3H3-03

リスプによる RDFS プロセッサ
A Lisp-based RDFS Processor

小出 誠二 Seiji Koide　川村 正則 Masanori Kawamura　永野 進 Susumu Nagano

株式会社ギャラクシーエクスプレス
Galaxy Express Corporation

Semantic Web is a promising application area for Lisp programming language as well as Prolog, because basic requirements for semantic web processing such as axiom, entailment, and property constraint are also basic functionalities in AI reasoning. Roughly speaking, from the viewpoint of the implementation, semantic web processors fall into a few categories; statement-based (XML-based), graph-based (triple-based), and object-based (or frame-based). Despite that there are diverse semantic web toolkits in various programming languages; there is no object-based realization in Lisp. We have developed an object-based RDFS processor on top of Common Lisp Object System (CLOS) in reflective object-oriented approach, where RDFS classes are straightforwardly mapped to CLOS class objects and RDFS instances are mapped to CLOS instance objects. The Meta-Object Protocol (MOP) of CLOS allowed us to program the reflection in RDFS, that is, every RDFS class is an instance of rdfs:Class exactly according to RDFS W3C Recommendation.

1. はじめに

次世代ウェブ技術として，セマンティックウェブが注目されている．RDF(Resource Description Framework)はウェブ上の情報についてメタ記述するための表記方法であり，RDFS(RDF-Schema)は RDF 表記によるスキーマ記述のための最小限のボキャブラリである．先にシャンクらの Memory Organization Package を CLOS (Common Lisp Object System)上で開発し，それをさらに RDFS プロセッサに拡張した結果について述べたが[小出 03]，そこでは rdfs:subPropertyOf の実装を行っていなかった．RDFS はプロパティ中心のオントロジ記述言語であり，オントロジ記述において RDFS プロパティの rdfs:domain(領域)と rdfs:range（値域）の制約が重要な役割を果たす．プロパティに包摂(subsumption)概念を持ち込むものが rdfs:subPropertyOf であり，これなくしては RDFS 処理系実装の価値が半減する．今回，RDFS の公理(axiom)やその伴意ルール(entailment rule)を完備した処理系を開発した．

セマンティックウェブ処理系は，用いる計算機言語にかかわらず基礎となるデータ表現に何を用いるかによって，大きくわけて宣言ベース，グラフベース，オブジェクトベースに分けられる．宣言ベースは DOM のような RDF/XML 表記の素直な計算機内表現であり，これをもとに RDFS や OWL の意味論を実装することは困難である．グラフベースは N-Triple 表記の計算機内表現であり，この実装ではノードとエッジをエンティティとして計算機内部に有向グラフを生成する．RDF モデル論では RDF グラフがモデル論の理論的基礎となっていることから，グラフベースの実装において RDF あるいは RDFS の意味論を実現するのは容易であるが，オブジェクト指向的な OWL（オブジェクト中心の記述でプロパティに局所的制約を持つ）の意味論の効率的な実現には困難が予想される．一方，オブジェクトベースの実装では，RDF グラフ中の一つのノードを中心に，それから放射するエッジとその先のノードまでの情報をまとめて一つにしたものを計算機実現上でも一つのオブジェクトの実装とする．この実装では RDF あるいは RDFS の意味論を実現することはグラフベースに比べて難しいが，OWL の意味論においては効率的で自然な実現が比較的容易である．

RDFS においてはすべてのリソースはクラスと呼ばれるグループに分けられ，クラスの要素はそのクラスのインスタンスと呼ばれる．クラスはそれ自身リソースでもある*[1]．さらに，OWL Full においては，クラスは個体(individual)の収集物(collection)として扱われると同時にそれ自身固有の性質を持つ個体でもあるとされている*[2]．すなわち，クラスはそれ自身あるクラスのインスタンスでもある．このような関係はリフレクティブなプログラミング言語実現の基本方略として長年研究されてきているが[Smith 84] [Kiczales 92] [Paepcke 93]，一見理解しがたく RDFS の意味論解釈においてもセマンティックウェブ研究者の一部に混乱を引き起こしている[Pan 03].

