Vol.2012-SE-175 No.5
2012/3/15

モデル検査器 NuSMV を用いたオントロジーの検証

阿部雄貴[†1]　和泉諭[†2]
小林秀幸[†1]　高橋薫[†1]

本研究では，ソフトウェアや組込みシステムの検証において有効であるモデル検査法をオントロジーの検証に応用する．具体的には，モデル検査ツールで扱うためのオントロジーの検証モデルを与え，この検証モデルに基づいて，オントロジーの構文的な矛盾や意味的な矛盾について，モデル検査ツールを用いて検証する．本稿では，モデル検査ツールとして NuSMV を使用し，そのための検証モデルと検証例について述べる．

Verification of ontology using the model checker NuSMV

Yuki Abe ,[†1] Satoru Izumi ,[†2] Hideyuki Kobayashi [†1]
and Kaoru Takahashi [†1]

We apply the model checking technique for consistency checking of various systems such as software and embedded systems to the verification of ontology. Specifically, a verification model of the ontology is given. Then, we check its syntactic and semantic consistency using a model checker. In this paper, we use NuSMV as the model checker. We show the ontology verification model for NuSMV and verification examples.

†1 仙台高等専門学校
Sendai National College of Technology
†2 東北大学電気通信研究所/情報科学研究科
Research Institute of Electrical Communication / Graduate School of Information Sciences, Tohoku University

1. はじめに

Semantic Web を始めとする様々な取り組みにオントロジーの利用が行われている．オントロジーは概念化の明示的な記述であり，知識処理システムが対象とする世界のオントロジーを構築することで，対象となる概念の合意による知識の共有／再利用が可能である．しかし，オントロジーは通常，人手によって設計し記述されるため，その中に矛盾や一貫性が保たれていない記述や誤った記述が存在する可能性がある．そこで，記述したオントロジーをその正しさについて検証することが望ましい．

既存の研究として，オントロジーエディタ Protégé[1] や推論機構 Racer[2] を用いたツールによる検証が挙げられる．文献[3] では，家族オントロジーを例とし，上記ツールを用いたオントロジーの包含関係の整合性の検証，推論を用いた推論ルールでの家族関係の補完を行っている．これらは包含関係を中心としたオントロジーの一般的な整合性，無矛盾性を記述論理に基づいて検証する方法である．しかし，各ドメインに対してどのような性質，条件を満たしているか，単純な作成ミスはないのか，一貫性が保たれているか等，個々に対しての問題を検証することは難しい．文献[4] では，オントロジーを形式化したグラフで表すことで，素と補集合の関係に着目した構文に関するオントロジーの矛盾の検証を行っている．また，代表的な構文的矛盾として 6 つのパターンを挙げ，それぞれを検証可能としている．文献[5] では，PVS（Prototype Verification System）を用いて，OWL（Web Ontology Language）[8] と ORL（OWL Rule Language）の検証を行っている．この手法は ORL まで推論できるのが大きな特徴であり，構文の矛盾や包含関係，インスタンスの検証を行っている．文献[6] では，オントロジー内の包含関係に着目し，矛盾の発見，修復などを行っており、自動的に修復すべき点を探し出すアルゴリズムを持っているのが特徴である．他にも，オントロジーの作成支援，検証を行うツール[9] や，クラスの矛盾をクラスの演算を用いて数量的に表す手法[10]，オントロジーを用いてモデル変換を行う研究[11] などが行われている．

一方，システムの検証方法の 1 つとしてモデル検査[12] がある．モデル検査は，システムの動作を有限状態モデルで表現し，その状態空間を網羅的に探索することによって，与えられた性質が満たされるか否かを判定する検証方法である．与えられた性質に対し，真偽値でその性質を満たしているか否かを判定し，偽であった場合に反例（counter example）を出力する．この反例を分析することにより，検証の失敗がどのように起きたのかを調べることができる．モデル検査をツール化し，その利用や利用の応用といった研究が行われている[13][14][15]．

本稿では，ソフトウェアや組込みシステムの検証において有効であるモデル検査法をオントロジーの検証に応用する．モデル検査ツールとしては NuSMV[7] を用いる．NuSMV でオントロジーの検証を行うためには，オントロジーを NuSMV の検証モデルへ変換する必要があり，オントロジーの性質を損なわないように変換しなければならない．ここでは，この変換した検証モデルを状態遷移モデルと呼ぶ．この状態遷移モデルで検証可能とする検証項目は，文献[4] で取り上げられている代表的な構文的矛盾の他，構文上では矛盾は存在しないが，意味として誤りがあるような意味的矛盾（例えば，兄弟の兄弟が親として記述されているような誤り）である．NuSMV を利用することで，検証項目が満たされていないときに反例が出力され，それを基に誤り容易に調べることができ，オントロジーの誤りを修正することができる．

以下，2 節ではオントロジーの矛盾について説明し，3 節でオントロジーを NuSMV で取り扱うための状態遷移モデルを与える．4 節では，状態遷移モデルに基づいた矛盾検証のための手法を示し，その検証例を 5 節で述べる．最後に，6 節でまとめと課題を述べる．

