ソフトウェア工学 118－18
（1998．3．11）

オブジェクト指向方法論のための形式的モデルの検証

石田 至† 青木利晃† 片山卓也†

現在、品質の良いソフトウェアを開発するために、オブジェクト指向方法論が注目され、実際のシステム開発に適用されている。しかし、従来のオブジェクト指向開発法ではモデル化に関しての形式的取り扱いが十分ではなく、そのため計算機による支援を十分に行うことができていない。そのような要求に対して、FOVM が提案されている。FOVM を用いて分析モデルを構築することで、対象システムの形式的取り扱いが可能になる。本稿では、FOVM で定義されている分析モデルをもとに、モデルの構文チェックや、モデルの性質に関する検証の支援といった機能を提供する環境を構築することを目的とする。

Verification of Formal Model for Object-Oriented Methodology

ITARU ISHIDA,† TOSHIAKI AOKI† and TAKUYA KATAYAMA†

Object-oriented methodologies are focused for development of high quality software and applied to practical systems. But, traditional Object-oriented methodologies are not defined formal system modeling. Therefore it makes difficult to support system developments by computers .As for these requirements, FOVM (Formal model for Object-oriented Analysis Model) is proposed. It allowss us to support checking consistencies in models, semi-automatically. In this paper, for the purpose of constructing environment which provides syntax checker and property verification ability for analysis model in FOVM.

1. はじめに

システム開発の手法として、現在オブジェクト指向開発が注目されている。そのための方法論として、OMT[4] をはじめさまざまなものが提案されている。一方、大規模なシステム開発においては計算機による支援が必須であるが、今までの方法論で用いられているモデルは形式的取り扱いが十分ではない。そのため、システム開発において、計算機の支援を困難なものとしている。オブジェクト指向開発を形式的におこなうために、その分析モデルを形式化したFOVM (Formal model for Object-oriented Analysis Model)[1] が青木により提案されている。FOVM は、OMT で提案されているシステムを3つの側面により記述する分析モデルにもとづいて形式化をおこなったものである。このモデルを用いることで、形式的な分析モデルを構築することができる。それにより、オブジェクト指向開発を形式的にすすめることが可能になり、また、計算機による支援も容易になる。

† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology.

FOVM を用いて対象システムの分析モデルを構築する時、計算機による支援としては、対象システムのモデルに関する構文のチェックや well-formed チェック、さまざまな性質の検証等が挙げられる。しかしながら、FOVM は用いられている定義が多くその構造が複雑であるため、計算機の支援は重要であると考えられる。

そこで、本稿では FOVM を用いて構築されたモデルに関する検証環境の構築を目的とする。

2. FOVM

FOVM は、システム開発プロセスの分析フェーズで適用されるモデルアーキテクチャであり、従来のオブジェクト指向方法論で不十分であったモデル記述の形式的取り扱いに関して十分な基礎を与えるものである。

このモデルを用いて対象システムの分析モデルを構築することで、分析モデルを形式的に扱うことが可能になる。又、その形式性は計算機により取り扱い可能な水準であるため、従来の方法論のモデル記述では困難であった、計算機による詳細な支援が可能になる。

この形式的モデルは、OMT で提案されているオブジェクトモデル、動的モデル、機能モデルをベースと

した。これらのモデルは「三種の神器」と言われているER図、状態遷移図、データフロー図であり、システムの特徴を表現する場合の主成分でかつ直交するモデルとなっている。OMT法では、それぞれのモデルが持つシステムの特徴の情報は互いに干渉しており、個々の概念に対する開発者の曖昧な意味づけをもとにして一貫したモデルを構築する。このような思考の順序はどうあれ、モデルの仕組みとしては、視点毎に独立な部分とそれぞれの視点間を接続する部分が存在する。そこで、FOVMでは、このような独立な部分のモデルを基本モデル、視点間を接続したモデルを統合モデルとして形式化されている。