オブジェクト指向プログラミングは，C++や Java の普及により今日では何ら特別なことではなく，それはセマンティックウェブにおいても同様であるが，先に述べた意味でのオブジェクトベースのセマンティックウェブの実装は数少なく，さらに本来リフレクティブな構造を持つセマンティックウェブ用マークアップ言語でありながら，定義どおりにリフレクティブな構造と機能を実現したオブジェクトベースの処理系は提供されていない．

オブジェクト機能を持たない Prolog では三つ組みの宣言的表現が Prolog のファクトと本質的に同等であることから，伴意ルールを書くことにより RDFS や OWL の意味論の実装は容易であるが，rdfs:subClassOf や rdfs:subPropertyOf の効率的な実装が必要[Wielemaker 03]となる．

我々は文科省 IT プロジェクト「IT を活用した大規模運用システムの支援システムの構築」[小出 02][Koide 03]において，ロケット打上げを題材として大規模運用支援システムを開発しているが，ここではセマンティックウェブ技術とウェブサービス技術の活用を目指している．OWL 記述ドメインオントロジと OWL-S 記述タスクオントロジの処理を目標に，RDF/XML 記述を読み，RDFS 公理と伴意関係を処理する RDFS プロセッサを開発した．この処理系では Common Lisp Object System (CLOS)によるオブジェクトを基本エンティティとし，リフレクティブなプログラミング

連絡先：小出誠二，(株)ギャラクシーエクスプレス，港区浜松町 1-18-16，Fax:03-5733-7190，koide@galaxy-express.co.jp

[1] http://www.w3.org/TR/rdf-schema
[2] http://www.w3.org/TR/owl-guide/

機能を提供するメタオブジェクト・プロトコル(MOP)を用いてRDFS の意味論をオブジェクトベースで実装した．なおここで「リフレクティブ」とは，言語の実現機構をその言語を用いて変更できるという意味であり，C#や Java におけるリフレクティブ機能では，プログラム中で呼び出すべきメソッドの動的な決定と呼出しを実行することはできるが，言語の機能を変更することはできないことを注意しておく．

2. RDFS 意味論

2.1 クラス，インスタンス，メタクラス

RDFS において rdf:type プロパティはあるリソースがあるクラスのインスタンスであることを述べるために用いられる．もしあるクラス C が他のクラス C' のサブクラスであれば，そのクラス C のすべてのインスタンスは C' のインスタンスでもある．RDFS ドキュメントに記載されたリソースについて，rdf:type，rdfs:subClassOf，rdfs:subPropertyOf 関係を図示すると図1のようになる．CLOS を用いて図1に示されたクラスとプロパティを実装するにあたり，すべての RDFS リソースを CLOS オブジェクトと捉え，RDFS リソース間の rdf:type 関係を CLOS クラス・インスタンス関係に，rdfs:subClassOf 関係を CLOS サブクラス・スーパクラス関係に写像した．CLOS のクラスは Common Lisp の型になるため，このような素直な写像を行うことにより，特にプログラムを必要とせずに，Common Lisp の型機能により包摂律と rdfs:subClassOf の推移律を実行することができる．

たとえば，本システムにおいて N-Triple のS式表現のためのマクロを用いて以下のように入力したとき，

```
(defTriple
 vin:Wine rdfs:subClassOf
 food:PotableLiquid)
(defTriple
 vin:Zinfandel rdfs:subClassOf vin:Wine)
(defTriple
 vin:ElyseZinfandel rdf:type vin:Zinfandel)
```

リスプの型判定述語 typep を用いて[CLtL2]，以下のように包摂律（RDFS 伴意ルール **rdfs9**，すなわちこの推論機能は RDF 意味論*[1]における伴意ルール **rdfs9** に相当する，以下同様）を推論することができるし，

```
(typep vin:ElyseZinfandel vin:Wine) → t
(typep vin:ElyseZinfandel food:PotableLiquid) →t
```

リスプの副型を判定する述語 subtypep を用いて，以下のように推移律（RDFS 伴意ルール **rdfs11**）を推論することができる．

```
(subtypep vin:Zinfandel food:PotableLiquid)
 → t
```