2. オントロジーの矛盾

本節では，本研究で取り扱うオントロジーの矛盾について説明する．

オントロジーを作成する際に，クラス間の補集合関係や互いに素である関係により生じる構文的な誤りにより，オントロジーに矛盾が生じてしまうことが考えられる．クラス間の補集合関係や互いに素である関係において代表的な構文的矛盾には以下の 6 つのパターン (図 1) が考えられる[4]．これらの構文的矛盾はオントロジーエディタによってエラーとして検出されず，オントロジーの作成後に検証しなければならない．

[補集合や互いに素の関係で起こりうる構文的矛盾]

- **(a)** 互いに素なスーパークラス
- **(b)** スーパークラスと自身が互いに素
- **(c)** 互いに素なクラスへのプロパティ値の全称制約と存在制約の混在
- **(d)** 互いに素なクラスの積へのプロパティ値の存在制約
- **(e)** 自身とは素な定義域を持つプロパティ値の存在制約の使用
- **(f)** 存在制約を持つ、プロパティ値の値域とは素なクラスの使用

一方，オントロジーの意味的矛盾とは，構文的には合っているが意味として矛盾しているものを指す．例えば，家族を対象としたオントロジーにおいて，兄弟のプロパティで推移的に結ばれた関係でありながら，親として記述されているといったものである．

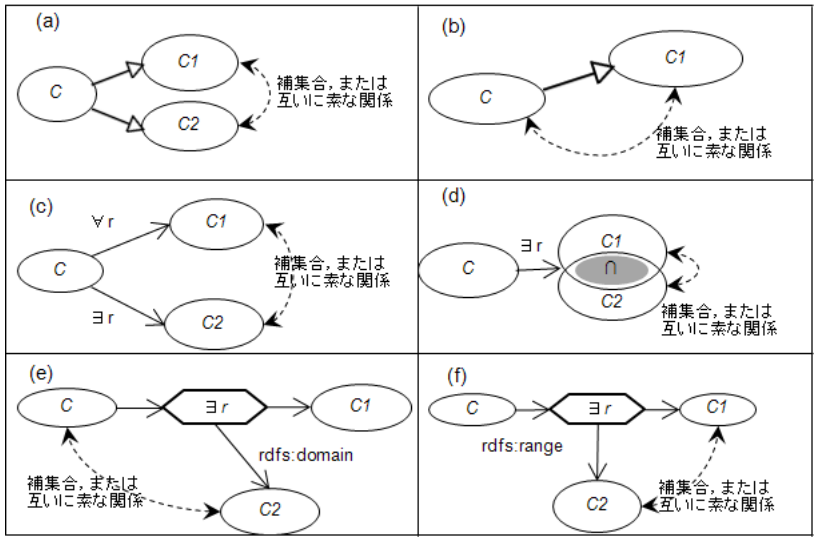


図 1 補集合や互いに素の関係で起こりうる構文的矛盾の代表的な 6 つのパターン

このような矛盾は，下位クラスに対する厳密な制約の記述がされていない場合などに起こりうる．図 2 に意味的矛盾が記述されているオントロジーの例を示す．クラス Human のサブクラスとして Man，Woman の 2 つのクラスが定義されており，インスタンスとして Bob と Alice がそれぞれ存在し，クラス Human の上にはプロパティ hasSister の関係が定義されている．このとき，クラス Man のインスタンス Bob が，クラス Woman のインスタンス Alice のプロパティ hasSister として記述されている．これは，クラス Man のインスタンスが「姉妹である」と記述されているので矛盾している．しかし，Human の定義は正しく，構文上の誤りとして見つけることが困難であり，OWL の全称制約（owl:allValuesFrom）の未記述などで起こり得ると考えられる．

