オブジェクト指向におけるリフレクションの代数的意味論

田原康之[†,☆] 粂野文洋[†,††]
大須賀昭彦[†,☆☆] 本位田真一[†,☆☆]

リフレクション（自己反映計算）あるいはメタレベルアーキテクチャは，コンピュータシステムの機能拡張および動的適応のための技術として，近年注目されている．本論文では，書換え論理に基づく代数的仕様記述言語にリフレクションを導入する手法を提案する．書換え論理では並行オブジェクト指向システムのモデル化が可能であるので，本論文の手法により，メタオブジェクトおよびメタオブジェクトプロトコルなどのオブジェクト指向におけるリフレクションがモデル化できる．さらに，厳密な検証の基盤として，カテゴリ論に基づく代数的意味論を与える．

Algebraic Semantics of Reflection in Object-oriented Systems

Yasuyuki Tahara,[†,☆] Fumihiro Kumeno,[†,††] Akihiko Ohsuga[†,☆☆]
and Shinichi Honiden[†,☆☆]

Reflection or metalevel architecture is receiving attention in recent years, since they are technologies for functional extension and dynamic adaptation of computer systems. In this paper, we propose a technique to introduce reflection into a kind of algebraic specification languages called rewriting logic. Since rewriting logic can model concurrent object-oriented systems, our technique enables modeling of reflective object-oriented systems including metaobjects and metaobject protocols. Moreover, algebraic semantics is given based on a category-theoretic model.

1. はじめに

リフレクション（自己反映計算）あるいはメタレベルアーキテクチャは，コンピュータシステムの機能拡張および動的適応のための技術として，近年注目されている．一般に，データや手続きなどの，ベースレベルの対象と呼ばれるようなシステムの構成要素を，プログラミング言語あるいは仕様記述言語で記述する場合，内部状態や継続といった，メタレベルの対象と呼ばれるような言語の実行モデルに属する概念は，メタ言語と呼ばれる他の言語で記述される．一方，リフレクションにおいては，メタレベルの対象をもベースレベルの対象と同じ言語で記述することが可能である．しかも，メタレベル記述の操作はベースレベルの実行モデルの変さらに反映され，その結果システム自体の実行モデルが変更されることになる．そのため，このような機構はリフレクション（自己反映）と呼ばれている．

特にオブジェクト指向システムにおいては，たとえばCLOS[5]で実装されているように，リフレクションはメタオブジェクトとメタオブジェクトプロトコルによって実現されることが多い．メタオブジェクトは，メタレベルのデータと操作をカプセル化したオブジェクトで，メタオブジェクトプロトコルによりその（メタ）メソッドが起動され，その結果はベースレベルの状態の変化へと反映される．このように，メタオブジェクトの利用者は，扱いが困難なメタレベル操作の実現の詳細を知る必要がないので，上述のメタレベルカプセル化の概念は有効である．

リフレクションをコンピュータシステムの機能拡張や動的適応に適用するために，インタプリタレベルの概念をメタレベルの対象として扱おうとする研究者は多い．一方，ソフトウェア工学の分野においては，プログラムや仕様記述そのものの動的変更により，シ

† 情報処理振興事業協会（IPA）
Information-Technology Promotion Agency (IPA)
☆ 現在，株式会社東芝より英国City大学に赴任中
Presently with Toshiba Corporation and visiting City University, UK
†† 株式会社三菱総合研究所より出向中
Also with MRI Corporation
☆☆ 現在，株式会社東芝
Presently with Toshiba Corporation

ステムの稼働環境の変化に対応したいという要求に対し，リフレクションを適用することが試みられている[3),6),7),16),17)]．しかしその場合には，変更が行われた後でも，ユーザの要求との整合性が保たれることを保証する必要がある．したがって，記述の動的変更に対応できるような検証手法を確立することが重要となる．

本論文では，OBJ 系の言語に代表される代数的仕様記述言語にリフレクションを導入する手法を提案する．さらに，厳密な検証の基盤として，表示的意味論を与える．そのために，次のような課題を設定し，それらを解決する手法を検討する．

- 従来代数的仕様記述言語とリフレクションを結び付ける研究は行われている[8)]．しかし，いずれもリフレクションの計算モデルを明らかにすることを目的としており，記述そのものの変更やその検証といったことを可能にするものではない．

