ソフトウェア工学 96－5
（1994．1．20）

オブジェクト指向プログラミングにおける デメテル法則の定式化

黄 錫炯† 梁 海述†† 辻野 嘉宏† 都倉 信樹†

† 大阪大学基礎工学部情報工学科
†† 韓国 江原大学校 自然科学大学 電子計算学科

オブジェクト指向プログラミングにおいて、クラス間の不必要な相互依存関係を減らすためのプログラミングスタイルに関する規則としてデメテルの法則 (The Law of Demeter) が知られている。本稿では、従来の非形式的なデメテルの法則を実際のプログラムに適用、評価するために、クラス間の関係として継承と集約、そして関連を形式的に定義し、それらを用いてデメテルの法則を定式化する。またデメテルの法則を満たすか否かを判定するアルゴリズムと、デメテルの法則を違反したプログラムに対する変換アルゴリズムについて述べる。

和文キーワード オブジェクト指向、デメテルの法則、プログラミングスタイル

Formulations of the Law of Demeter in the Object Oriented Programming

Sukhyung HWANG† Haesool YANG†† Yoshihiro TSUJINO† Nobuki TOKURA†
{hwang, tsujino, tokura}@ics.es.osaka-u.ac.jp yang@cc.kangwon.ac.kr

† Faculty of Engineering Science, Osaka University, 560 JAPAN
†† College of Natural Science, Kangwon National University, 200-701 KOREA

In the last few years, several articles have been devoted to the study of the Law of Demeter in the object oriented programming. The Law of Demeter is a style rule that aims at eliminating unnecessary coupling among classes. Although a large number of studies have been made on the informal definitions, little is known about the formulation of the Law of Demeter.

In this article, we define three relationships among classes i.e. inheritance, aggregation and association, and formulate the Law of Demeter. We also propose the algorithms to decide whether a given program satisfies the law and to transform an unfulfilled program into a fulfilled one.

英文 key words Object-Oriented, the Law of Demeter, Programming Style

1 はじめに

この数年、オブジェクト指向に対する関心が高まり、数多くのオブジェクト指向言語が提案されてきた。また、再利用性と情報の隠蔽、継承、多態性を利用したソフトウェアの構築、そして分析と設計に関するオブジェクト指向方法論の登場など、ソフトウェア工学にも新しいパラダイムの時代を迎えるようになった [2]。

しかし、オブジェクト指向プログラミング言語で作成したプログラムに対する評価や複雑さに関する尺度の提案についての研究はまだ進んでいないのが現状である。とはいえ、最近、オブジェクト指向言語のクラスの品質評価に関する研究 [1] が行われている。この研究では、クラスの品質評価尺度として、クラスをモジュールと考えたモジュール強度、そしてクラス間の継承関係による結合度といった評価尺度を提案している。さらに提案尺度を C++プログラムに適用した結果、多くのソフトウェア開発者が抽象データ型と継承機構を理解せずに開発に取り組んでいることが報告されている。

また、クラスの間の相互依存関係に注目して、メッセージの宛先に制限を課すことに関する研究 [3, 4] が行われてきた。この研究では、クラス間の結合度を減らすための制限として、デメテルの法則 (The Law of Demeter) という指針を提案している。あるクラスがこの法則に従うと、そのクラスのメソッドは制限されたクラスのメソッドのみを利用するようになり、クラスの間の結合度が低くなる。しかし、この研究では、非形式的なデメテルの法則の定義とその法則に従うようにプログラムを変換する方法のアイデアの提案だけで、形式的な定義が行われていない。

本研究では、クラス間の相互依存関係に注目してデメテルの法則を定式化し、与えられたオブジェクト指向プログラム (C++) に対して判定するアルゴリズムを提案する。また違反プログラムを変換するいくつかの方法 [3, 4] の基本となったリレーメソッド追加法のアルゴリズムを定式化し、その正当性を検討する。最後にはデメテルの法則を満たすための新しい変換方法に対して考える。

2 諸定義

ここでは、プログラム上に登場するクラスとメソッド、そしてクラス間の関係などに対する定義を行う。まず、プログラム $\mathcal{P}$のクラスの集合を $\mathcal{C}_\mathcal{P}$、メソッドの集合を $\mathcal{M}_\mathcal{P}$とする。