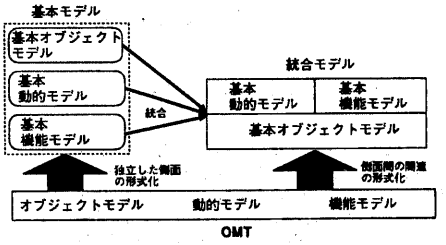

図1 形式化の方針

2.1 基本モデル

基本モデルは互いに独立な、基本オブジェクトモデル、基本動的モデル、基本機能モデルにより構成される。これらのモデルは、他の視点とは独立な識別子集合を基本集合として、それらの上に定義されている。各視点における構造は識別子から構成される式により表現される。このような識別子は、それぞれの視点固有の概念の抽象化であり、他の視点とは独立なため、それぞれのモデルを独立に定義することを可能にしている。又、識別子はその記号が持つ意味を定義する必要があり、これは意味記述として別途ドキュメント化される。このドキュメントは必ずしも形式的な記述である必要はなく、自然言語でもかまわない。基本モデルは、それぞれの視点の情報が複雑に絡み合った要求仕様を、視点毎に論理的に整理した世界を表現するものであり、意味記述の形式はどうあれ、それぞれの視点固有の概念の関係が形式的に定義されていれば良いのである☆。

2.2 統合モデル

独立に定義された基本モデルでは、側面は直交しているが、同じ対象システムを射影したものであるため、システムの同一の部分をモデル化しているものがある。そこで、意味が共通する部分を対応づけるメカニズムである統合写像の概念を導入する。このような対応関係を導入して独立に分析された結果を対応づけることにより、モデル同士のすりあわせやトレードオフがおこなわれ、それぞれの側面を反映した1つに統合されたモデルを構成することができる。対応づけは、基本モデルとそれぞれの構成要素の意味記述をもとにおこなわれ、対応関係を示す統合写像を形式的に定義する。これにより基本モデルでは意味記述として非形式的に記述されていた部分が段階的に形式的記述に変換されることになる。このように、統合モデルでは独立に定義された基本モデルと、それらの要素の間の対応関係が定義されている。そして、1つの一貫した分析モデルを提供している。

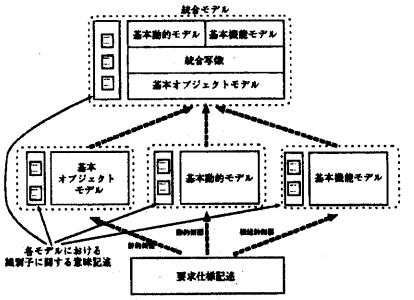

図2 FOVMのモデル構築の概念図

3. 検証支援環境

FOVMを用いて構築したモデルに対する検証としては、普遍的な検証と、その対象システムに依存した検証がある。普遍的な検証は、FOVMのモデルアーキテクチャに沿ったモデリングを行う際、一般的に適用できる検証であり、我々はすでに、モデルの一貫性の検証のための公理系[2]を提案している。このような検証法はFOVM特有の検証法として抽出可能である。一方、後者のような検証法は無数に存在し、あらかじめすべて抽出することは不可能である。そこで、検証を行うためのフレームワークを構築し、個々の検証を簡単に行えるようにする。このような支援環境では、前者のような普遍的な検証法はパッケージ化されたライブラリとして蓄えられ必要な時に適用できるようにすることで支援を行い、後者のような開発毎に依存した検証法は、フレームワーク上に実装することにより支援を行うようにする。

☆ 当然、意味記述の形式性が保証されていればより詳細な検証などが可能であるが、3つのモデルが保証する構造の本質ではない。

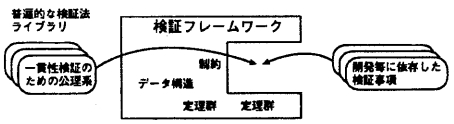

図3　検証フレームワークの概要

3.1 検証フレームワーク

FOΨM ではオブジェクト指向開発のために、対象システムを形式的に記述するためのモデルとその理論を提供している。このようなモデルと理論をフレームワークとして計算機システム上に実装することにより、FOΨM に基づいた検証の枠組を提供することができる。このようなフレームワークのことを、検証フレームワークと呼ぶ。検証フレームワークでは、FOΨM で定義されているデータ構造、補助関数、定義/定理群、制約などをフローズンスポットとして実装される。一方、開発毎に依存した検証法の具体的な手法やアルゴリズムなどは検証フレームワークに対するホットスポットとして、検証を行う際に実装される。