  一方，本論文では，近年提案された**書換え論理**[9)~12)]と呼ばれる形式的体系を採用する．書換え論理に基づく代数的仕様記述によれば，等式論理の場合のようなシステムの静的な性質だけではなく，動的な性質をも抽象的に記述できるので，記述の動的な変更を扱える．

  書換え論理は，並行オブジェクト指向システムのモデル化も可能である．したがって，メタレベルカプセル化も定式化できる．さらに，やはり等式論理と同様に，代数的モデル（詳しくはカテゴリ論的モデル）に基づいた表示的意味論が与えられる．
- また，リフレクションの表示的意味論に関する研究も従来より行われている[1),2),13),15),18)]．しかしこれらは，変数の割当てや継続といった計算の状態に対する，(参照や変更などの) 操作の意味しか与えていない．したがって，本論文で取り扱っているような記述の変更の意味を与えることが可能でない．

  また，表示的意味論においては，構文領域と意味領域とを厳密に区別し，前者から後者への意味写像により意味論を与える必要がある．しかし，記述の変更は構文領域に反映されるため，このような枠組みによる意味記述は困難であった．

  一方，本論文では，上記 2 種類の領域を，カテゴリ論的概念を用いて，個別にかつ統一的に定義する．さらに，ベースレベルとメタレベルの関係を，内部カテゴリとその外部化という概念により定式化する．これにより，構文領域と意味領域とを区別し，両者の間の写像を定義することができる．

本論文の構成は次のとおりである．2 章では，書換え論理とその意味論，および書換え論理によるオブジェクト指向のモデル化について簡単にまとめる．3 章では，メタオブジェクトプロトコルの例を通じて，オブジェクト指向におけるリフレクションについて述べる．4 章では，**RL/R**，すなわちリフレクティブ書換え論理について説明し，その表示的意味論について論ずる．5 章では，RL/R によるメタオブジェクトプロトコルのモデル化の例を示す．6 章では，関連研究について述べる．最後に 7 章において結論を示す．

2. 書換え論理によるオブジェクト指向のモデル化

本章では，書換え論理とその意味論，および Maude 言語[12)]として実現されている並行オブジェクト指向のモデル化の概略を述べる．書換え論理は，最初並列・並行計算のモデルとしての並行項書換えのための論理として提案された．その後，並行オブジェクトのための論理の枠組みとして有効であることが明らかになってきている．

書換え論理は書換え理論と推論規則から構成される．これら理論と規則から，項やそれらの間の等号関係，および等号関係による項の同値類の間の書換え関係が導出される．

定義 1 （ラベル付き）書換え理論 $\mathcal{R}$ とは，次を満たす 6 つ組 (S, Σ, V, E, L, R) である．ただし，S はソートの集合，Σ は S 内のソートで型が付けられた関数記号の集合，V はやはり S で型付けられた変数記号の集合，E は S で型付けられた等式の集合，L はラベルの集合，R は S で型付けられ，L でラベル付けされた書換え規則の集合である．ここで，等式および書換え規則は一般に条件付きである．すなわち，$\Sigma \cup V$ から構成された項の集合を $\mathrm{Term}(\Sigma, V)$ と書くと，

- $E \subseteq \bigcup_{n=0}^{\infty} (\mathrm{Term}(\Sigma, V)^2) \times (\mathrm{Term}(\Sigma, V)^2)^n$
  なお，$((t_1, t_2), ((t_3, t_4) \ldots (t_{n-1}, t_n)) \in E$ のとき，
  $$t_1 = t_2 \text{ if } t_3 = t_4, \ldots, t_{n-1} = t_n$$
  と書く．
- $R \subseteq \bigcup_{n=0}^{\infty} L \times (\mathrm{Term}(\Sigma, V)^2) \times (\mathrm{Term}(\Sigma, V)^2)^n$
  なお，$(l, (t_1, t_2), ((t_3, t_4) \ldots (t_{n-1}, t_n)) \in R$ のとき，
  $$l : t_1 \to t_2 \text{ if } t_3 \to t_4, \ldots, t_{n-1} \to t_n$$
  と書く．

書換え関係を導出するための推論規則には，次の4つがある．なお，ここで $[t]$ とあるのは，項 t を含む，等号関係に関する同値類である．

反射性（Reflexivity） 任意の $t \in \mathrm{Term}(\Sigma, \mathrm{V})$ に対し，

$$\frac{}{[t] \longrightarrow [t]}$$

適合性（Congruence） 任意の $f \in \Sigma$ に対し，

$$\frac{[t_1] \longrightarrow [t'_1] \ldots [t_n] \longrightarrow [t'_n]}{[f(t_1, \ldots, t_n)] \longrightarrow [f(t'_1, \ldots t'_n)]}$$