2.1 クラスの定義から求められる集合の定義

クラスにはデータとそれをアクセスするためのメソッドが定義されている。クラスに定義したデータをデータメンバと呼び、それには他のクラスのインスタンス変数も定義できる。また、クラスの継承関係によって各クラスにはその親クラス (祖先クラス) が定義される。以上のことに基づき、任意のクラス Cに対して、次の集合を定義する。

$OB(C)$: クラス Cのオブジェクトの集合
$MD(C)$: クラス Cに定義されたデータメンバのクラスの集合
$MF(C)$: クラス Cに定義されたメソッドの集合

但し、$MF(C), MD(C)$ には継承されたメソッドやデータメンバのクラスは含まれない。

継承に関する定義を行うために、クラス Cがクラス Pを直接に継承するというのを $P \gg C$または、$P \overset{1}{\gg} C$と表し、クラス Pから n 回の継承を経てクラス Cが得られた場合は $P \overset{n}{\gg} C$と表現する。また、クラス Cがクラス Pを 0 階層以上に渡って継承した場合には $P \overset{*}{\gg} C$と表す。これに基づいて、クラス Cの祖先クラスの集合 $SC(C)$ を次のように定義する。

$$SC(C) \triangleq \{P \in \mathcal{C}_\mathcal{P} \mid P \overset{*}{\gg} C\}, \text{ where } C \in \mathcal{C}_\mathcal{P}$$

最後に、任意のメソッド m に対して、メソッド m が定義されたクラスを $CM(m)$ と定義する。すなわち、$m \in MF(CM(m))$ である。

2.2 述語の定義

任意の 2 つのクラスの間にメッセージのやりとりとか一方のクラスで他方のクラスのデータメンバを参照する場合、両クラス間には相互依存関係があるという。しかし、本研究ではクラス間のメッセージ転送のみに着目する。実際のプログラムではそのような振舞いはクラスに定義したメソッドに記述される。メソッド m に関して、次のような述語を定義する。

$ref(m, i)$: メソッド m のボディー部でグローバル変数 i を参照する
$arg(m, a)$: メソッド m の仮引数リストにオブ

ジェクト a がある

$contain(m,\ C)$: メソッド m のボディー部にクラス Cのオブジェクトを生成する文を含む

$send(m, f, C)$: メソッド m 中にクラス Cのオブジェクトにメッセージ fを送る文を含む

これらの述語は静的に型付けられた言語ではプログラムテキストから真偽を決定できる。

2.3 メソッドに関するオブジェクトのクラス

任意のメソッド m に対して、そのメソッドの内部で現れたオブジェクトのクラスの集合を次のように定義する。

- メソッド m で使われているグローバル変数のクラスの集合 $GV(m)$
  $GV(m) \triangleq \{A \in \mathcal{C}_\mathcal{P} \mid ref(m,\ i),\ i \in OB(A)\}$
- メソッド m の引数リストで使われているオブジェクトのクラスの集合 $AC(m)$
  $AC(m) \triangleq \{A \in \mathcal{C}_\mathcal{P} \mid arg(m,\ a),\ a \in OB(A)\}$
- メソッド m が属しているクラスに定義されたデータメンバのクラスの集合 $IV(m)$
  $IV(m) \triangleq \{A \in \mathcal{C}_\mathcal{P} \mid A \in MD(CM(m))\}$
- メソッド m で新しく生成されたオブジェクトのクラスの集合 $CRE(m)$
  $CRE(m) \triangleq \{A \in \mathcal{C}_\mathcal{P} \mid contain(m,\ A)\}$

最後に、メソッド m の供給者クラス $SU(m)$ を、

$$SU(m) \triangleq \{A \in \mathcal{C}_\mathcal{P} | send(m, f, A),\ f \in \mathcal{M}_\mathcal{P}\}$$

のように定義する。つまり、メソッド m の供給者クラスは、メソッド m のボディー部で呼び出されるメソッドが定義されたクラスである。

2.4 クラス間の関係

クラスの間には以下のような3つの関係がある。

継承 (*i*nheritance)

$$i = \{(B, A) | A, B \in \mathcal{C}_\mathcal{P} \wedge A \overset{*}{\gg} B\}$$

継承はクラスの間の関係であり、各子供クラスは親クラスの属性とメソッドを共有する。

集約 (*p*art_of)

$$p = \{(A, B) | A, B \in \mathcal{C}_\mathcal{P} \wedge B \in MD(A)\}$$

集約は、「全体-部分」あるいは"a-part-of" と表現される関係である。それは何かの部品を表すクラスを全体の組立を表すクラスに関連づけるものである。

関連 (*a*ssociation)

$$a = \{(A, B) | A, B \in \mathcal{C}_\mathcal{P} \wedge (B \in GV(m) \vee B \in AC(m) \vee B \in CRE(m)) \wedge m \in MF(A)\}$$

