

RV: Resource Shape に基づく RDF 文書の検証方法と評価

中島 啓貴† 成田 貴大† 脇田 宏威† 青山 幹雄†

†南山大学 情報理工学部 ソフトウェア工学科

1 背景

現在, RDF 文書の検証方法は確立されていない. 推論規則定義言語を型定義に用いた検証方法 [1][2] が提案されているが, これらの言語は開世界仮説や唯一名仮説を前提とするため, 型定義として用いることはできない. このような語彙の意味を無視した運用は, 語彙に対する複数の意味の混在という弊害を伴う.

このような現状に対し RDF 文書の制約定義言語として Resource Shape が提案されている.

2 関連研究

2.1 Resource Shape[3]

RDF モデルに対する文書型と要素型を RDF モデルとして定義する制約定義言語である. Resource Shape を型定義に用いる検証方法は提案の段階に留まる.

2.2 RDFUnit[1]

RDF モデルの品質評価のための SPARQL クエリで構成されたテストスイートを実行するフレームワークが提案されている.

2.3 Pellet Integrity Constraint Validator[2]

OWL 用いて RDF モデルに対する制約表現を行い, SPARQL クエリを生成することで検証を実現している.

3 研究課題

RDF は開世界仮説を前提としており, 閉世界仮説を前提とする一般的な文書検証の定義を適用できない. この前提の違いを考慮し, RDF 文書検証を定義する.

RDF 文書検証とは, RDF 文書とサービス記述を入力として, RDF 文書作成者の参照可能範囲で検証し, 検証結果を表す RDF グラフの URI を有限時間内に出力するものとする.

RDF 文書作成者の参照可能範囲は以下の RDF グラフを含む.

- (1) RDF 文書に含まれる RDF グラフ
- (2) サービス記述に含まれる RDF グラフ
- (3) (1) と (2) から再帰的に参照可能なりソースに含まれる RDF グラフ

検証の結果を表す RDF グラフは結果グラフと呼ぶこととする. 検証結果, RDF 文書の URI, 作成日時, 違反箇所を特定可能な診断情報が含まれる.

この定義に基づいた検証を実現する検証プロセスの提案と評価を研究課題とする.

RV: RDF Documents Validator Based on Resource Shape and its Evaluation

†Hiroki NAKASHIMA, Takahito NARITA, Kouki WAKITA, Mikio AOYAMA, Department of Software Engineering, Nanzan University

4 アプローチ

4.1 部分グラフ検索による制約充足性の形式的判断

制約充足の成否に影響する部分グラフを検索する SPARQL クエリを作成することで, 制約充足性の形式的判断を実現した.

4.2 検証の局所的中断による有限時間内での検証遂行

Resource Shape における制約定義の木構造に着目し, 検証の局所的中断を導入することで, 検証結果を有限時間内に決定可能にした.

5 提案方法

5.1 前提条件

RDF 文書は構文的に正しく, かつシェイプは構成的に正しいものと仮定する. また, RDF において URI は正しく利用され, 1つの RDF 文書は1つの連結グラフで構成されるものと仮定する. 検証プロセスでは1つの SPARQL エンドポイントを一貫して利用する.

5.2 検証結果の定義

検証結果のメタモデルを図1に示す.

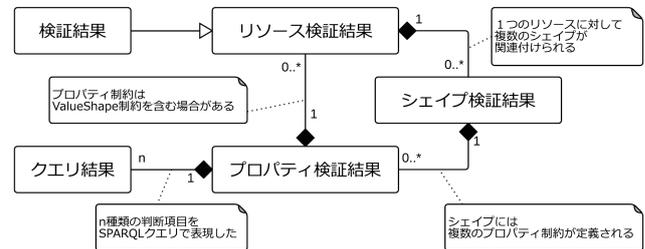


図1: 検証結果のメタモデル

5.3 検証プロセス

検証は図2に示すアクティビティ図に従って行う.

5.3.1 制約充足性の判断と診断情報の生成

検証では, 制約充足性判断と診断情報生成のために SPARQL クエリを用いる. 制約充足性の判断には, 制約充足に必要な部分グラフを検索する ASK クエリ (判定クエリ) と, 制約充足を妨げる部分グラフを検索する ASK クエリ (例外クエリ) の2種類を用いる. 診断情報の生成には, 制約充足を妨げる部分グラフから診断情報を抽出し, RDF グラフを生成する CONSTRUCT クエリ (診断クエリ) を用いる.

これらの SPARQL クエリは Resource Shape の仕様をもとにテンプレートとして作成したため, 検証過程でコンテキストに応じて URI をバインドして利用する.

5.3.2 循環の検出方法

検証では、ValueShape 制約が形成する循環定義を検出するためにコールスタックを用いる。形成される依存関係はリソースとシェイプの組み合わせによって決定されるため、スタックフレームはリソースとシェイプの組とし、図2に示すようにシェイプ検証結果取得の最初にウィンド、シェイプ検証結果取得の最後にアンウィンドを行う。

