UML のクラス図における論理プログラミングを用いた無矛盾検査について

Consistency Check for UML Class Diagram Using Logic Programming

佐藤　健[†]　　兼岩　憲[†]
Ken Satoh　　Ken Kaneiwa
[†] 国立情報学研究所 および 総合研究大学院大学
National Institute of Informatics and Sokendai
ksatoh@nii.ac.jp　　kaneiwa@nii.ac.jp

UML のクラス図において無矛盾性チェックをするため、クラス図の各コンポーネントを論理プログラミングのルールとして表し、その論理プログラミング上で矛盾が導かれるかどうかをチェックし、さらに、その矛盾を導くルールの集合を求める手法を提案する。

1 はじめに

UML [6, 4] は、ソフトウェアシステムを設計するときに使われる標準的なモデル表現言語である。UML では静的な概念をモデル化するためにクラス図を用いる。[2, 1, 3] において, UML クラス図と記述論理の関係が述べられており、UML クラス図の矛盾が記述論理の矛盾に対応していることが述べられている。本研究ではこれらのアプローチとは異なり UML クラス図を論理プログラミングへ変換し、その上での無矛盾性チェックを行うことを試みる。これは、論理プログラミングで直接実行することで通常の定理証明システムなどを使うよりも高速処理が図られると考えたからである。本稿ではその第一歩として、クラス属性、多重度、汎化関係、排他関係 (disjoint) のみを許す UML クラス図において, まず最初に [5] にしたがって UML クラス図に数量子 (counting quantifier) 付き一階述語論理式による意味を与え、さらに、それを論理プログラミングへ変換する手法を示し、その上での矛盾検出について述べる。

2 クラス図の数量子付き一階述語論理式による意味論

ここでは [5] にしたがって UML クラス図の意味を数量子付き一階述語論理式によって与える。

- 属性 $a[i..j]{:}t$ を持ったクラス c は、以下の数量子付き一階述語論理式で意味を与える。ここで t はクラスであり、$[i..j]$ は多重度を表す。

$$\forall x \forall y(c(x) \rightarrow (a(x,y) \rightarrow t(y)))$$

$$\forall x(c(x) \rightarrow \exists_{\geq i} z(a(x,z)))$$
$$\forall x(c(x) \rightarrow \exists_{\leq j} z(a(x,z)))$$

ここで $\exists_{\geq i} z$, $\exists_{\leq j} z$ は数量子付き変数であり、それぞれ、z の条件を満たす要素が i 以上、j 以下あることを表す。

- クラス階層に関しては、以下で意味を与える。クラス $c_1, ..., c_n$ の一般化が c であれば、

$$\forall x(c_1(x) \rightarrow c(x)), \ldots, \forall x(c_n(x) \rightarrow c(x))$$

- クラス $c_1, \ldots, c_n$ が排他であるときは、以下のように意味を与える。

$$\forall x(c_i(x) \rightarrow \neg c_{i+1}(x) \wedge \cdots \wedge \neg c_n(x))$$
$$\text{for } i \in \{1, \ldots, n-1\}$$

クラス図での矛盾は上記の論理式およびすべてのクラス c に対してインスタンスが存在するという論理式 $\exists x(c(x))$ を加えた理論の上での矛盾として定義される。

3 数量子付き一階述語論理式から論理プログラミングへの変換

属性 $a[i..j]{:}t$ を持ったクラス c に関する一階述語論理式から以下の論理プログラミングへ変換する。

```
t(f_a(X)) :- c(X).
(num(f_a(X)) >= i) :- c(X). (ただし i > 0)
(num(f_a(X)) =< j) :- c(X).
```

ここで f_a は属性 a に対するスコーレム関数であり、クラス C の要素 X からその属性 a の要素への関数を表す。さらに num は、その属性の要素の数を表すメタ関数である。

これは以下のように上の一階述語論理式をホーン節に変換し、部分評価を行うことで得られる。すなわち、

$$\forall x \forall y(c(x) \rightarrow (a(x,y) \rightarrow t(y)))$$
$$\forall x(c(x) \rightarrow \exists_{\geq i} z(a(x,z)))$$
$$\forall x(c(x) \rightarrow \exists_{\leq j} z(a(x,z)))$$