関連は、あるクラスとそのクラスのメソッドで参照するクラスとの関係である。ここではメソッドで参照したグローバル変数のクラスと、メソッドの引数に現れてボディー部で参照したクラス、そしてメソッドで新しく生成されたオブジェクトのクラス、という3つの場合のみを考える。

3 デメテルの法則の定式化

3.1 Strong/Weak version の定式化

提案されたデメテルの法則 [3] ではクラス間の関係やメッセージ転送について定義が曖昧であるので、実際に与えられたプログラムに適用する場合、適用する人によって結果が異なる場合がある。ここでは、前節の諸定義を用いてより明確にデメテルの法則 (Strong version) を定義する。

定義 1 デメテルの法則 *(Strong version)*

プログラム $\mathcal{P}$は
$\forall C \in \mathcal{C}_\mathcal{P}[\forall m \in MF(C)\{$
$SU(m) \subset IV(m) \cup AC(m) \cup CRE(m) \cup GV(m)\}]$
を満たさなくてはならない。

すなわち、任意のクラス Cとそのクラスに属しているメソッド m に対して、メソッド m からメッセージを送れるクラスとしては、クラス Cに定義されたデータメンバのクラス、メソッド m の引数オブジェクトのクラス、メソッド m で生成されたオブジェクトのクラス、メソッド m で使われているグローバル変数のクラスでなければならない。

定義 1 を少し緩めて祖先クラスに定義したデータメンバのクラスのインスタンスへのメッセージ転送を許すことも考えられる (Weak version)。その場合には次のように定義できる。

定義 2 デメテルの法則 *(Weak version)*

プログラム $\mathcal{P}$は、
$\forall C \in \mathcal{C}_{\mathcal{P}}[\forall m \in MF(C)\{$
$SU(m) \subset TIV(m) \cup AC(m) \cup CRE(m) \cup GV(m)\}]$
を満たさなくてはならない。

但し、$TIV(m) = \bigcup_{A \in SC(C)} MD(A)$

すなわち、任意のクラス Cとそのクラスに属しているメソッド m に対して、メソッド m からメッセージ転送のできるオブジェクトのクラスは、クラス Cとその祖先クラスに定義されたデータメンバのクラス、メソッド m の引数クラス、メソッド m で生成されたオブジェクトのクラス、メソッド m で使われているグローバル変数のクラスである。つまり、**Weak version** では継承関係に基づいて、祖先クラスに定義されたデータメンバのクラスへもメッセージを転送できる。

3.2 クラスの分類

大抵のクラス間のメッセージのやりとりはクラス間の関係に沿って行われる。従って、受け手のクラスと送り手のクラスとの間にはどのような関係であるかを知って置くことによってデメテルの法則をより明確に理解できる。以下では、クラスの各メソッドに対してその受け手のクラスを分類する [4]。

あるクラス **C** に定義された任意のメソッド **m** のボディー部に他のクラス **D** にメッセージ **f** を送る文がある場合、クラス **C** はメッセージ **f** に関して依頼者クラスとなり、クラス **D** はメッセージ **f** に関してメソッド **m** の供給者クラスとなる。このような依頼者-供給者モデルに基づき、任意のメソッド m の供給者クラスのうち、以下の 3 種のクラスに着目する。

知り合いクラス $(AQ\_C(m))$

$$AQ\_C(m) \triangleq \{A \in \mathcal{C}_{\mathcal{P}} \mid A \in SU(m) \text{ and } (A \notin SC(AC(m)) \text{ and } A \notin SC(IV(m)) \text{ and } A \notin SC(CM(m)))\}$$

上記の式は次のように表すことができる。

$$AQ\_C(m) = SU(m) - SC(AC(m)) - SC(IV(m)) - SC(CM(m))$$

すなわち、知り合いクラスは、引数クラスでも、データメンバのクラスでも、そして、メソッド m が属しているクラスでもない供給者クラスであり、メソッド m によって生成されたオブジェクトのクラスやグローバル変数のクラスのインスタンスにメッセージを送る場合などが当てはまる。