図1からわかるように，rdfs:Class と rdfs:Datatype を除くクラスはこれらのクラスのインスタンスである．逆に rdfs:Class と rdfs:Datatype はその他クラスのクラスであり，CLOS においてそのようなクラスはメタクラスと呼ばれる[CLtL2]．ここで特徴的なことは rdfs:Class は自身のインスタンスであり，rdfs:Datatype は rdfs:Class の rdfs:subClassOf であると同時にインスタンスでもあることである．このような関係はメタ巡回(meta-circularity)と呼ばれ，一部では定義に矛盾をもたらすものとされているが[Pan 03]，リフレクティブ・プログラミングにおいては重要な実現技術とされてきた．すなわち，リフレクティブ・プログラミングでは実行時にシステム定義を変更したり，プログラミング言語仕様の変更をユーザに開放したりすることを目的に，インスタンスとそれを制御するクラスの関係と同様に，システムクラスとそれを制御するメタクラスを設定し，このメタクラスのプログラム機能をユーザに開放している．理論的にはこのリフレクティブなレイヤー階層は何段にもなり得るが，実際にはメタクラスのクラスは自分自身として無限に階層が伸びていくことを終端させている．RDFS もあきらかにこのリフレクティブ・プログラミングの考え方を取り入れており，rdfs:Class は自分の型は自分自身であるとしている．CLOS における standard-class は（自身と standard-object も含め）すべてのクラスをインスタンスとするクラスであり，standard-object は（standard-class も含め）すべてのクラスのスーパクラスとなっているが，rdfs:Class（すべてのクラスのクラス）と rdfs:Resource（すべてのクラスのスーパクラス）の関係はそれと全く同一である．

RDFS のクラスを CLOS クラスに RDFS のインスタンスを CLOS インスタンスに写像した場合，rdfs:Class と rdfs:Datatype は CLOS メタクラスに写像される．実装上これは困難な問題を引き起こす．通常は母型となる型が先にあって，その情報を用いてインスタンスが定義されるが，rdfs:Class においてはこれから定義しようとするときに自分は存在していない．この問題は CLOS の実装でも同様であり（`standard-class` について）ブートストラッピングの手法が研究開発されている[Kiczales 92]．RDFS 処理システムをスクラッチで開発する場合には，リフレクティブ言語に関する過去の経験に学んで RDFS 処理システムを開発することになるが，本システムは CLOS システム上に実装したため，既存のリフレクティブシステム上にどのように RDFS リフレクティブを実現するかという問題になる．そこで問題となったのが CLOS システム上での rdfs:Class の取り扱いであった．rdfs:Resource と rdfs:Class との間接的メタ巡回，および rdfs:Class と rdfs:Datatype との間接的メタ巡回は実現できても，現在の実装言語である Allegro Common Lisp では明示的に自己参照しようとするとエラーとなる．そこでやむを得ず，rdfs:Class のみは内部の実装として仕様とは異なって自分とは異なるクラスとし，外部から型判定文やメソッドで見たときには RDFS の仕様通りになるように，型判定が RDFS 仕様に従うような `gx:subtypep`，`gx:type-of`，`gx:typep` を導入した．実装上の詳細は省略するが，結果として以下に示すように，メタ巡回も含め，RDFS の仕様とおりのタイプ関係を可能にした．

```
(gx:typep rdfs:Class rdfs:Class) → t
(typep rdfs:Class rdfs:Resource) → t
(typep rdfs:Resource rdfs:Class) → t
(typep rdfs:Resource rdfs:Resource) → t
(typep rdfs:Datatype rdfs:Class) → t
(typep rdfs:Datatype rdfs:Resource) → t
```

2.2 プロパティの領域制約と値域制約

RDFS は Java のようなオブジェクト指向プログラム言語と似ているところがあるが，プロパティ中心の言語であり，クラス定義においてプロパティがクラス記述に用いられるのではなく，まずプロパティが適応されるインスタンスのタイプを考えてプロパティを定義することが中心となっている．プロパティの領域属性(rdfs:domain)は N-Triple 表記におけるサブジェクトのリソースタイプを制約し，値域属性(rdfs:range)はオブジェクトのリソースタイプを制約する．

[1] http://www.w3.org/TR/rdf-mt/

あるプロパティに既に定義済みの領域･値域制約があって, そのプロパティをプレディケイトに持つ三つ組みが入力されたとき, 入力サブジェクトとオブジェクトが制約に違反しないか, チェックされなければならないし, もしそのサブジェクトやオブジェクトが未定義であれば, それは制約となるべきクラスのインスタンスとして定義されなければならない (RDFS 伴意ルール **rdfs2**, **rdfs3**). この機能は CLOS のメソッドとメタオブジェクト･プロトコル機能を用いて実現された.