3.2 検証フレームワークの機能

検証フレームワークを実装する前に、その環境が持つべき機能を洗い出す必要がある。そこで、FOΨM で構築されたモデルに関する検証フレームワークを考えた場合、以下のような機能を持つべきであると考えられる。

(1) FOΨM で構築されたモデルの構文チェック
FOΨM を用いて構築されたモデルは、FOΨM の構文に従って記述されている必要がある。例えば、クラス式は属性識別子と関数識別子から構成するよう定義されている。それが定義と異なる識別子を用いて構成されていたら、それはモデルに関する構文を満たしていない。このような、モデルの間違った構築に関するチェックを簡単におこなう機能が必要である。

(2) FOΨM で構築されたモデルに関する well-formed チェック
FOΨM を用いて構築されたモデルの性質は FOΨM の理論が定義する制約を満たしている必要がある。たとえば、継承関係に対しては、その順序関係にはループが存在してはいけないという制約が定義される。構文的には正しくても、この制約を満たさないモデルを構築してしまう場合がある。検証フレームワークは、このような FOΨM 理論に関する、モデルの意味的な制約に関するチェック機能を提供するべきである。

(3) 対象システムの検証とその理解の支援
システム開発において対象システムが満たしている性質が明らかであることは少ない。モデル構築の後にそのモデルの性質を検証していくことで、そのシステムの性質を明らかにできる。検証フレームワークを利用することで、対象システムの性質を理解し、明らかにできるような機能が必要である。
また、モデルに関する性質に関して、それがモデル上のどのような規則から成立しているかを理解したいという要求もある。検証フレームワークは、以上のようなモデルの検証とその理解に関する支援を十分に行うことが可能な機能を提供するべきである。

4. 実装環境

4.1 定理証明系

本稿では、検証フレームワークの実装環境として定理証明系を用いる。これは、FOΨM の理論を公理系として、FOΨM が含んでいる定義を公理群として実装することが可能であり、対象システムの性質の検証を、証明として実装することが可能となるからである。さらに、性質の検証を証明によっておこなうことは、その証明過程を理解することにより、対象システムの性質の深い理解につながる。また、FOΨM の識別子の意味記述を利用することが可能になる。

現在、計算機の計算速度の飛躍的な向上と、自動定理証明アルゴリズムの研究の成果として、多くの有用な定理証明系が実装されている。我々はまず、検証フレームワークを実装するにあたり、どのような定理証明系が適切であるかを検討した。そこで、異なる性質を持つ定理証明系として PVS と HOL により実験を試みた。

4.1.1 PVS

PVS[6] は SRI で開発された定理証明系であり、プロトタイプ記述言語と妥当性評価のための論理証明を支援する機能で構成されている。プロトタイプ記述言語は高階述語論理に基づいた言語であり、FOΨM のデータ構造や性質をこの言語で記述した。

この PVS の用途は、プロトタイプ記述言語で記述された内容の妥当性評価を行うことである。よって、以上のように実装を行った場合、FOΨM のデータ構造や性質を基礎として、それらを用いて他の検証事項を証明するのではなく、FOΨM の実装自身の妥当性の評価の部分も含めた証明になってしまった。この場

合、検証事項の内容に集中して証明することは不可能であり、本来必要ではないFOVMの実装自身の展開や妥当性証明が検証作業の大部分となる。

4.2 HOL

実装環境として用いる定理証明系が満たすべき性質としては、その定理証明系の上に、更に環境を構築することが可能であるような、メタ言語的な利用が可能な性質を持っていることが望ましい。

