiplos T, caso que denota a múltipla instanciação, ou seja, quando um conceito P é instância simultânea direta de mais que um tipo:

Figura 3 – Diagrama *Rail Road* da sintaxe de instâncias



Fonte: Os autores.

$$[\![P :: [T_1, \ldots, T_n]]\!]$$
$$\Rightarrow$$
$$[\![P :: T_1]\!]$$
$$\vdots$$
$$[\![P :: T_n]\!]$$

Como última combinação possível das listas, apresenta-se a produção para o caso em que há listas em cada lado do operador `::` em tela:

$$[\![[P_1, \ldots, P_n] :: [T_1, \ldots, T_n]]\!]$$
$$\Rightarrow$$
$$[\![P_1 :: T_1]\!]$$
$$\vdots$$
$$[\![P_n :: T_1]\!]$$
$$[\![P_1 :: T_m]\!]$$
$$\vdots$$
$$[\![P_n :: T_m]\!]$$

Por fim, acrescenta-se a definição do açúcar sintático que combina o operador `disjoint`, em que α e β estão, respectivamente, antes e depois do operador de instância:

$$[\![\texttt{disjoint}\ \alpha :: \beta]\!]$$
$$\Rightarrow$$
$$[\![\texttt{disjoint}\ \alpha]\!]$$
$$[\![\alpha :: \beta]\!]$$

6 Semântica de tradução

O Código 3 contém um exemplo de uma especificação Ontoprolog construída utilizando a sintaxe base definida no Código 1. Trata-se de uma Especificação de Teoria construída como instância direta da Metateoria de Hypertypes. No exemplo, duas propriedades, `nome` e `data_morte`, são criadas como instâncias do hypertype `slotProperty`. Uma hierarquia de entidades é definida no âmbito de uma teoria de tipos: as entidades `ser`, `pessoa`, `vivo` e `morto` são definidas como instâncias de `entity`, e relacionadas formalmente entre si por meio da relação de subsunção. Além disso, `vivo` e `morto` são definidas como entidades disjuntas. As propriedades `nome` e `data_morte` são atribuídas, respectivamente, a `pessoa` e a `morto`. Com base nessa construção, define-se as noções de `pessoa_viva`, como sendo uma entidade que estende características de `pessoa` e de `vivo`, e de `pessoa_morta`, como extensão

de `pessoa` e de `morto`. Por fim, define-se uma teoria de indivíduos, em que lauro é uma instância de `pessoa_viva`; e `sinatra` é instância simultânea de `pessoa_morta` e de `vivo`. Um valor é atribuído à propriedade `nome` em `lauro`.

Código 3 – Exemplo de instanciação da linguagem

```
% properties
[nome, data_morte] :: slotProperty.

% type theory
ser :: type.
pessoa :: type extends ser.

disjoint [vivo, morto] :: type.

property nome on pessoa.
property data_morte on morto.

pessoa_viva :: type extends [pessoa, vivo].
pessoa_morta :: type extends [pessoa, morto].

% individual theory
lauro :: pessoa_viva.
nome at lauro := 'lauro cesar'.

sinatra :: [pessoa_morta, vivo].

:- otp_compile,
   check_semantics.
```

O Código 4 contém a saída da avaliação semântica do Código 3. O predicado `opt_compile/0` consiste na execução do filtro de tradução (seção 5) e na produção das cláusulas que denotam as relações semânticas. O predicado `check_semantics/0`, por sua vez, aplica as NRP na Teoria produzida pelo filtro e verifica a noção de consistência do modelo conceitual. Como era esperado, o mecanismo de avaliação da Teoria denotada pela Especificação verifica que há uma inconsistência a respeito do conceito `sinatra`, capturado pela regra dio_disjoint_types (Definição formal 2). No caso, `sinatra` não pode instanciar simultaneamente os tipos `pessoa_morta` e `vivo` por serem disjuntos. Esse resultado demonstra o mecanismo que define a noção de consistência no âmbito das teorias de Ontoprolog, conforme a seção 4.

Código 4 – Fragmento da saída da avaliação semântica do Código 3

```
Getting candidate sentences...
Starting Ontoprolog compilation...
Retracting current semantic terms...
Checking syntax of Ontoprolog candidate sentences...
Compiling sentences ...
  iteration 1...
All sentences are syntactically well-formed.

Asserting semantic terms produced by sentence decoding... 667
Asserting semantic expansions helpers...
   asserting semantic_expansion_dio_hierarchy/2...197
   asserting semantic_expansion_deo_hierarchy/2...387
Asserting semantic expansions...606
Asserting extensionof_irreflexive/2 relations...387
Asserting instanceof/2 relations...199
Asserting drefine/3 relations...8
Asserting propertyon/2 relations...80

Compilation of the Ontoprolog specification was successful.

Checking semantics using negative rules of the current Ontoprolog theories...
--------------------------------------------------
```

```
Checking ngrule/3...
 sinatra:
   -> "sinatra" instantiates disjoint types "[pessoa_morta,vivo]". (dio_disjoint_types)
-------------------------------------------------
```

Com base na Especificação e na Teoria por ela denotada, é possível construir programas que utilizem ou forneçam informações a respeito de ontologias. Aplicações no âmbito de sistemas especialistas podem utilizar a estrutura ontológica de descrição como insumo a decisões e avaliações de regras. Por exemplo, o Código 5 apresenta como uma teoria de indivíduos baseada na Teoria de universais apresentada na Código 3 poderia ser preenchida com base em um banco de dados existente.