たとえば以下のような三つ組み入力が行われたとき, もし vin:hasColor が未定義ならばそれは rdfs:domain や rdfs:range の領域制約から伴意ルールにより rdf:Property のインスタンスと定義されるし, vin:Wine や vin:WineColor が未定義ならばそれは値域制約から rdfs:Class のインスタンスと定義される. このような制約の働きはユーザ定義のプロパティについても同様であり, 下記例では vin:ElyseZinfandel が未定義ならば, それは vin:Wine のインスタンスとして定義されるし, vin:Red が未定義ならば vin:WineColor のインスタンスとして定義される.

```
(defTriple
 vin:hasColor rdfs:domain vin:Wine)
(defTriple
 vin:hasColor rdfs:range vin:WineColor)
(defTriple
 vin:ElyseZinfandel vin:hasColor vin:Red)
```

また, この領域と値域の制約は, 制約として指定されたクラスのスーパクラスについても包摂律によって成立する. すなわち, 以下に示すように, あるプロパティの領域あるいは値域制約が成り立てば,

```
(defTriple
 vin:hasColor rdfs:domain vin:Wine)
(defTriple
 vin:hasColor rdfs:range vin:WineColor)
(defTriple
 vin:Wine rdfs:subClassOf food:PotableLiquid)
(defTriple
 vin:WineColor rdfs:subClassOf
 vin:WineDescriptor)
```

その領域あるいは値域のスーパクラスについても制約が成り立つ (RDFS 伴意ルール **ext1**, **ext2**). 本システムでは, この伴意ルールを領域と値域の判別関数 domainp および rangep で実行するようにした. 実行例を以下に示す.

```
(domainp vin:hasColor food:PotableLiquid) → t
(rangep vin:hasColor vin:WineDescriptor) → t
```

2.3 rdfs:subPropertyOf 関係

通常のオブジェクトシステムでは, クラス間にスーパ･サブ関係があるがインスタンス間にはない. ところが, RDFS では rdf:Property のインスタンスであるプロパティが rdfs:subPropertyOf を有する. この rdfs:subPropertyOf は rdfs:domain や rdfs:range などの RDFS プロパティの属性に関する包摂概念をもたらす(RDFS 伴意ルール **ext3**, **ext4**). この機能によってたとえば rdfs:member の下位プロパティである rdf:_1 や rdf:_2 のように, 実行時に動的にプロパティが生成されるような場合に, あらかじめそのようなプロパティに属性を定義するようなことが可能になる. [Wielemaker 03]はこの機能を利用して, WordNet における wns:hyponymOf を rdfs:subClassOf の下位プロパティとして定義した.

本システムでは以下に示すように, `get-domain`, `get-range` 関数によって, あるプロパティの上位プロパティに定義された領域情報や値域情報を取り出すことができるようにした. これらの関数はあるプロパティとその上位プロパティに存在する領域情報や値域情報のうち, 最も特殊な値を, すなわち rdfs:subClassOf 関係で最も下位のものを取り出す. また, 領域や値域に関する包摂概念のための判定述語, `domainp` および `rangep` を用意した.