そこで本研究では、検証フレームワーク構築のベースとしてHOL[5]を用いる。HOLはML上に実装された定理証明系で、以下の特徴を持つ。

- MLからHOLの環境に対する操作が可能である。
  定理証明系がML上に構築されていることから、HOLの項や定理を扱うML関数を記述することが可能で、証明の際に関数の副作用を利用できる。また、証明の過程をML変数に保存しておき、後の証明に利用することが可能である。
- 高階論理をサポートしている。
  高階論理をサポートしていることにより、ラムダ計算を利用したユーザによる柔軟な型の定義が可能となっている。
- 公理系のモジュール的な扱いが可能である。
  HOLでは1つの公理系に関する公理や定理を、theoryという単位で扱うことが可能である。公理系はtheory単位でモジュールとして扱うことができ、他の公理系の定義と明確に分割することができる。証明はtheoryを読み込むことによって、そのモジュールで定義されている公理系に関する性質を利用しておこなわれる。

HOLをFOVMの検証フレームワークを構築する環境として見たときに、ML環境を用いた検証補助環境を構築できる点や、theoryのモジュール性、柔軟にユーザ定義型を作ることがが可能な点で有効である。

5. 検証フレームワークの実装

5.1 実装の方針

FOVMの検証フレームワークを構築するにあたり、FOVMの理論をどのようにHOL上に実装するかを決定する必要がある。そのために、FOVMの理論の構造を明らかにする。そして、その構造を考慮した上でのFOVMの理論の実装方針を決定する。

5.1.1 FOVMの構造

FOVMの理論は、以下のようなマクロ的な構造と、ミクロ的な構造を持つ。

- FOVMの理論のマクロ的構造。
  FOVMの理論は3つの基本モデルと統合モデルから構成されている。各基本モデルはそれぞれ他の基本モデルから独立に定義されており、統合モデルは基本モデルと統合写像の概念により定義されている。
- FOVMの理論のミクロ的構造
  FOVMを構成する各モデルは、それぞれのモデルにおける対象システムの構造を表現する要素(式や識別子)と、その要素が満たすべき性質から構成される。それらは各モデルで独立した定義である。

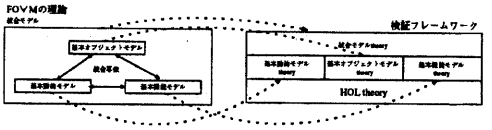

図4 FOVM理論の実装の方針

このような構造を踏まえて、FOVMの検証フレームワークの構築の方針を決定する。

5.1.2 フレームワークの実装の方針

まず、FOVMの検証フレームワークのマクロ的な構造を実装する。検証フレームワークでは、FOVMの各基本モデルに関して、それぞれ独立したHOLの公理系(theory)として実装する。統合モデルはその基本モデルのtheoryを用いて定義されるthoeryとして実装する。theoryのモジュール性を利用することで、検証の際には各モデルの公理系を読み込んで利用することが可能になる。

各モデルの公理系に関しては、そのモデルを構成する要素である識別子や式等をデータ構造として定義する。そして、データ構造の上に定義される性質を、公理群として定義する。

このように、FOVMにおけるデータ構造と基本的性質の定義を核として、各モデルのtheoryを実装する。そして、各モデルのtheory全体で、FOVMの検証フレームワークを構築する。

5.2 識別子の実装

識別子は、FOVM理論を構成する最小単位の要素である。各基本モデルは、識別子を基本集合として定義される。よって、各基本モデルの構成要素は、識別子のレベルまで分解することができる。各識別子はそれぞれ他の識別子と明確に区別されるものなので、それぞれの種類の識別子をHOL上の1つの型として実装する。識別子を表現する型を識別子型とする。識別子型は、文字列型から型コンストラクタによって構築する。

5.2.1 識別子の実装例

FOVM の基本オブジェクトモデルには、クラスの構成に関する識別子として、クラス識別子、属性識別子、関数識別子がある。それらを例にした識別子の FOVM による記述と HOL 上の実装例を以下に示す。$ClassID$ はクラス識別子集合、$AttrID$ は属性識別子集合、$FuncID$ は関数識別子集合である。また、$classA, attrA, funcA$ はそれぞれの識別子集合上のメタ変数を表す。