3. オントロジーの状態遷移モデル

本節では，オントロジーを NuSMV で扱うための検証モデル（以下，状態遷移モデルと言う）について述べる．

オントロジーはトリプル〈S, P, O〉の集合であり，ラベル付き有向グラフとして表され

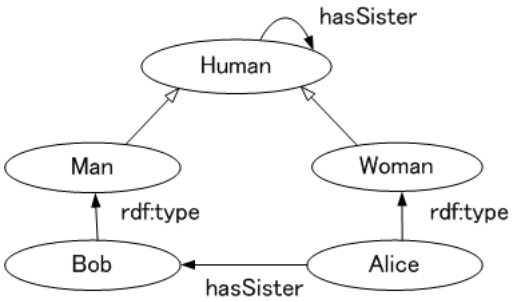


図 2 意味的矛盾を含むオントロジーの例

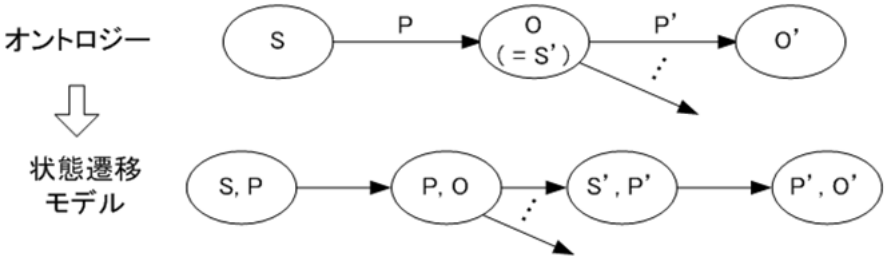


図 3 オントロジーと状態遷移モデルの対応

る．一方，NuSMV で扱うモデルはクリプキ構造によるラベル無し有向グラフである．オントロジーを単純な対応付けにより，NuSMV の検証モデルへと変換するとオントロジーが持つ関連性が崩れてしまう．また，NuSMV は検証時に検証内容が満たされていない場合に反例を出力する．この反例を用い，誤りに対する修正を行うことができるため，オントロジーを状態遷移モデルへ変換する際には，オントロジーが持つ関連性と反例の出力を考慮し，変換を行う必要がある．

状態遷移モデルでは，オントロジーのトリプル〈S, P, O〉を，主語を表す (S, P) と目的語を表す (P, O) の 2 つの状態で表現する（図 3）．また，クラスとインスタンスをそれぞれ別の変数として同時に記述する．クラスのみを表す場合はインスタンス変数の値は empty とし，クラスとインスタンスが同時に記述されている場合はそのクラスに属するインスタンスを表す．クラスとインスタンスを別々の変数で表すことで，クラスレベルのみの検証や，クラスとインスタンスの関連性に対するきめ細かな検証を行うことができる．さらに，オントロジーの制約条件やクラス演算等を表現するために，状態遷移モデルでは，それぞれに関する変数 Restriction と ClassOperator を加える．

素と補集合の関係で起こり得る矛盾において，検証後の誤りを指摘するための変数を追加する．具体的には，OWL のプロパティ owl:disjointWith と owl:complementOf に対し，互いに素である（または補集合）関係を示すブール変数（シンボル）を付加し、さらにプロパティ変数として状態遷移モデルに記述する．各々をシンボルとプロパティ変数として状態遷移モデルに記述することにより，各クラスについての互いに素（または補集合）である情報を正確に記すことが可能となり，検証の際に誤りを指摘できる．

次に，包含関係（サブクラス関係）を示すプロパティに対しての変換を行う．rdfs:subClassOf については，本来，下位クラスから上位クラスへの関係として記述されるが，状態遷移モデルで扱う際は，元のオントロジーにおいて上位クラスから下位クラスへの関係 hasSubClass へと変換する．この変換により，オントロジーは，owl:Thing を最上位クラスとし他の全てのクラスを owl:Thing の下位クラスとして構成することで，owl:Thing を頂点としたツリー構造となり，全状態を網羅した検証が可能となる．同様に rdf:type も hasType へ変換して扱うことでインスタンスに対しても探索を行う．

以上の要件を考慮して，状態遷移モデルの状態 S を以下のように表現する：

$$S = \langle c,\ i,\ p,\ so,\ res,\ cop,\ dc \rangle$$

ここで，状態要素としての各変数は，c がクラス，i がインスタンス，p がプロパティ，so が主語か目的語，res が制約，cop がクラス演算子，dc が素か補集合をそれぞれ表している．

4. 状態遷移モデルによる検証

本節では，前節で与えた状態遷移モデルを用いて，2 節で示した構文的矛盾 (a)∼(f) に対しての検証方法について述べる．

それぞれの構文的矛盾を含んだオントロジーの状態遷移モデルとその検証式を示す．検証式は NuSMV の CTL（Computation Tree Logic）及び LTL（Linear Temporal Logic）時相論理式で表す．なお，以下で与える状態遷移モデルでは，最上位クラス owl:Thing のサブクラスから遷移した後からの状態遷移（以下の各図の点線矢印）を示し，owl:Thing についての記述は省略している．C, C1, C2 はクラス，r はプロパティ，S と O はそれぞれ主語，目的語，D1∼D6 は互いに素である関係を示している．ここでは構文的矛盾においてクラス定義による矛盾の検証を行うため，インスタンスは示さずに状態要素 i=empty としている．なお，インスタンスについても検証可能である．