たとえば以下のような入力があったとき,

```
(defTriple
 vin:hasWineDescriptor  RDFS:domain  vin:Wine)
(defTriple
 vin:hasWineDescriptor  RDFS:range
 vin:WineDescriptor)
(defTriple
 vin:hasColor  RDFS:subPropertyOf
 vin:hasWineDescriptor)
```

vin:hasColor 自身には領域と値域が定義されていないため, 下記のようにメソッド `rdfs:domain` ではエラーを起こし, 関数 `domain-value` は `cl:nil` を返すが, `get-domain` や `get-range` では上位プロパティである vin:hasWineDescriptor の値を継承する.

```
gx(12): (rdfs:domain vin:hasColor)
Error: The slot rdfs:domain is unbound in the
object
#<rdf:Property vin:hasColor> of class
#<rdfs:Class rdf:Property>.
[condition type: unbound-slot]
gx(13): (domain-value vin:hasColor)
common-lisp:nil
gx(14): (get-domain vin:hasColor)
#<rdfs:Class vin:Wine>
gx(15): (get-range vin:hasColor)
#<rdfs:Class vin:WineDescriptor>
gx(16): (domainp vin:hasColor vin:Wine)
t
gx(17): (rangep vin:hasColor vin:WineDescriptor)
t
```

3. 簡単な実行例

旧 RDFS ドキュメント*[1]に掲載された簡単な RDF グラフの本システムにおける入力例を示す. これまで示したように三つ組みでも入力できるが, 以下に示すように RDF/XML 表記により近いS式によるオブジェクト表現としても入力することができる. この入力例では RDF グラフ中のブランクノードがここでは `eg:author` スロットのフィラーの無名オブジェクト記述として自然に表現されていること, 伴意ルールが実行されていることに注意されたい.

```
gx(4): (defProperty eg:name
  (rdfs:domain eg:Person)
  (rdfs:range rdfs:Literal))
Warning: Entail: eg:Person rdf:type rdfs:Class.
eg:name
gx(5): (defProperty dc:title
  (rdfs:domain eg:Document)
  (rdfs:range rdfs:Literal))
Warning: Entail: eg:Document rdf:type rdfs:Class.
dc:title
```

[1] http://www.w3.org/TR/2002/WD-rdf-schema-20021112/

```
gx(6): (defProperty eg:author
  (rdfs:domain eg:Document)
  (rdfs:range eg:Person))
eg:author
gx(7): (defResource eg:Person
  (rdfs:subClassOf eg:Agent))
Warning: Entail: eg:Agent rdf:type rdfs:Class.
eg:Person
gx(8): (defResource eg:Document
  (rdfs:subClassOf eg:Work))
Warning: Entail: eg:Work rdf:type rdfs:Class.
eg:Document
gx(9): (defIndividual eg:Proposal
  (rdf:type eg:Document)
  (eg:author (eg:Person (eg:name "Tim Berners-
 Lee")))
  (dc:title "Information Management: A Proposal")
  (rdf:about "http:/…/Proposal/"))
eg:Proposal
```

このような入力のあと, ユーザは CLOS の `slot-value` 関数を利用して任意の情報を取り出すことができるが, そのほかに RDF グラフパスを与えてパスに沿った値の取り出し関数を用意した. これはグラフ上を与えられたパスに従ってトラバースする行為に相当する.

```
gx(10): (print-form eg:Proposal)
(eg:Document eg:Proposal
 (rdf:about "http:/…/Proposal/")
 (eg:author (eg:Person (eg:name "Tim Berners-
 Lee")))
(dc:title "Information Management: A Proposal"))
gx(11): (<- eg:Proposal dc:title)
"Information Management: A Proposal"
gx(12): (<- eg:Proposal eg:author eg:name)
"Tim Berners-Lee"
gx(13): (<- eg:Proposal eg:author rdf:type)
eg:Person
```

4. 単調性と前方参照

三つ組み入力あるいは RDF/XML 表記の入力は前方参照可能であることが要求されている. 幸いなことに, RDFS は知識の単調性(monotonicity)が前提となっている. すなわち, 知識は増加方向に一方的に追加され, 詳細化されることはあってもそれまでの知識が新しい知識で否定されることはない. 言い換えれば, 同一ファイル内で前方参照があったとしても, 最初の前方参照時に最も汎用の知識を仮定してオブジェクト実体を実現しておき(たとえば, クラスならそのメタクラスを rdfs:Class とし rdfs:Resource をスーパクラスとする, インスタンスなら rdfs:Resouce のインスタンスとし, プロパティなら rdf:Property のインスタンスとする), 後方で実際の定義が現れたときにその定義に従った詳細化を行えばよい. もし最初に詳細な知識が入力され, その後抽象的な知識がたとえ入力されても, 関連する rdfs:subClassOf 関係にあればそれは無視し, 独立した rdfs:subClassOf 関係にあればその情報を追加すればよい.