優先知り合いクラス $(PA\_C(m))$

$$PA\_C(m) = SC(CRE(m)) \cup SC(GV(m))$$

メソッド m を起動することによってその結果として生成されるオブジェクトのクラスと、メソッドのボディー部で用いるグローバル変数のクラスは、知り合いクラスの中で比較的関係の深いクラスで、ここでは優先知り合いクラスとして分類する。

優先供給者クラス $(PS\_C(m))$

$$PS\_C(m) \triangleq \{B \in \mathcal{C}_{\mathcal{P}} \mid B \in SU(m) \text{ and } (B \in SC(IV(m)) \text{ or } B \in SC(AC(m)) \text{ or } B \in SC(CM(m)) \text{ or } B \in PA\_C(m))\}$$

上の式は次のようにも表せる。

$$PS\_C(m) = SU(m) \cap (PA\_C(m) \cup SC(IV(m)) \cup SC(AC(m)) \cup SC(CM(m)))$$

優先供給者クラスは、優先知り合いクラス、メソッド m が属しているクラスに定義されたデータメンバのクラス、メソッド m の引数クラス、メソッド m が定義されたクラス、またはそれらの祖先クラスであるという条件を満たす供給者クラスである。

3.3 Strict version の定式化

前節で定義したクラスの分類に基づいて、3 番目のデメテルの法則 (Strict version)[4] を定義する。

定義 3 デメテルの法則 *(Strict version)*

プログラム $\mathcal{P}$は、
$\forall C \in \mathcal{C}_{\mathcal{P}}[\forall m \in MF(C)\{SU(m) = PS\_C(m)\}]$
を満たさなくてはならない。

すなわち、全てのメソッドの供給者クラスは優先供給者クラスでなければならないということである。

デメテルの法則 (Strict version) に従うようにプログラミングを行うと、クラス間の相互依存関係を局所化でき、クラスの記述を理解しやすくなる。

4 判定アルゴリズム

ここでは、与えられたプログラムに対して、そのプログラムがデメテルの法則を満たすか否かを判定するアルゴリズムを提案する。まず、アルゴリズムを記述するために次のようなクラス階層グラフ G を定義する。

定義 4 クラス階層グラフ

$\mathbf{G} = (V, E)$

V : クラスの集合, $E = RE \cup ME$

RE : 関係辺の集合, $RE \subseteq V \times V \times R$

ME : メッセージ辺の集合,

$ME \subseteq V \times V \times W$

R : 継承 (i) *or* 集約 (p) *or* 関連 (a)

W : メッセージ転送文の数 (正の整数)

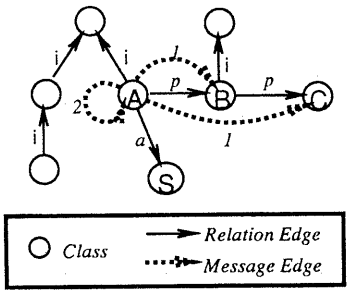


図 1: クラス階層グラフの例

与えられたオブジェクト指向プログラムはクラス階層グラフ G に変換できる。クラス階層グラフ G は、クラスを頂点とし、REとMEの 2 通りの辺をもつ。

REの辺はクラス間の関係を表し、クラス間の関係、継承 (i)、集約 (p)、関連 (a) の内、いずれか 1 つがラベルとして付けられる。任意の 2 頂点 vと w との間に直接の継承関係辺が存在する時 $(w \gg v)$、$(v, w, i) \in RE$と表す。また、n 階層に渡って継承した場合 $(w \overset{n}{\gg} v)$ には、$(v, w, i^n) \in RE$と表す。同様に任意の 2 頂点 v, wが集約関係で結びつけられた場合 $(w \in MD(v))$、$(v, w, p) \in RE$と表す。関連関係についても同様である。一方、MEの辺はメッセージ転送を表す。すなわち、$m \in MF(v)$ の時、$send(m, f, w)$ であれば、$(v, w, W) \in ME$となる。但し、Wは、クラス vのメソッドの中に書かれたクラス wへのメッセージ転送文の総数を意味する。図 1 にクラス階層グラフの例を示す。以上のデメテルの法則の定義とクラス階層グラフ G に基づいて、判定アルゴリズムを提案する。