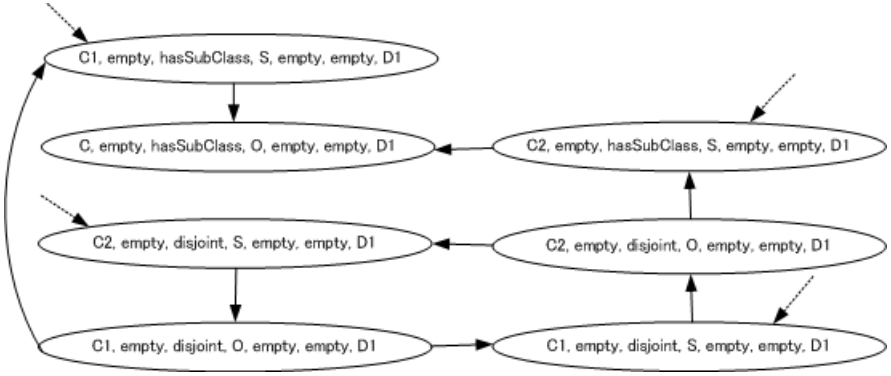


図 4 （a）の矛盾を検証するための状態遷移モデル

4.1 構文的矛盾（a）の検証

構文的矛盾（a）では，クラス C が互いに素であるスーパークラス C1 と C2 のサブクラスとして記述されているため，クラス C は空集合となり，個体が存在しえない定義となっている．図 4 にこの矛盾を検証するための状態遷移モデルを示す．検証式は以下である．

```
G !(((so = o & p = hasSubClass) & c = C)
      & Y(((p = hasSubClass & so = s) & D1) & c != C1))
```

この検証式では，C のスーパークラス同士が互いに素でないかどうかを検証している．

この検証式が真となれば矛盾が存在しないことを示す．偽であれば矛盾が存在し，反例が出力される．反例では，C のスーパークラス同士が互いに素であることを示す状態遷移が出力される．反例を解析する際は，C へ hasSubClass で遷移するスーパークラス C1 に着目し，その C1 と互いに素または補集合関係（D1）にあるクラスのサブクラスに C があるかどうかを見ることで確認することができる．

4.2 構文的矛盾（b）の検証

構文的矛盾（b）では，クラス C がプロパティ hasSubClass でクラス C1 と結ばれており，かつ，C と C1 が互いに素（D2）であるため，包含関係が成り立たない矛盾が生じている．図 5 にこの矛盾を検証するための状態遷移モデルを示す．検証式は以下の通りである．

```
!G(((so = o & p = hasSubClass) & D2)
```

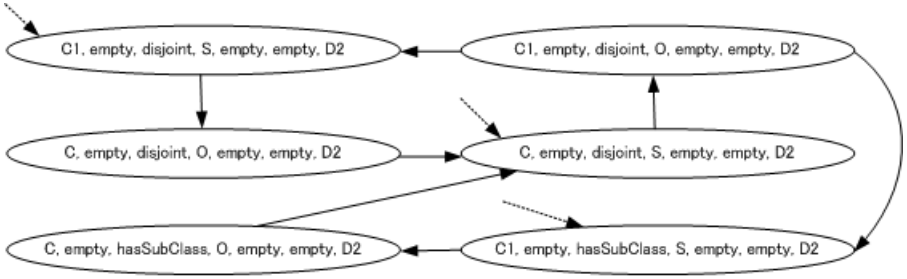


図 5 （b）の矛盾を検証するための状態遷移モデル

```
      & Y((p = hasSubClass & so = s) & D2))
```

この検証式ではクラスとそのスーパークラスが互いに素である関係（D2）で結ばれていないことを検証している。この式が真となれば矛盾が存在しないことを示し，偽であれば，反例が出力される．出力される反例では，プロパティ hasSubClass で結ばれたクラスとそのスーパークラスが互いに素または補集合関係（D2）を示す状態遷移に着目し，確認することで修正を図ることができる．

4.3 構文的矛盾（c）の検証

構文的矛盾（c）は，互いに素または補集合関係にあるクラスへのプロパティ値に全称制約と存在制約（owl:someValuesFrom）が混在している．全称制約によってプロパティ値とされたクラスと存在制約によってプロパティ値にされたクラスが互いに素または補集合関係にあるため，存在制約によるプロパティ値は存在してはならず，矛盾である．図 6 にこの矛盾を検証するための状態遷移モデルを示す．検証式は以下である．

(1)
```
AG(((c = C & p = r) & so = s)
      -> AX(((res = allValuesFrom & p = r) & so = o) & c = C1))
```
(2)
```
EF((c=C1 & p=disjoint & so=s) & EX(c=C2 & so=o & p=disjoint))
```

(1) 式により，クラス C のプロパティ r の値は常にクラス C1 であることを検証する．この検証が偽であり，反例が出力された場合，C1 とは別のクラス C2 がプロパティ値として記述されていることが確認できる．次に (2) 式で，その C2 が C1 と互いに素または補集合関係にあるか検証を行う．この検証式が真となれば，C1 と C2 が互いに素であることが確