置換（Replacement） 任意の書換え規則 $r : t(x_1, \ldots, x_n) \longrightarrow t'(x_1, \ldots, x_n)$ に対し，

$$\frac{[w_1] \longrightarrow [w'_1] \ldots [w_n] \longrightarrow [w'_n]}{[t(\bar{w}/\bar{x})] \longrightarrow [t'(\bar{w'}/\bar{x})]}$$

推移性（Transitivity）

$$\frac{[t_1] \longrightarrow [t_2] \quad [t_2] \longrightarrow [t_3]}{[t_1] \longrightarrow [t_3]}$$

ではここで，書換え論理による並行システムの記述について，例をあげて説明する．ここで取りあげる例は，Meseguer により提案された，書換え論理に基づく仕様記述・プログラミング言語 **Maude** による，並行オブジェクト指向システムの記述例である．

Maude においては，並行オブジェクトを次のような項により記述する．

$$\langle O : C \,|\, a_1 : v_1, \ldots, a_n : v_n \rangle$$

ただし，$\langle \_|\_ \rangle$ は並行オブジェクトのための関数記号，O はオブジェクトの識別子（名前），C はそのクラス，a_i 達はオブジェクトの属性名，v_i 達は対応する属性値である．たとえば，$\langle B : \mathrm{Buff} \,|\, \mathrm{contents} : Q \rangle$ は，名前が B で，内容が Q であるバッファオブジェクトを表す．

次に並行オブジェクト指向システムの状態は，**コンフィギュレーション**と呼ばれる項で記述する．コンフィギュレーションとは，並行オブジェクトと未処理メッセージを表す項達を，結合的かつ可換で単位元（ϕ で表す）を持つ☆，目に見えない演算子で結ぶことにより，それらの項を要素とする（多重）集合を表現している項である．たとえば，$(\mathrm{put}\ 1\ \mathrm{in}\ b1)\langle b1 : \mathrm{Buff} \,|\, \mathrm{contents} : \mathrm{null} \rangle$ はコンフィギュレーションである．項がコンフィギュレーションを表すことは，`Configuration` というソートを与えることによって示す．

そして並行オブジェクト指向システムの挙動は，次のようなコンフィギュレーションに対する書換え規則で記述する．

☆ Associative and Commutative with an Identity, ACI と略す．

```
omod BUFF[X :: TRIV] is
    protecting LIST[X] .
    class Buff .
    att contents : Buff -> List [hidden]
    msg put_in_ : Elt Id.Buff -> Msg.Buff .
    msg getfrom_replyto_ : Id.Buff OId -> Msg.Buff .
    msg elt-in_is_to_ : Id.Buff Elt OId -> Msg .
    var B : Id.Buff .
    var I : OId .
    var E : Elt .
    var Q : List .
    rl (put E in B) <B : Buff | contents : Q>
       => <B : Buff | contents : E Q>
    rl (getfrom B reply to I)
       <B : Buff | contents : E Q>
       => <B : Buff | contents : Q> (elt-in B is E to I).
endom
```

図1 Maude 言語によるバッファクラスの記述
Fig. 1 Description of Buffer class in Maude language.

$$M_1 \ldots M_n \langle O_1 : C_1 | attrs_1 \rangle \ldots \langle O_m : C_m | attrs_m \rangle \longrightarrow \langle O_{i_1} : C_{i_1} | attrs'_{i_1} \rangle \ldots \langle O_{i_k} : C_{i_k} | attrs'_{i_k} \rangle \langle Q_1 : D_1 | attrs''_1 \rangle \ldots \langle Q_p : D_p | attrs''_p \rangle M'_1 \ldots M'_q$$

ただし，

- $M_i\ (i = 1, \ldots, n)$: メッセージ
- $\langle O_i : C_i | attrs_i \rangle\ (i = 1, \ldots, m)$: 動作の前のオブジェクトの状態
- $O_{i_j}\ (j = 1, \ldots, k)$: O_i のうち，動作後も消滅せずに残ったオブジェクト
- $attrs'_{i_j}\ (j = 1, \ldots, k)$: O_{i_j} の動作後の属性
- $Q_i\ (i = 1, \ldots, p)$: 動作後に生成したオブジェクト
- $M'_i\ (i = 1, \ldots, q)$: 動作時に送信されるメッセージ