を上の f_a および num に関してホーン節に変換すれば、

```
t(Y) :- c(X),a(X,Y).
a(X,f_a(X)) :- c(X).
(num(f_a(X)) >= i) :- c(X).
(num(f_a(X)) =< j) :- c(X).
```

が得られる。ここで、第 1 式の a(X,Y) を第 2 式を用いて部分評価すると t(f_a(X)):-c(X),c(X) を得る。これが t(f_a(X)):-c(X) と等価であるから変換された論理プログラムの第 1 式が得られる。また、他に

$$\forall x(d(x) \rightarrow \exists_{\geq i} z(a(x,z)))$$

のような式があったとしても、それをホーン節に変換した式は、a(X,f_a(X)) :- d(X). となり、これを上の t(Y) :- c(X),a(X,Y). の部分評価に使うと t(f_a(X)):-c(X),d(X) となり、t(f_a(X)):-c(X) に包摂される。したがって他に a に関する他のホーン節があってもそれとこのルールに関する部分評価を施す必要がない。

さらに、多重度に関する無矛盾チェックを行うために以下のルールを追加する。

```
contradiction :-
  num(f_a(X)) >= I, num(f_a(X)) =< J,
  J<I.
```

クラス階層に関しては、以下のように変換する。クラス $c_1, ..., c_n$ の一般化が c であれば、

```
c(X) :- c1(X).
      ⋮
c(X) :- cn(X).
```

クラス $c_1, \ldots, c_n$ が排他であるときは、以下のように変換する。

すべての $i = 1, \ldots, n-1$ に対して、

```
contradiction :- ci(X),ci+1(X).
      ⋮
contradiction :- ci(X),cn(X).
```

さらに、すべてのクラス c に対して以下の事実を付け加える。

```
c(ec).
```

ここで e_c はそのクラスに対する固有の要素を表す。この事実を付加することで、クラスにインスタンスがある場合に矛盾が導かれるかどうかを検査することができる。

4 論理プログラミング上での無矛盾性チェック

変換した論理プログラムに対して、contradiction を呼び出して、それが成功すれば矛盾が検出される。ここでは、さらにどのルールの集合が矛盾しているかを検査することを考える。そのために各ファクトおよびルールに対してルール番号 N を付加し、N@G(ファクトの場合) または N@(G:-B)(ルールの場合) とし、さらに以下のメタインタプリタを用いることで矛盾ルール集合をすべて見つけることができる。

```
solve(G,UsedRules,NewUsedRules):-
    \+G=(_,_),
    !,
    solve1(G,UsedRules,NewUsedRules).
solve((G1,G2),UsedRules,NewUsedRules):-
    !,
    solve1(G1,UsedRules,UsedRules1),
    solve(G2,UsedRules1,NewUsedRules).

solve1(call(R),UsedRules,UsedRules):-!,R.
solve1(G,[N@G|UsedRules],UsedRules):-
    N@G.
solve1(G,[N@(G:-B)|UsedRules],NewUsedRules):-
    N@(G:-B),
    solve(B,UsedRules,NewUsedRules).
```