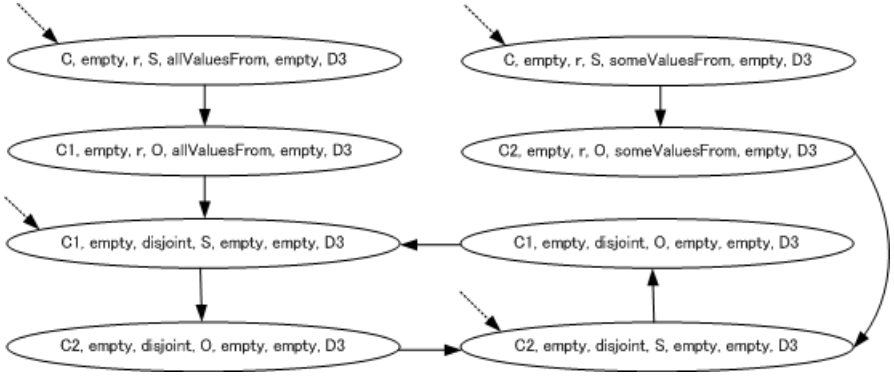


図 6 （c）の矛盾を検証するための状態遷移モデル

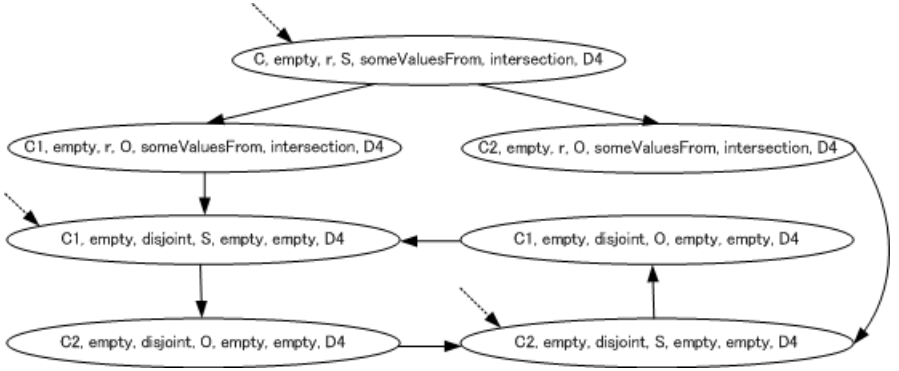


図 7 （d）の矛盾を検証するための状態遷移モデル

認でき，これを基に修正を行える．

4.4 構文的矛盾（d）の検証

構文的矛盾（d）は，互いに素または補集合関係にあるクラスの積（owl:intersectionOf）へのプロパティ値に存在制約を課している．互いに素または補集合関係のクラスの積演算であるため，空集合のクラスへプロパティ値が存在すると記述しており矛盾を生じる．図 7 にこの矛盾を検証するための状態遷移モデルを示す．この検証を行う式は以下の 3 つであり，(1) から順に実行する．

(1) `AG((so = s & cop=intersection)->(AX(!D4)))`

(2)
```
AG(so = s & c = C & p = r & cop = intersection
                          -> AX(D4 & EX EG p=Disjoint))
```

(3) `AG!((so = s & c = C & p = r & cop = intersection) & EX EX EG (D4))`

まず (1) 式により，クラス演算変数 cop で intersection で結ばれたものの中で，互いに素である関係があるか検証を行う．ここで偽となれば互いに素である関係のクラスが存在していることがわかり，そのクラスが 1 つ出力される．次に，(2) 式により，そのクラスについて，どのクラスと互いに素で結ばれているか確認を行う．(2) 式により真となれば互いに素で結ばれており，かつ積演算がされていることがわかる．(3) 式により反例を出力させ，修正することができる．(2) 式により互いに素であるクラスが存在していることが確認されているため，(3) 式によって出力される反例では，クラス C1 と C2 が互いに素であり，かつクラス C からその積クラスへ存在制約を課しているプロパティ r によって状態遷移されていることが解析できる．これを用いてオントロジーの修正を行うことが可能である．

4.5 構文的矛盾（e）の検証

構文的矛盾（e）では，存在制約の課されたプロパティの定義域のクラスと互いに素または補集合関係にあるクラスが主語として記述されているため矛盾である．図 8 にこの矛盾を検証するための状態遷移モデルを示す．この検証を行うには以下の検証式を用いる．

(1)
```
!EF((((so = s & c != C2) & p = r ) & res = someValuesFrom)
                & EX((so = o & c = C1) & p = r))
```

(2)
```
!EF(so=s & c=C2 & p=disjoint & D5
                & EX(so=o & c=C & p=disjoint & D5))
```

(1) 式により，存在制約が掛かったプロパティr で結ばれる主語のクラス C2 以外のものが存在しないか検証する．この検証式が偽となった場合，定義域以外で主語となっているクラス C が確認できる．(2) 式により，クラス C2 がクラス C とは互いに素または補集合関係（D5）にないか検証を行う．(2) 式が偽となったとき，反例が出力され，定義域であるクラス C2 と互いに素であるクラス C が反例として出力されることを確認することでこの矛盾に関する修正を行うことができる．