たとえば，

$$(put\ E\ in\ B)\langle B : \mathrm{Buff} \,|\, \mathrm{contents} : Q \rangle \longrightarrow \langle B : \mathrm{Buff} \,|\, \mathrm{contents} : E\ Q \rangle$$

は，データ E がバッファ B に格納されるという挙動を表す書換え規則である．

以上のまとめとして，Maude 言語によるバッファクラスの記述例の全体を**図1** にあげる．

書換え論理にはカテゴリ論に基づく意味論が与えられる．以下にその概要を示す．

定義2 書換え理論 $\mathcal{R} = (S, \Sigma, V, E, L, R)$ に対し，$\mathcal{R}$-系（$\mathcal{R}$-system）とは，次のような構造を持つ 2-カテゴリ $\mathcal{C}$ のことである．

- (S, Σ, V, E)-構造である．すなわち，任意の $f \in \Sigma_{\bar{s}s}$，つまり関数記号 $f : \bar{s} \to s$ に対し，関手 $[\![f]\!]_{\mathcal{C}} : [\![s_1]\!]_{\mathcal{C}} \times \ldots \times [\![s_n]\!]_{\mathcal{C}} (= [\![\bar{s}]\!]_{\mathcal{C}}) \to [\![s]\!]_{\mathcal{C}}$ ☆☆が

☆☆ $[\![]\!]_{\mathcal{C}}$ は，$\mathcal{R}_0$-システムとしての $\mathcal{C}$ への意味関数の値を表す．なお，ソート s に対し，$[\![s]\!]_{\mathcal{C}}$ は，ソート s の項を対象とし，項の間の書換えを射とするカテゴリである．

対応し，さらに E 内の等式が成立する．

- $\mathcal{R}$ の各書換え規則

  $r : t(\bar{x}) \to t'(\bar{x})$

  if $u_1(\bar{x}) \to v_1(\bar{x}) \wedge \ldots \wedge u_k(\bar{x}) \to v_k(\bar{x})$

  に対し，自然変換

  $r : J_{\mathcal{C}} * [\![t]\!]_{\mathcal{C}} \Rightarrow J_{\mathcal{C}} * [\![t']\!]_{\mathcal{C}}$

  が対応する．ただし，

  $J_{\mathcal{C}} : \mathrm{Subeq}(([\![u_j]\!]_{\mathcal{C}}, [\![v_j]\!]_{\mathcal{C}})_{1 \le j \le k}) \to [\![\bar{s}]\!]_{\mathcal{C}}$（$\bar{s}$ は $\bar{x}$ のソートの列）はサブイコライザ（sub-equalizer[10]）である．

3. オブジェクト指向におけるリフレクション

リフレクション（自己反映計算）あるいはメタレベルアーキテクチャは，コンピュータシステムの機能拡張および動的適応のための技術として，近年注目されている．一般に，データや手続きなどの，ベースレベルの対象と呼ばれるようなシステムの構成要素を，プログラミング言語あるいは仕様記述言語で記述する場合，内部状態や継続といった，メタレベルの対象と呼ばれるような言語の実行モデルに属する概念は，やはりメタ言語と呼ばれる他の言語で記述される．一方，リフレクションにおいては，メタレベルの対象をもベースレベルの対象と同じ言語で記述することが可能である．しかも，メタレベル記述の操作はベースレベルの実行モデルの変更に反映され，その結果システム自体の実行モデルが変更されることになる．そのため，このような機構はリフレクション（自己反映）と呼ばれている．

特にオブジェクト指向システムにおいては，たとえばCLOS[5]で実装されているように，リフレクションはメタオブジェクトとメタオブジェクトプロトコルによって実現されることが多い．メタオブジェクトはメタレベルのデータと操作をカプセル化したオブジェクトで，メタオブジェクトプロトコルによりその（メタ）メソッドが起動され，その結果はベースレベルの状態の変化へと反映される．このように，メタオブジェクトの利用者は，扱いが困難なメタレベル操作の実現の詳細を知る必要がないので，上述のメタレベルカプセル化の概念は有効である．