与えられたプログラムを上記のクラス階層グラフに変換し、判定アルゴリズムの入力とする。アルゴリズム Demeter では、先ずグラフ上の各頂点のメソッド m に対して、そのボディ部のメッセージ文から供給者クラス $SU(m)$ を求める。次には、任意の頂点 x から集約関係の頂点 yを求めて集合 $MD(x)$ とする。また、関連関係に対して同様に集合 $GAC(x)$ を求める。この 2 つの集合 ($MD(x)$ と $GAC(x)$) とクラス x をクラス x の暫定優先供給者クラス ($TPS(x)$) とする。次に頂点 x と yが継承関係 (x が yの子供クラス) であれば、クラス x の暫定優先供給者クラスはクラス yの暫定優先供給者クラスをも含むようにする。

以上を深さ優先探索を用いて未探索頂点が存在しない時まで再帰的に行い、各依頼者クラスに対して、定義 3 のデメテルの法則を用いて判定する。最後には、デメテル法則を違反するメソッドの集合 $VD\_M(v)$ と、その違反メソッドの供給者クラスの集合 $VS\_C(m)$ を出力する。ここで、全てのクラスに対して、集合 $VD\_M(v)$ が空集合の場合は入力プログラムはデメテル法則を満たすという意味で適切なスタイルのプログラムと言える。次に判定アルゴリズム Demeter を示す。

```
Demeter(G)
 for each vertex v ∈ V[G] do
     for each method m of v do
         m のボディ部から SU(m) を求める
         VS_C(m) ← ∅
     VD_M(v) ← ∅
 for each vertex v ∈ V[G] do
     color[v] ← WHITE
 for each vertex v ∈ V[G] do
     if color[v] = WHITE then DFS(v)
```

```
for each vertex v ∈ V[G] do
    for each method m of v do
        if SU(m) ⊈ TPS(v) then
            VD_M(v) ← VD_M(v) ∪ {m}
            VS_C(m) ← SU(m) − TPS(v)
end_Demeter

DFS(x)
color[x] ← GRAY
MD(x) ← ∅
GAC(x) ← ∅
for each (x, y, p) ∈ RE do
    MD(x) ← MD(x) ∪ {y}
for each (x, y, a) ∈ RE do
    GAC(x) ← GAC(x) ∪ {y}
TPS(x) ← x ∪ MD(x) ∪ GAC(x)
for each (x, y, i) ∈ RE do
    if color[y] = WHITE then
        DFS(y)
    TPS(x) ← TPS(x) ∪ TPS(y)
color[x] ← BLACK
end_DFS
```

5 変換

デメテルの法則に違反したプログラムに対して、デメテルの法則を満たすように変換するいくつかの方法のアイデアが提案されている [3, 4]。ここでは、それらの変換法の基になったリレーメソッド追加法のアルゴリズムを示す。また、新しい変換方法を提案する。

5.1 リレーメソッド追加法 [3]

いくつかのメッセージを並べて書く、つまり、先行のメッセージからの戻りオブジェクトにまた新しいメッセージを送る場合がある。このとき、戻りオブジェクトが依頼者側の優先供給者クラスのオブジェクトでないと、デメテルの法則に違反することになる。この場合、依頼者から供給者へメッセージをリレーする新たなメソッド (リレーメソッドと呼ぶ) を、クラス階層グラフ上の依頼者のクラスから供給者のクラスへの関係辺からなる経路上の各クラスに追加していく変換方法がある。図 2 はリレーメソッド追加法の例を示す。

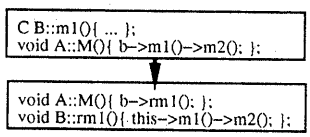


図 2: リレーメソッド追加法の例

図 2 は図 1 のクラス階層をベースとする。クラス A からクラス B にメッセージ $\mathbf{m_1}$ を送り、その返りオブジェクトにまたメソッド $\mathbf{m_2}$ を送る、といったメッセージ転送が行われている。しかし、この例では、クラス C がクラス A のメソッド M の優先供給者クラスではないとすれば、デメテルの法則に違反する。

メソッド M に対して、デメテルの法則を満たすように変換を行う。まず、新しいメソッド $\mathbf{rm_1}$ をクラス B に定義し、メソッド M のメッセージ文をリレーする。そして、メソッド M ではクラス B にメッセージ $\mathbf{rm_1}$ を送るように変更する。もし M で b->m_1()->m_2()->$\cdots$->m_n() となっている場合には、同じ方法でリレーメソッド $\mathbf{rm_2}$ 等を追加して以上の変換を繰り返すと、全てのメソッドはデメテルの法則を満たすようになる。