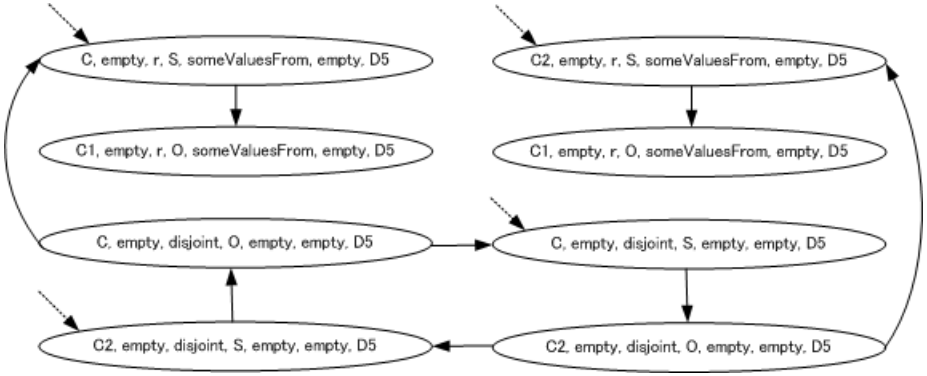


図 8 （e）の矛盾を検証するための状態遷移モデル

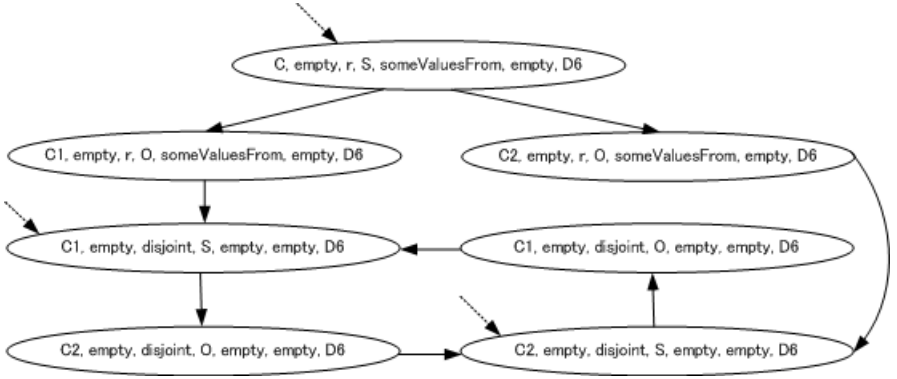


図 9 （f）の矛盾を検証するための状態遷移モデル

4.6 構文的矛盾（f）の検証

構文的矛盾（f）では，存在制約の課されたプロパティの値域のクラスと互いに素または補集合関係にあるクラスが目的語として記述されており矛盾である．図 9 にこの矛盾を検証するための状態遷移モデルを示す．検証式は以下である．

(1) `!EF((((so = s & c = C) & p = r) & res = someValuesFrom)`
`& EX((so = o & c != C2) & p = r))`

(2) `!EF(so=s & c=C2 & p=disjoint & D6`
`& EX(so=o & c=C1 & p=disjoint & D6))`

(1) 式により，存在制約が掛かったプロパティ r で結ばれる目的語のクラス C2 以外のものが存在しないか検証する．この検証式が偽となった場合，値域以外で目的語となっているクラス C1 が確認できる．(2) 式により，クラス C2 がクラス C1 とは互いに素または補集合関係（D6）にないか検証を行う．(2) 式が偽となったとき，反例が出力され，値域であるクラス C2 と互いに素であるクラス C1 が反例として出力されることを確認することでこの矛盾に関する修正を行うことができる．

5. 検 証 例

本節では，モデル検査ツール NuSMV を用いて，具体的な構文的矛盾と意味的矛盾の検証例について述べる．NuSMV を用いた検証では，まず検証を行うオントロジーを 3 節で示した状態遷移モデルに変換し，NuSMV へ SMV コードとして入力する．そして，4 節で示したような検証したい内容を時相論理式として与え，矛盾がないか確認する．矛盾が存在する場合，NuSMV はその矛盾を反例として出力する．

5.1 構文的矛盾の検証例

構文的矛盾の検証例として，文献[4] で述べられている例（図 10）を用いる．これは 2 節で説明した矛盾 (d) に対応する．このオントロジーは,「Cow かつ Sheep であるものを Lion が食べるという事実が少なくとも 1 つ存在する」ことを示している．しかしながら，Cow と Sheep は互いに素であるため，Cow かつ Sheep なるものは存在しない．つまり，空であるものを存在制約してしまう，構文的矛盾である．

このオントロジーから生成した状態遷移モデルを図 11 に示す．この状態遷移モデルを NuSMV に入力し，検証内容を検証式として，CTL 時相論理式で記述し，モデル検査を実行する．検証内容が満たされていない場合に反例が出力され，この反例の解析を行うことでオントロジーの修正を図る．