さらにそれらの矛盾集合をすべて見つけておいてその極小なものだけを選べば極小矛盾集合を見つけることもできる。

5 実行例

図 1 に矛盾を含む UML 図を示す。この図を上記の論理プログラムへ変化すると以下が得られる。

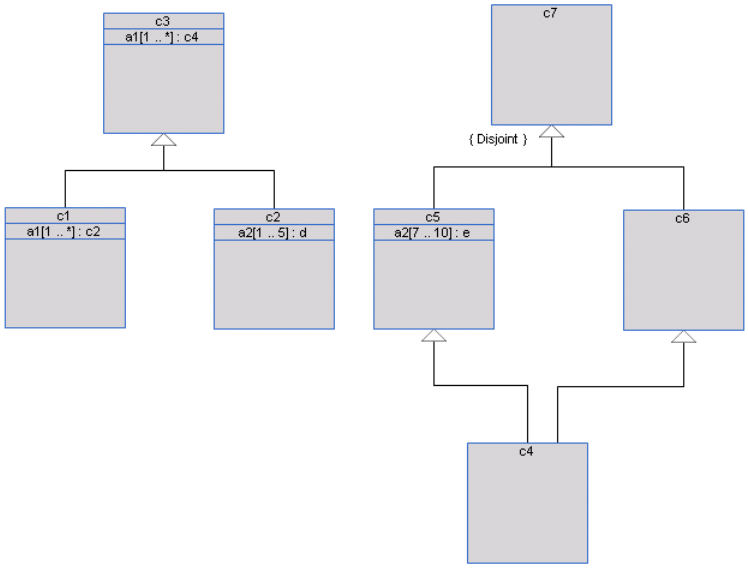

図 1: 矛盾を含む UML 図

```
c4(a1(X)):-c3(X).
num(a1(X)) >= 1:-c3(X).
c2(a1(X)):-c1(X).
num(a1(X)) >= 1:-c1(X).
d(a2(X)):-c2(X).
num(a2(X)) >= 1:-c2(X).
num(a2(X)) =< 5:-c2(X).
e(a2(X)):-c5(X).
num(a2(X)) >= 7:-c5(X).
num(a2(X)) =< 10:-c5(X).
c3(X):-c1(X).
c3(X):-c2(X).
c5(X):-c4(X).
c7(X):-c5(X).
c7(X):-c6(X).
c6(X):-c4(X).
contradiction:-c5(X), c6(X).
contradiction:-
    X >= I, X =< J, J<I.
c1(ec1).
c2(ec2).
c3(ec3).
c4(ec4).
c5(ec5).
c6(ec6).
c7(ec7).
d(ed).
e(ee).
```

これに対して各ルールに番号をつけ、上のメタインタプリタを実行させると以下の 2 つの極小矛盾ルール集合を得ることができる。ただし矛盾は「各クラスの要素が存在する」という事実をこのルール集合に付け加えたときに生じることに注意されたい。

1. 極小矛盾集合 1 :

```
contradiction:-c5(X), c6(X).
c5(X):-c4(X).
c6(X):-c4(X).
```

2. 極小矛盾集合 2 :

```
num(a2(X)) >= 7:-c5(X).
c5(X):-c4(X).
c4(a1(X)):-c3(X).
c3(X):-c1(X).
num(a2(X)) =< 5:-c2(X).
c2(a1(X)):-c1(X).
```

6 UML デバッガ

UML デバッガを試作した。UML エディタを用いて UML クラス図を作成し、その UML クラス図か

らそれをホーン節に変換して矛盾検出を行ない、矛盾があれば、その極小矛盾ルール集合を順に表示するシステムになっている。上の例では、図 2,3 に示す２つの極小矛盾ルール集合に対応する UML クラス図の部分の色が変化する。これは各ルールが UML クラス図の各部分に１：１に対応しているために可能となっている。

7 おわりに

クラス属性、多重度、汎化関係、排他関係 (disjoint) のみを許す UML のクラス図において無矛盾性チェックをするため、クラス図の各コンポーネントを論理プログラミングのルールとして表し、その論理プログラミング上で矛盾が導かれるかどうかをチェックする手法を提案し、さらにそのシステムを構築した。今後の課題としては、UML クラス図の他の表現（完全関係 (complete), 関連関係）への拡張がある。

参考文献

[1] Berardi, D., Cali, A., Calvanese, D. and De Giacomo, G., Reasoning on uml class diagrams. Technical Report 11-03, Dipartimento di Informatica e Sistemistica, Universita di Roma “La Sapienza” (2003).

[2] Berardi, D., Calvanese, D. and De Giacomo, G., Reasoning on uml class diagrams is exptime-hard. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)* (2003).

[3] Cali, A., Calvanese, D. and De Giacomo, G., Lenzerini, M., A formal framework for reasoning on UML class diagrams. *Lecture Notes in Computer Science*, 2366:503–513 (2002).