- FOVM の記述

  $classA \in ClassID, attrA \in AttrID,$
  $funcA \in FuncID$

- HOL の記述

  (--‘ ‘--) でかこまれた表記は、HOL 上の項を示す。

```
(--‘ CLASSID "classA" ‘--),
(--‘ ATTRID  "attrA" ‘--)
(--‘ FUNCID  "funcA" ‘--)
```

5.3 式の実装

FOVM における式は、対象システムの各視点における構造を表現するもので、識別子から構成される。各基本モデルは、自身の構造を表現する複数の式を持っている。よって式は、その式を構成する識別子型と型コンストラクタを用いて実装する。

5.3.1 式の実装例

FOVM では、継承関係を継承式で定義している。継承式は継承関係にある親クラスと子クラスのクラス識別子から構成される。$classA, classB, classC$ をクラス識別子集合 $ClassID$ 上の変数として、$classB$ と $classC$ が $classA$ を継承するという継承関係を表現する継承式の例を以下に示す。

- FOVM の記述

  $(classA, (classB, classC))$
  $classA, classB, classC \in ClassID$

- HOL の記述

```
(--‘ INHER (CLASSID "classA")
           [(CLASSID "classB");
            (CLASSID "classC")] ‘--)
```

5.4 写像の実装

FOVM では、写像を用いて識別子とそれに対応する式を決定している。この写像により、識別子からその識別子が示す式を特定することが可能である。検証フレームワークでの実装としては、識別子型から式型への写像をおこなう関数を実装する。

5.4.1 写像の実装例

クラス識別子に対応するクラス式への写像の例を示す。クラス式は属性識別子と関数識別子から構成される。$classA, attrA, funcA$ はそれぞれクラス識別子集合 $ClassID$、属性識別子集合 $AttrID$、関数識別子集合 $FuncID$ 上の変数である。

- FOVM の記述

  $classA = ((attrA), (funcA))$
  $classA \in ClassID, attrA \in AttrID,$
  $funcA \in FuncID$

- HOL の記述

```
(--‘ CLASS_MAP (CLASSID "classA") =
     (CLASS [(ATTRID "attrA")]
            [(FUNCID "funcA")]) ‘--)
```

5.5 規則/制約の実装

FOVM では識別子や式、写像などの対象システムの構造に関するデータ構造を表す要素に対して、その上に成立する規則や制約を定義している。それらの規則や制約に関しては、これまでで定義した FOVM のデータ構造をもとに定義される公理や定理として実装する。

5.5.1 規則の実装例

FOVM では継承式に対して、継承関係にあるクラス識別子の間に継承関係を順序関係とする半順序関係を定義している。また、この順序関係には推移律と非対称性が成立している。これは継承式に対する規則である。式の実装で示した継承式で定義されている $classA, classB$ の間の順序関係を例にして、実装例を示す。

- FOVM の記述

  $classB \prec classA$
  $classA, classB \in ClassID$

- HOL の記述

```
(--‘ INHER_ORDER (CLASSID "classB")
                 (CLASSID "classA") ‘--)
```

5.6 モデル情報

これまでの定義にしたがって、検証フレームワークにおけるフローズンスポットの部分を実装した。検証フレームワークで実際に検証を行うためには、対象システムに関する情報を与える必要がある。これをモデル情報とする。

モデル情報は、識別子や式などの型をとる述語を用いて、論理式の形で検証フレームワークに与える。

証明の際には、このモデル情報として与えられたもののみが、対象システムの上で成立すると定義する。よって、対象システムの性質に関しては、全てモデル

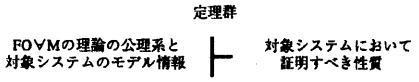

図5　FOVM で記述されたモデルの検証方針

情報と FOVM の理論の公理系から導くことが可能である。逆に、モデル情報から導くことが出来ない性質は、もとのモデルで成立しない性質であるといえる。

5.7 検証方針

構築した検証フレームワークを用いて対象システムのモデルに関する検証を行う場合は、モデル情報を入力して、検証フレームワークで定義されている公理系を利用して検証を行う。