この矛盾に対し，検証を行う時相論理式は以下の 3 つであり，(1) から順に実行する．

(1) `AG((so=s & cop=intersection) -> (AX(!D)))`

(2) `AG(so=s & c=Lion & p=eat & cop=intersection`
`-> AX(D & EX EG p=disjoint))`

(3) `AG !((so=s & c=Lion & p=eat & cop=intersection) & EX EX EG (D))`

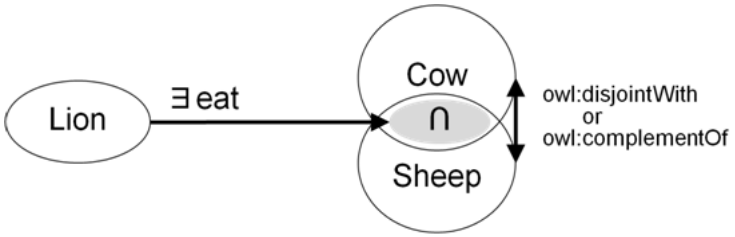


図 10 構文的矛盾 (d) を含んだオントロジー

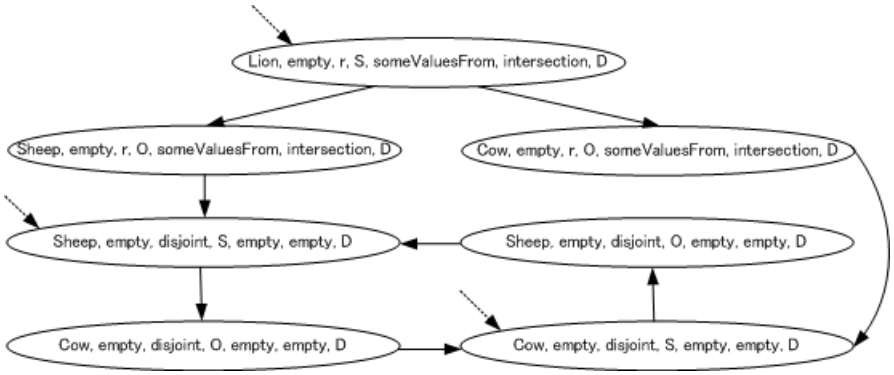


図 11 図 10 のオントロジーを変換して得られた状態遷移モデル

まず (1) 式により，クラス演算 intersectionOf で結ばれたものの中で，互いに素である関係（D）があるか検証を行う．このとき，偽となれば互いに素である関係のクラスが積演算されたことがわかり，そのクラスの 1 つ，Sheep が反例により出力される．次に，(2) 式により，Sheep のクラスについて，互いに素であり，積演算の対象となるクラスが存在するかを確認する．ここで真となれば互いに素である関係で積演算の対象となっているクラスの存在を確認でき，(3) 式により Sheep と Cow のクラスが互いに素であることを示す反例（図 12）を出力させ，修正することができる．

5.2 意味的矛盾の検証例

本研究で扱う意味的矛盾とは，クラス定義は正しいが，制約などにより意味として矛盾してしまうもののことを指す．このような意味的矛盾の検証はオントロジーが意図したものとなっているか，各ドメインに応じて行う必要がある．

Loop	Step	C1	D	D2	D4	D41	D5	D6	c	cop	i	p	res	so
	0	0	0	0	0	0	0	0	Thing	empty	empty	hasSubClass	empty	s
	1	1	1	1	0	0	0	1	Camivore	empty	empty	hasSubClass	empty	o
	2	1	1	1	0	0	0	1	Camivore	empty	empty	hasSubClass	empty	s
	3	0	1	1	0	0	0	0	Lion	empty	empty	hasSubClass	empty	o
	4	0	1	1	0	0	0	0	Lion	intersection	empty	eat	some	s
	5	0	1	0	1	0	0	0	Sheep	intersection	empty	eat	some	o
↱	6	0	1	0	1	0	0	0	Sheep	empty	empty	Disjoint	empty	s
|	7	0	1	0	1	0	0	0	Cow	empty	empty	Disjoint	empty	o
|	8	0	1	0	1	0	0	0	Cow	empty	empty	Disjoint	empty	s
|	9	0	1	0	1	0	0	0	Sheep	empty	empty	Disjoint	empty	o
└	10	0	1	0	1	0	0	0	Sheep	empty	empty	Disjoint	empty	s

図 12 構文的矛盾 (d) の検証結果（反例）の出力

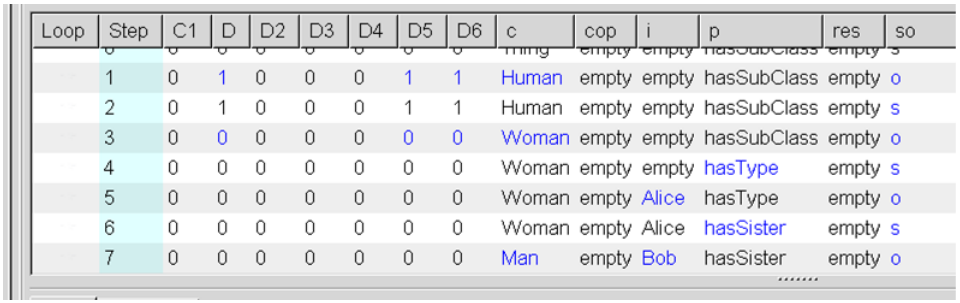
例として，図 2 のオントロジーの検証を行う．クラス Man のインスタンス Bob が「Alice の姉妹である」と記述されているため，意味の矛盾が生じている．しかし，クラス Human の定義によれば，構文上の誤りとして見つけることが困難であり，全称制約を記述していないなどで起こり得ると考えられる．