Código 5 – Exemplo de integração com programa padrão

```
% database
% customer(ID, NAME, BIRTH_COUNTRY)
customer(1, 'Lauro Cesar', brazil).
customer(2, 'Picasso', spain).
customer(3, 'Sinatra', eua).
% dead(ID)
dead(2).
dead(3).

% individual theory based on universal theory from Código 3
ID :: pessoa_viva :-
   customer(ID, _, _),
   \+ dead(ID).

ID :: pessoa_morta :-
   customer(ID, _, _),
   dead(ID).

:- otp_compile,
   check_semantics.
```

Isso demonstra como especificações ontologicamente bem fundamentadas em metateorias de Ontoprolog podem ser integradas com banco de dados e programas comuns, inclusive os já existentes.

7 Conclusão e trabalhos futuros

Este artigo apresentou a abordagem Ontoprolog de tratamento lógico da informação referente à especificação de discursos sobre ontologias. O sistema é definido sobre o arcabouço da Programação em Lógica e utiliza o metamodelagem para prover suporte a ontologias de fundamentação. Uma teoria base, chamada Metatoria de Hypertypes, foi apresentada como um constructo de referência inicial.

Embora a lógica subjacente da versão de Ontoprolog apresentada neste artigo seja o fragmento de Lógica Clássica com as extensões não monotônicas da abordagem padrão de Programação em Lógica, a estratégia adotada com o arcabouço permite que diferentes tipos de lógicas sejam utilizadas para a especificação de raciocínios sobre ontologias. Por exemplo, a lógica subjacente padrão pode ser substituída por um dos sistemas modais de Nguyen (2009). Isso possibilita a utilização de modalidades epistêmicas de multiagentes para raciocinar sobre discursos produzidos por diferentes sujeitos. Além da substituição da lógica subjacente, a Metateoria de Hypertypes pode ser utilizada como base para definição da ontologia UFO, de modo que esta funcione como ontologia

de fundamentação mais detalhada a ontologias específicas de domínio definidas em Ontoprolog. Nesse cenário, ontologias de domínio seriam definidas como instâncias da ontologia UFO, que por sua vez seria definida como instância da Metateoria de Hypertypes.

Parte desses resultados já foram obtidos, como a definição da ontologia UFO como instância da Metateoria de Hypertypes, bem como a extensão de Ontoprolog com sistemas de semântica epistêmica de MProlog, que permitem que aspectos de integração de ontologias sejam explorados. Esses resultados serão apresentados em outras publicações. Ainda assim, a implementação de referência de Ontoprolog, que contempla esses resultados adicionais, pode ser obtida em http://github.com/laurocesar/ontoprolog.

Referências

ALLIOT, J.-M.; HERZING, A.; LIMA-MARQUES, M. Implementing Prolog extensions: A parallel inference machine. In: Institute for New Generation Computer Technology ICOT (Ed.). *Proceedings of the International Conference of Fifth Generation Computer Systems***.** [S.l.]: IOS Press, 1992. v. 2, p. 833–842.

ALMEIDA, M. B. Uma abordagem integrada sobre ontologias: Ciência da Informação, Ciência da Computação e Filosofia. *Perspectivas em Ciência da Informação*, v. 19, n. 3, p. 242-258, 2014.

ARAUJO, L. C. *Uma linguagem para formalização de discursos com base em ontologias***.** Tese (Doutorado em Ciência da Informação) — Universidade de Brasília, Faculdade de Ciência da Informação, CPAI/UnB, Brasília, DF, 2015. Disponível em: <http://repositorio.unb.br/handle/10482/19319>. Acesso em: 3 jun 2017.

BALBIANI, P.; HERZIG, A.; LIMA-MARQUES, M. TIM: The Toulouse inference machine for non-classical logic programming. In: BOLEY, H.; RICHTER, M. M. (Ed.). Processing Declarative Knowledge. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence). Berlin, Heidelberg: Springer, 1991. v. 567, p. 365–382.

BRAGA, B. F. B. et al. Transforming OntoUML into Alloy: towards conceptual model validation using a lightweight formal method. Innovations Syst Softw Eng, v. 6, p. 55-63, 2010.