[4] Fowler, M., *UML Distilled: A Brief Guide to the Standard Modeling Object Language*. Object Technology Series. Addison-Wesley, third edition (2003).

[5] Kaneiwa, K., Satoh, K., *Consistency Checking Albgorithms for Restricted UML Class Diagrams*, NII technical report, NII-2005-013E (2005).

[6] Rumbaugh, J., Jacobson, I., and Booch, G., *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, USA, 1 edition (1999).

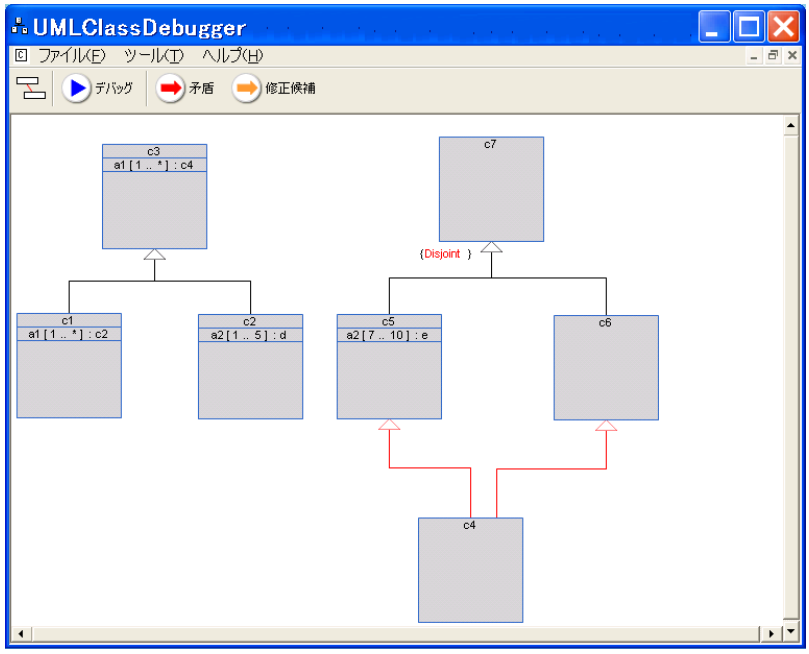

図 2: UML デバッガの出力 1

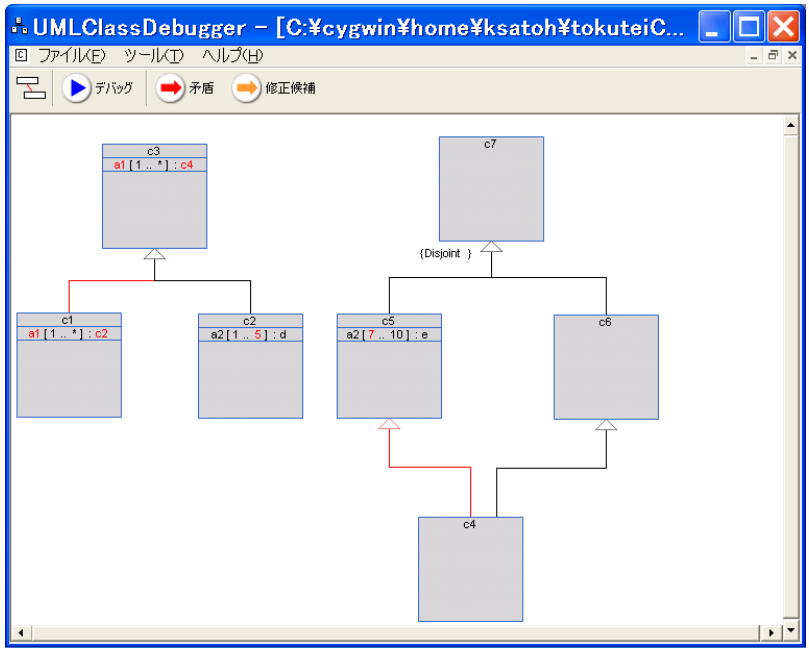

図 3: UML デバッガの出力 2