このような意味的矛盾は以下の検証式で検証可能である．これは「男のクラスのインスタンスであり，かつ，姉妹として記述されているインスタンスとして存在するか」を検証している．ここで，i != empty はインスタンスが存在していることを示している．NuSMV による検証結果を図 13 に示す．

```
AG !(so=s & i!=empty & EX(so=o & c=Man & p=hasSister))
```

この結果として図 13 の反例が得られる．この反例では，クラス Man のインスタンス Bob が，クラス Woman のインスタンス Alice の姉妹の目的語として記述されていることがわかる．

このように，クラス定義のミスや制約の記述もれによって生じるインスタンスでの矛盾に対して検証可能である．



Loop	Step	C1	D	D2	D3	D4	D5	D6	c	cop	i	p	res	so
	1	0	1	0	0	0	1	1	Human	empty	empty	hasSubClass	empty	o
	2	0	1	0	0	0	1	1	Human	empty	empty	hasSubClass	empty	s
	3	0	0	0	0	0	0	0	Woman	empty	empty	hasSubClass	empty	o
	4	0	0	0	0	0	0	0	Woman	empty	empty	hasType	empty	s
	5	0	0	0	0	0	0	0	Woman	empty	Alice	hasType	empty	o
	6	0	0	0	0	0	0	0	Woman	empty	Alice	hasSister	empty	s
	7	0	0	0	0	0	0	0	Man	empty	Bob	hasSister	empty	o

図 13 意味的矛盾の検証結果の出力

6. おわりに

本稿では，モデル検査ツール NuSMV を用いて，オントロジーの代表的な構文的矛盾と各ドメインに応じた意味的矛盾の検証を行うための状態遷移モデルを与え，さらに具体的な検証方法について述べた．状態遷移モデルを扱うことで，制約やクラス演算を含んだオントロジーに対してそれらの検証を行うことができた．また，モデル検査ツールを用いたことにより，出力される反例を利用し，矛盾の発見と修正の手掛かりとすることができた．

今後は，状態遷移モデルに従い，検証対象のオントロジーを RDF/XML 形式から NuSMV へ入力する検証支援ツールを実装する予定である．

参考文献

1) “Protégé”, http://protege.stanford.edu/.
2) “Racer Systems GmbH & Co. KG”, http://www.racer-systems.com/.
3) C. Golbreich, “Combining Rule and Ontology Reasoners for Semantic Web”, LNCS 3323, pp.6-22, 2004.
4) S. C. Lam, D. Sleeman and W. Vasconcelos, “Graph-Based Ontology Checking”, Proc. KCAP05 Workshop on Ontology Management: Searching, Selection, Ranking and Segmentation, pp.33-40, 2005.
5) J. S. Dong, Y. Feng and Y. F. Li, “Verifying OWL and ORL Ontologies in PVS”, Z. Liu and K. Araki (Eds.): ICTAC 2004, LNCS 3407, pp.265-279, 2005.
6) P. Lambrix, Q. Liu and H. Tan, “Repairing the Missing is-a Structure of Ontologies”, Proc. the 4th Asian Semantic Web Conference - ASWC09, LNCS 5926, pp.76-90, 2009.
7) A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, “NuSMV: A New Symbolic Model Verifier”, LNCS 1633, pp.495-499, 1999.
8) “OWL Web Ontology Language Overview”, http://www.w3.org/TR/owl-features/.
9) A. B. Benevides and G. Guizzardi, “A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML”, J. Filipe and J. Cordeiro (Eds.): ICEIS 2009, LNBIP 24, pp.528-538, 2009.
10) A. Segev and A. Gal, “Ontology Verification Using Contexts”, Proc. the 2nd Workshop on Contexts and Ontologies: Theory, Practice and Applications, 2006.
11) S. Roser and B. Bauer, “Ontology-based Model Transformation”, Proc. Satellite Events at the MoDELS 2005 Conference, LNCS, pp.355-356, 2005.
12) 田中譲 (監修), “ソフトウェア科学基礎”, 近代科学社, 2008.
13) 門野雅弥, 土屋達弘, 菊野亨, “モデル検査器 NuSMV を利用したテストケース自動生成”, 情報処理学会研究報告, 2008-SLDM-134/2008-EMB-8, 2008.
14) E. D. Sciascio,F. M. Donini, M. Mongiello and G. Piscitelli, “Web Applications Design and Maintenance using Symbolic Model Checking”, Proc. the 7th European Conference on Software Maintenance and Reengineering (CSMR’03), pp.63-72, 2003.
15) K. Homma, S. Izumi, K. Takahashi and A. Togashi, “Modeling, Verification and Testing of Web Applications Using Model Checker”, IEICE Trans. Information and Systems, Vol.E94-D, No.5, pp.989-999, 2011.