CHRISTIANSEN, H. Using prolog as metalanguage for teaching programming language concepts. *In:* KACPRZYK, J.; KRAWCZAK, M.; ZADROZNY, S. (Ed.). *Issues in Information Technology*. Warszawa: EXIT, 2002. p. 59-82.

FARIÑAS DEL CERRO, L. Molog: A system that extends prolog with modal logic. New Generation Computing, Springer-Verlag, v. 4, n. 1, p. 35-50, 1986.

FRÜHWIRTH, T. Theory and practice of constraint handling rules. *The Journal of Logic Programming*, v. 37, n. 1-3, p. 95-138, 1998.

GUIZZARDI, G. *Ontological foundations for structural conceptual models*. Tese (Doutorado em 2005) — Centre for Telematics and Information Technology, University of Twente, Enschede, The Netherlands, 2005.

GUIZZARDI, G. Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. *In:* INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 33., 2014. *Proceedings***...** Atlanta, USA: [s.n.], 2014.

GUIZZARDI, G.; WAGNER, G. Using the Unified Foundational Ontology (UFO) as a Foundation for General Conceptual Modeling Languages. In: POLI, R.; HEALY, M.; KAMEAS, A. (Ed.). Theory and applications of ontology: computer applications. Netherlands: Springer Netherlands, 2010. p. 175-196.

GUIZZARDI, G.; ZAMBORLINI, V. Using a trope-based foundational ontology for bridging different areas of concern in ontology-driven conceptual modeling. Science of Computer Programming, v. 96, part. 4, p. 417-443, December 2014.

GUPTA, G. Horn logic denotations and their applications. *In*: APT, K. R. et al. (Ed.). *The Logic Programming Paradigm:* a 25 year perspective (Proceedings of Workshop on Current trends and Future Directions in Logic Programming Research). Springer Berlin Heidelberg, 1998. (Artificial Intelligence), p. 127-160. Disponível em: <http://www.cs.nmsu.edu/ gupta/logden/lnai.ps>. Acesso em: 3 dez 2013.

HOFKIRCHNER, W. Toward a new science of information. *Information,* v. 2, p. 372-382, 2011.

KOWALSKI, R. Predicate logic as programming language. Information Processing, North-Holland Publishing Company, v. 74, p. 569-574, 1974.

LIMA-MARQUES, M. *De la connaissance à la paraconsistance*: un modèle d'application pour la résolution des conflits aériens. Tese (Doutorado em 1992) — L'Universite Paul Sabatier de Toulouse, Toulouse Cedex – France, Décembre 1992.

LIMA-MARQUES, M. Outline of a theoretical framework of Architecture of Information: a school of Brasilia proposal. *In:* BÉZIAU, J.-Y.; CONIGLIO, M. E. (Ed.). *Logic without frontiers:* festschrift for Walter Alexandre Carnielli on the occasion of his 60th Birthday. London: College Publications, 2011, p. 311-320. (Tribute Series, v. 17).

LLOYD, J. W. *Foundations of logic programming*. 2. ed. ex. Germany: Springer-Verlag, 1993. (Symbolic Computation).

LOWE, E. J. *The Four-Category Ontology*: a metaphysical foundation for natural science. Oxford: Clarendon Press, 2006.

NGUYEN, L. A. Modal logic programming revisited. *Journal of Applied Non-Classical Logics,* v. 19, n. 2, p. 167-181, 2009.

NIELSON, H. R.; NIELSON, F. Semantics with applications: an appetizer. London: Springer-Verlag, 2007. (Undergraduate topics in Computer Science).

REITER, R. Deductive question-answering on relational data bases. In: GALLAIRE, H.; MINKER, J. (Ed.). Logic and Data Bases. [S.l.]: Springer US, 1978. p. 149-177.

RODRIGUES, T. G. *Sobre os fundamentos da programação lógica paraconsistente.* Dissertação (Mestrado em 2010) — Departamento de Filosofia do Instituto de Filosofia e Ciências Humanas da Universidade Estadual de Campinas, Campinas, SP, Setembro 2010.

SALES, T. P. Ontology validation for managers. Dissertação (Mestrado em 2014) — Universidade Federal do Espírito Santo, Departamento de Informática, Vitória, Outubro 2014.

SNEYERS, J. *et al.* As time goes by: constraint handling rules: a survey of CHR research from 1998 to 2007. *Theory and Practice of Logic Programming*, v. 10, p. 1-47, 1 2010.

SWIFT, T.; WARREN, D. S. XSB: Extending prolog with tabled logic programming. *Theory and Practice of Logic Programming,* v. 12, p. 157-187, jan. 2012.

W3C. *Extensible Markup Language (XML) 1.0.* 5. ed. [S.l.], 2008. Disponível em: <http://www.w3.org/TR/REC-xml/#sec-notation>. Acesso em: 5 dez. 2013.

1