以上の変換をアルゴリズムで示すために、まず、クラス階層グラフ上に現れるメッセージ転送経路を次のように定義する。

定義 5 メッセージ転送経路 mp
クラス階層グラフ上でメッセージ辺 (ME) で結びつけられた頂点 A と Cとの間の各頂点間に存在する関係辺 (RE) に沿った経路をメッセージ転送経路と呼び、mp $=< v_1, v_2, \cdots, v_k >$と表す ($A = v_1$, $C = v_k$, $v_i = v_{i+1}$の場合もあり得る)。ここで、メッセージ転送経路の長さ |mp| は経路上にある頂点間の関係辺の総数である。

次のアルゴリズム RMI はリレーメソッド追加法を実現している。

アルゴリズム RMI

1. デメテルの法則に違反するメッセージ転送文を含むメソッドを m_1とする。

   $C_1 :: m_1(\{\cdots c_2\text{->}m_2()\text{->}m_3()\text{->}\cdots\text{->}\ m_k();$

   $\cdots\}$

   但し、$i = 1, \cdots, k$に対して、$m_i \in MF(C_i)$。

mp=< $C_1, C_2, \cdots, C_k$ >はメッセージ転送経路であると仮定する。

2. c_2->m_2()->m_3()->$\cdots$->m_k() で->m_i() はデメテルの法則を満たしているが->m_{i+1}() は違反しているようなメッセージ転送 (i が最小なもの) を見つける。

3. C_iに新たなメソッド $\mathbf{rm_i}$を追加し、次のように定義する。

   $rm_i()\{$this->$m_i()$->$m_{i+1}()$->$\cdots$->$m_k()$; $\}$

4. メソッド $\mathbf{m_1}$の c_2->$m_2()$->$m_3()$->$\cdots$->$m_k()$ の部分を c_2->$m_2()$->$\cdots$->$m_{i-1}()$->$rm_i()$ に書き換える。

5. デメテルの法則に違反する各メッセージ転送に対して繰り返し、以上の変換を行う。

上記のアルゴリズムでは、まず与えられたプログラムのクラス階層グラフ Gに対してデメテル法則を満たすか否かをアルゴリズム **Demeter** で判定する。判定アルゴリズムから法則違反と判ったメソッド $m \in VD\_M(C_i)$ とそのメソッドに定義されたメッセージ転送文を用いて、判定アルゴリズムで求めた暫定優先供給者クラスの中でメッセージ転送経路上でクラス C_iと直接関係しているクラス C_jにリレーメソッド rm_jを追加する。

5.2 変換方法の正当性に対する検討

デメテルの法則に違反したプログラムは、アルゴリズム **RMI** によって法則を満たすように変換できるということを検討する。リレーメソッド追加法による変換では、次の定理が成り立つ。

定理 1 デメテルの法則違反プログラムはアルゴリズム **RMI** のステップ *1* の条件を満たす時、**RMI** によって法則を満たすように変換される。

(証明) 法則に違反するメッセージ転送を含む一連のメッセージ転送文 c_2->$m_2()$->$\cdots$->$m_k()$ を、**RMI** が法則に違反しないように変換することを帰納法で示す。

(1) $(C_1, C_2, R) \in RE$、又は $C_1 = C_2$なので->$m_2()$ は違反していない。

(2)->$m_j()$ (但し、$j \leq i$) の部分では違反していないとする。->$m_{i+1}()$ が違反していれば、ステップ 3,4 によって、

c_2->$m_2()$->$\cdots$->$m_{i-1}()$->$rm_i()$

$C_i :: rm_i()\{$ this->$m_i()$->$m_{i+1}()$->$\cdots$->$m_k()$ $\}$

と変換される。

変換前の c_2->$m_2()$->$\cdots$->$m_i()$ が違反していないことから、変換後の c_2->$m_2()$->$\cdots$->$m_{i-1}()$->$rm_i()$ も違反していない (m_i, $rm_i \in MF(C_i)$ であるから)。また、変換後の->$m_i()$ は同じクラスへのメッセージ転送なので違反していない。そして、変換後の->$m_{i+1}()$ はクラス C_iから C_{i+1}への転送なので違反していない。ゆえに、->$m_j()$ (但し、$j \leq i+1$) は違反していないように変換される。