検証は、ますモデル情報を前提条件として与える。そして、対象システムにおいて検証したい性質を証明のゴールとして与える。前提条件から、検証フレームワークで定義されている公理群を用いてゴールとして与えられた性質が証明できれば、その性質が対象システムのモデルにおいて成立していることになる。

つまり、与えられたモデル情報から目的とする性質が証明できれば、その性質は対象システム上で成立していることを保証できることになる。

5.8 実装のまとめ

この節では、基本オブジェクトモデルをもとに検証フレームワークの実装に関して述べた。他のモデルに関しても、基本的に構造が同じであるため、同様の方針で実装する。結果として、FOVM を構成する各モデルの実装ができ、FOVM 理論を HOL 上に実装できた。

6. 検　　証

この節では、簡単な例題をもとに FOVM の検証フレームワークの利用に関する解説を行う。

6.1 例　　題

図6の3つのクラス間の継承関係を表す基本オブジェクトモデルを用いて検証の例を示す。また、FOVM の記述の $inherA$ は図の継承式に対応する継承識別子である。$classA, classB, classC$ はクラス識別子集合 $ClassID$、$attrA, attrB, attrC$ は属性識別子集合 $AttrID$、$funcA, funcB, funcC$ は関数識別子集合 $FuncID$、$inherA$ は継承識別子集合 $InherID$ 上の、それぞれのメタ変数である。

$inherA = (classA, (classB, classC))$
$classA = ((attrA), (funcA))$
$classB = ((attrB), (funcB))$

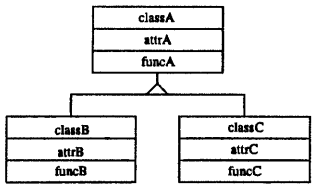

図6　OMT の図

$classC = ((attrC), (funcC))$
$inherA \in InherID,$
$classA, classB, classC \in ClassID,$
$attrA, attrB, attrC \in AttrID,$
$funcA, funcB, funcC \in FuncID$

検証項目として、クラス $classB$ が属性 $attrA$ を持つことの検証を設定する。モデル情報としては、クラス $classA$ とクラス $classB$ を含んだ継承式と、クラス識別子 $classA$ からそれに対応するクラス式への写像を与える。

6.2 検証手順

- 検証は、基本オブジェクトモデルに関したものであるため、まずその theory を読み込む。これにより、基本オブジェクトモデルを扱うための定義が利用可能になる。
- 次に、検証に必要なモデル情報を仮定とし、検証したい性質を結論とする論理式を証明のゴールとして設定する。以後の証明ははゴールの結論部分のみ表記する。

```
(DEFINE_INHER
     (INHER (CLASSID "classA")
            [(CLASSID "classB");
             (CLASSID "classC")])) /\
(CLASS_MAP (CLASSID "classA") =
           (CLASS [(ATTRID "attrA")]
                  [(FUNCID "funcA")])
==> (HAS_ATTR (CLASSID "classB")
              (ATTRID "attrA"))
```

- HAS_ATTR の定義を用いて結論部を書き換える。

```
(MEMBER (ATTRID "attrA")
        (ATTR (CLASS_MAP
                (CLASSID "classB")))) \/
(?parent. (INHER_ORDER
           (CLASSID "classB") parent) /\
(MEMBER (ATTRID "attrA")
        (ATTR (CLASS_MAP parent) )))
```

- 継承関係の順序関係 INHER_ORDER を継承式から導くことができる。仮定の条件よりより parent をクラス識別子 $classA$ に具象化できる。

```
(MEMBER (ATTRID "attrA")
        (ATTR (CLASS_MAP
              (CLASSID "classB")))) \/
((INHER_ORDER (CLASSID "classB")
              (CLASSID "classA")) /\
(MEMBER  (ATTRID "attrA")
         (ATTR (CLASS_MAP parent) )))
```