RDFS の伴意ルールは, 前方参照を前提として未定義なエンティティが引用されたときどのような定義を行えばよいかという問題の解決に役立つ. たとえば, RDF 伴意ルール **rdf1** を用いて, 三つ組みのプレディケイトに現れた未定義なプロパティは rdf:Property のインスタンスとして定義する. プロパティではないエンティティは, RDFS 伴意ルール **rdfs2** と **rdfs3** を用いて 2.2 節で述べたように領域と値域の制約に従って定義する. 唯一問題となるのが, 領域・値域の制約も得られず, rdf:type も rdfs:subClassOf もなしに前方参照されたときであり, このとき RDFS 伴意ルール **rdfs4a** と **rdfs4b** (プロパティではないすべての引用は rdfs:Resource のインスタンスであるという伴意) を用いることができるが, 実際にそれを rdfs:Class のインスタンス(すなわちクラス)とするべきなのか rdfs:Resource のインスタンス(すなわちインスタンス)とするべきなのかという問題がある(どちらの場合でも伴意は満足される). その場合はインスタンスとして仮に実現しておき, その後それがクラスと判明した時点で `change-class` することとした.

5. おわりに

RDFS の公理を実現し, 伴意ルールを実装した RDFS プロセッサを, リスプオブジェクトシステム CLOS のメタオブジェクト・プロトコルを用いて開発した. このシステムではすべての RDFS エンティティはクラスまで含めて CLOS オブジェクトであり, RDFS の公理実現にあたって CLOS のリフレクション機能が有効に用いられた.

謝辞

本報告は, 文科省ITプログラム「ITを活用した大規模システムの運用支援システムの構築」の一部として実施されたものである. 本プロジェクト実施では大須賀節雄東京大学名誉教授に技術評価委員長をお願いし, オントロジ構築に関して大阪大学溝口理一郎教授にご協力戴いている. 記して感謝の意を表する.

参考文献

[CLtL2] Steele Jr., G.L.: Common Lisp The Language second edition, Digital Press, (1990).

[Kiczales 1992] Kiczales, G., des Rivières, J., Bobrow, D.G.: The Art of the Metaobject Protocol, MIT Press, (1992).

[小出 02] 小出ほか:「ITを活用した大規模システムの運用支援システム」, SICE システムインテグレーション部門講演会, pp.399-400, (2002).

[小出 03] 小出・北村:MOP3:セマンティックウェブプロセッサ, 第 17 回人工知能全国大会, 1D4-03, (2003).

[Koide 03] Koide, S., et al.: "Operation-Support System for Large-Scale System Using Information Technology", Proceedings of 5th Int. Conf. Enterprise Information Systems (ICEIS2003), Vol.4, pp.430-437, ESEO, (2003).

[Lassila 2001] Lassila, O.: Enabling Semantic Web Programming by Integrating RDF and Common Lisp, Proceedings of the first Semantic Web Working Symposium (SWWS'01), pp.403-410, Stanford Univ., California (2001).

[Lassila 2002] Lassila, O.: Taking the RDF Model Theory Out for a Spin, ISWC2002, pp.307-317, (2002).

[Paepcke 1993] Paepke, A. (ed.): Object-Oriented Programming – The CLOS Perspective, MIT Press, (1993).

[Pan 2003] Pan, J.Z. and I. Horrocks: RDFS(FA) and RDF MT: Two Semantics for RDFS, ISWC2003, pp.30-40, (2003).

[Smith 84] Smith, B.: Reflection and Semantics in Lisp, Proc. 1984 ACM Principles of Programming Language Conference, ACM, (1984) 23-35.

[Wielemaker 2003] Wielmaker, J., Guus Schreiber, and B. Weilinga: Prolog-Based Infrastructure for RDF: Scalability and Performance, ISWC2003, pp.644-658, (2003).