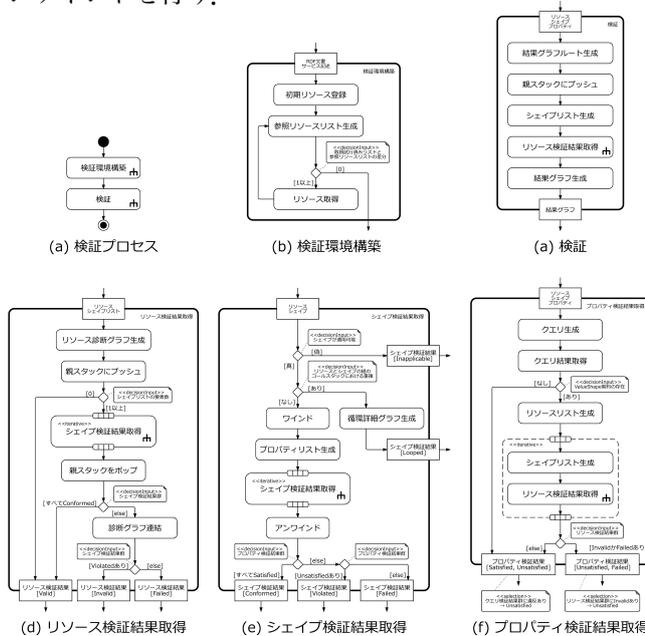


図2: 検証プロセスのアクティビティ図

6 適用

提案方法の妥当性を評価するため、例題を作成し提案方法を適用した。

ここでは、リスト1に挙げるRDF文書 <http://example.org/r1> を検証対象とし、検証範囲にはリスト1の他にリスト2とリスト3に挙げるRDF文書が含まれる。

シェイプs1は対象とするリソースの型を限定しないため、リソースr1に適用される。

r1のプロパティfooは"some long text"という値を持つが、s1によって値の長さを10文字以内に制約されるため、制約違反として図3の(1)に診断情報が提供される。

r1のプロパティbarはリソースr2という値を持つため、s1に従いシェイプs2がr2と関連付けられる。r2はT3型のリソースではないため、対象とするリソースの型を型T3に限定するs2は適用されない。r2に関連付けられたシェイプがすべて適用されなかったため、図3の(2)に適用不能な状態に関する診断情報が提供される。

制約違反が存在したため、検証結果は図3の(3)に示す通り無効(Invalid)となり、RDF文書検証の定義を満たす出力が得られた。

リスト1: リソース r1

```

1 @base <http://example.org/> .
2 @prefix osc: <http://open-services.net/ns/core#> .
3
4 <r1> a <T1> ;
5     osc:instanceShape <s1> ; # s1がr1と関連付けられる
6     <foo> "some long text" ;
7     <bar> <r2> .
8 <r2> a <T2> ;
9     <baz> "some text" .
    
```

リスト2: シェイプ s1

```

1 @base <http://example.org/> .
2 @prefix osc: <http://open-services.net/ns/core#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4
5 <s1> a osc:ResourceShape ;
6     osc:property <s1#foo>, <s1#bar> .
7 <s1#foo> a osc:Property ;
8     osc:name "foo" ;
9     osc:propertyDefinition <foo> ;
10    osc:occurs osc:Exactry-one ; # fooは1回だけ出現
11    osc:maxLength "10"^^xsd:integer . # fooの値は10文字以内
12 <s1#bar> a osc:Property ;
13    osc:name "bar" ;
14    osc:propertyDefinition <bar> ;
15    osc:occurs osc:One-or-many ; # barは1回以上出現
16    osc:valueShape <s2> . # barの値はs2と関連付けられる
    
```

リスト3: シェイプ s2

```

1 @base <http://example.org/> .
2 @prefix osc: <http://open-services.net/ns/core#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4
5 <s2> a osc:ResourceShape ;
6     osc:describes <T3> ; # s2の制約対象はT3型のリソース
7     osc:property <s2#baz> .
8 <s2#baz> a osc:Property ;
9     osc:name "baz" ;
10    osc:propertyDefinition <baz> ;
11    osc:occurs osc:Exactry-one ; # bazは1回だけ出現
12    osc:maxLength "10"^^xsd:integer . # bazの値は10文字以内
    
```

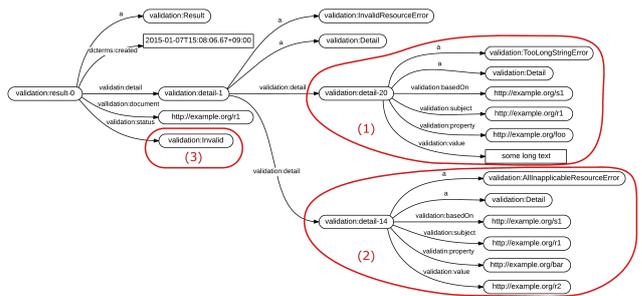


図3: 適用例から得られた結果グラフ

7 評価

提案方法により制約定義言語 Resource Shape を用いたRDF文書検証が可能となった。既存のアプローチ[1][2]では不可能だった意味の混在を伴わないRDF文書検証が可能となった。

8 まとめ

SPARQLを利用してResource Shapeを用いたRDF文書検証を実現し、適用例によって検証方法の妥当性を評価した。提案方法は、Linked Open Data[4]などの検証への活用が期待できる。

参考文献

[1] S. A. R. Cornelissen et al. RDFUnit, 2014. <http://aksw.org/Projects/RDFUnit.html>.
 [2] H. Prez-Urbina et al. Validating RDF with OWL Integrity Constraints, 2008. <http://docs.stardog.com/icv/icv-specification.html>.
 [3] A. G. Ryman. Resource Shape 2.0, 2014. <http://www.w3.org/Submission/2014/SUBM-shapes-20140211/>.
 [4] W3C. Linking Open Data. <http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>.