- よって、INHER_ORDER が成立することを証明できた。

```
(MEMBER (ATTRID "attrA")
       (ATTR (CLASS_MAP
             (CLASSID "classB")))) \/
( T /\
(MEMBER  (ATTRID "attrA")
       (ATTR (CLASS_MAP
             (CLASSID "classA"))))
```

ここで、モデル情報として与えられているクラス識別子の写像より、クラス識別子 *classA* に対応するクラス式を導くことができる。

```
(MEMBER (ATTRID "attrA")
        (ATTR (CLASS_MAP
              (CLASSID "classB")))) \/
(MEMBER (ATTRID "attrA")
        (ATTR (CLASS [(ATTRID "attrA")]
                     [(FUNCID "funcA")])))
```

ATTR はクラス式からそのクラス式を構成する属性集合を返す関数であり、それにより以下を導ける。

```
(MEMBER (ATTRID "attrA")
        (ATTR (CLASS_MAP
              (CLASSID "classB")))) \/
(MEMBER (ATTRID "attrA")
        [(ATTRID "attrA")])))
```

- そして、MEMBER の定義より書き換えを行う。その結果として、ゴールの証明が終了する。

```
(MEMBER (ATTRID "attrA")
        (ATTR (CLASS_MAP
              (CLASSID "classB")))) \/
 T
```

性質の検証はこのようにおこなわれる。

6.2.1 証明に用いる定義

- HAS_ATTR の定義。

```
!class. ?attr. (HAS_ATTR class attr) =
   ?parent.
    (MEMBER attr (ATTR (CLASS_MAP class) )) \/
    ((INHER_ORDER class parent) /\
    (MEMBER attr (ATTR (CLASS_MAP parent) )))
```

- INHER_ORDER の定義。

```
!parent child.
   INHER_ORDER child parent =
         ?iexp. DEFINE_INHER iexp /\
             (parent = (INHER_PARENT iexp)) /\
                (MEMBER child (INHER_CHILD iexp))
```

7. ま と め

本研究では、オブジェクト指向方法論のための形式的モデルである FOΛM のための検証フレームワークを構築した。

始めに検証フレームワークとして必要な機能を分析し、実装の視点から FOΛM の理論を考察し、その理論の実装方針を決定した。次に、その実装方針にしたがい実装環境を決定し、FOΛM の理論を扱うことができる検証フレームワークを実装した。

また、実装した検証フレームワークの評価を行い、その有効性を確認した。

今後の課題としては、証明の過程における中間定理の制御や目的の証明のための TACTIC の実装、そして FOΛM のモデル構築支援環境とのインターフェースなどが考えられる。

参 考 文 献

1) 青木利晃, 片山卓也: オブジェクト指向方法論のための形式的モデル, , 情報処理学会 OO'96 シンポジウム,pp.25-32,1996.
2) 青木利晃, 片山卓也: モデルの一貫性の検証のための公理系, , ソフトウェア科学会 第 3 回 ソフトウェア工学の基礎ワークショップ,pp178-181.
3) 青木利晃, 石田至, 古川順一, 片山卓也: オブジェクト指向方法論のための形式的モデル, ソフトウェア科学会 第 14 回全国大会,pp 465-468 , 1997.
4) Ramgaugh,J.Blaha,M.,Premerlani,M.,Eddy,F. and Lorensen,W.: Obect-Oriented modeling and design, Prentice-Hall International, 1991.
5) M.J.C.Gordon, T.F.Melham: Introduction to HOL, CAMBRIDGE UNIVERSITY PRESS, 1993
6) Judy,Crow,Sam Owre, John Rushby, Natarajan Shankar,Mandayam Srivas: A Tutorial Introduction to PVS, WIFT'95, 1995
7) 石田至: オブジェクト指向方法論のための形式的モデルの検証, Master's thesis, Japan Advanced Institute of Science and Technology, 1998.