従って、以上の変換を全ての法則違反のメソッドに対して繰り返し行うと、与えられたプログラム $\mathcal{P}$ はデメテルの法則を満たすプログラム $\mathcal{P}'$に変換される。 □

デメテルの法則に違反するプログラムに対して、法則を満たすように変換するリレーメソッド追加法をアルゴリズム **RMI** として定式化した。しかし、次の 2 つの場合、リレーメソッド追加法では変換できないことが明らかになった。図 3 の **Case**

```
void C1::m1(){ ... this->m2()->m3()->...->mk(); ... }
```

(a) Case 1

```
x1=o->m1(); ... x2=x1->m2(); ... x3=x2->m3(); ...
```

(b) Case 2

図 3: 変換アルゴリズムの問題点の例

1 でメソッド $\mathbf{m_1}$中のメッセージ転送文->$m_3()$ が法則に違反し、クラス C_2と C_3との間には関係 RE が存在しない場合 ($(CM(m_2), CM(m_3), R) \notin RE$) がある。この場合はアルゴリズム **RMI** でのように $CM(m_3)$ にリレーメソッドを追加し、$\mathrm{m_1}$のメッセージ転送文を変更しても違反は解決しない。**Case 2** のように一時的に局所変数に退避する場合は、クラス間の関係を変えないとすれば、実行時にメソッドをリレーするオブジェクトを特定できないため、実現できない。つまり、メソッド m_2への呼出時に返すオブジェクトをそのクラスのデータメンバに格納 ($CM(m_3) \in MD(C_2)$) するなどして置けば可能であるが、これはクラス間の関係を変更することになる。

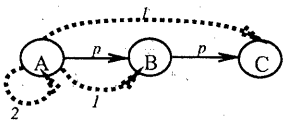
5.3 新しい変換方法の提案

これまで3つのクラス関係を用いてクラス階層を議論したが、特に集約関係の場合にはクラス構造を再構成するほうが良い場合もある。

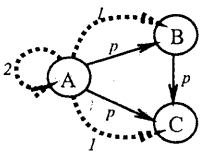
ここでは、法則違反が発生したクラスとそのメッセージ転送経路上のクラスとのクラス階層を変更して、デメテルの法則を満たすように変換する方法として次の2つの方法を提案する。

5.3.1 Moving Information

集約関係のクラス間でデメテルの法則に違反した場合、依頼者クラスと供給者クラスを関係づける方法が考えられる。図4には集約関係のクラス階層構造を変更した例を示す。



(a) Before moving information



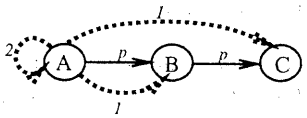
(b) After moving information

図 4: Moving information 法の適用例

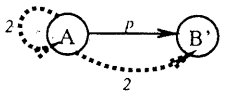
5.3.2 Creating New Class

集約関係では、場合によっては新しいクラスを加えると意味的により明確になることもある。

この考え方に基づき、集約関係の2つのクラスを組み合わせて新しいクラスを作ることも考えられる。図5は集約関係のクラス間でデメテルの法則に違反するメッセージ転送が行われている場合、集約関係の2つのクラスを合併した新しいクラスを加えた変換例を示す。



(a) Before Creating New Class



(b) After Creating New Class

図 5: Creating New Class 法の適用例

6 まとめ

本研究では、クラス間の関係とメッセージ転送について定義を行い、[3, 4] で提案されたデメテルの法則をより具体的に定式化した。また、これらの定義と定式化に基づき、デメテル法則を満たすか否かの判定と、違反したプログラムに対する変換アルゴリズムを提案した。

今後の課題としては、本研究で提案した形式化手法を他の評価基準に適用してみることなどがある。

参考文献

[1] 梁、辻野、都倉:" オブジェクト指向言語のクラスの品質評価について"、情報処理学会ソフトウェア工学研究会 (1991.2).

[2] B.Henderson-Sellers, A Book of Object Oriented Knowledge:Object Oriented Analysis, Design, and Implementation: a new approach to software engineering, Prentice Hall(1992).

[3] Karl J. Lieberherr, Ian Holland, and Arthur J. Riel, "Object-oriented programming:An objective sense of style," *Proc. Object-Oriented Programming Systems, Languages, and Applications Conf.*, ACM, New York, pp.323-334(1988).

[4] Karl J. Lieberherr and Ian Holland, "Assuring Good Style for Object-Oriented Programs", *IEEE Software*, 6(5)(1989).