ここでメタプロトコルの例として，前述のバッファオブジェクトにおける "addMethod" メタプロトコルを取りあげて説明する．たとえば，バッファオブジェクトがメタメッセージ (**addMethod get2from ...**) を受け取ることにより，バッファオブジェクトから同時に2つの要素を取り出す "get2from" メソッドが追加される場合を考える．このような状況をモデル化するためには，次の仕様を元のバッファクラスの仕様に追加するような動作を考える必要がある．

```
msg get2from_replyto_ : Id.Buff OId -> Msg.Buff .
msg elt2-in_is_and_to_ :
    Id.Buff Elt Elt OId -> Msg.Buff .
var E1 E2 : Elt .
rl (get2from B replyto I) <B : Buff | contents: E1 E2 Q>
   => <B : Buff | contents: Q>
      (elt2-in B is E1 and E2 to I) .
```

しかし，"addMethod" メタプロトコルを書換え論理でモデル化するにあたっては，次のような問題がある．

- 書換え論理による通常のオブジェクトのモデル化においては，メッセージ処理の結果として，項で表されたシステム状態の変化しか記述することができない．したがって，"addMethod" メッセージの処理によって起こる書換え理論の変化を，このような枠組みで記述することは困難である．
- "get2from" メソッドの仕様を "addMethod" メッセージの引数中で示すためには，ソートや変数を表す記号をも，データとして扱う必要がある．しかしこのような取扱いは，仕様記述の意味に対し矛盾や混乱を引き起こす元となる．

4. リフレクティブ書換え論理 RL/R

前章で述べた問題を解決するため，本章ではリフレクティブ書換え論理 RL/R を提案する．その特徴は次のとおりである．

- RL/R では，メタ表現の記法が与えられる．これにより，メタレベルの概念をデータとして扱うことが可能となる．なお，通常のリフレクションの定式化と同様に，メタレベルの対象をデータに変換する機構として，レイファイアと呼ばれる作用素を用いるが，さらにレイファイアの代数仕様が与えられる．
- RL/R は，リフレクションを表すための推論規則と，カテゴリ論に基づく意味論が与えられるので，書換え理論の変更をモデル化することができる．これらの手法により，メタレベルの挙動をベースレベルの変更と関係付けることが可能となる．

4.1 RL/R の構成要素

RL/R の記述の構成要素は次のとおりである．

- ベースレベルの記号の可算無限集合 $\mathcal{S}$ が与えられる．ただし，$\mathcal{S}$ は，後述のベースレベルを扱うプリミティブをすべて含んでいる必要がある．また，ベースレベルの記号 π に対し，そのメタ表記を $'\pi$ で表し，後述の **Sym** ソートの定数記号と

する．なお，メタ表記の集合を $\mathcal{S}_+$ で表し，ベースレベルの記号との和集合を，$\mathcal{S}_* = \mathcal{S}_+ \cup \mathcal{S}$ で表す．

- 仕様は，書換え理論 $\mathcal{R} = (S, \Sigma, V, E, L, R)$ により与えられる．ただし
  - $S \subseteq \mathcal{S}$.
  - $\Sigma = \{\Sigma_{\bar{s}s} | \bar{s} \in S^{*}$☆$, s \in S\}$ は次を満たす．
    - $\bigcup_{\bar{s} \in S^*, s \in S} \Sigma_{\bar{s}s} \subseteq \mathcal{S}_*$
    - $\mathcal{S}_+ \subseteq \Sigma_{\varepsilon \mathbf{Sym}}$ ただし，**Sym** は，後述のベースレベルの記号のメタ表現のソート．
- ベースレベルを扱うプリミティブの仕様を表す書換え理論を $\mathcal{R}_0$ で表す．その主要部分は以下のとおりである．

```
sort Sym Term State Obj Op Var Eq Rl Spec .
sort SymList ... % 各ソートに対するリストソート
op termCon : Sym -> Term .
op id : Obj -> Term .
        % カテゴリの構成要素を表す
        % 他に _~_, p1, p2, pair, unit, (), objCon,
        %_x_, dom, cod, _o_, statePair.
op spec : SymList OpList VarList EqList RlList
          -> Spec .
op opCon : Sym SymList Sym -> Op .
        % 仕様の構成要素を表す．他に varCon, eqCon,
        % rlCon.
op reifySym : Sym -> Term .
op reify : Term -> Term .
ops reifySpec : Spec -> Term . % 以上，レイファイア．
vars T T1 T2 : Term . % その他，変数記号の宣言

eq stateCon(T1 ~ T2, Sp)
   = stateCon(T1, Sp) o stateCon(T2, Sp) .
eq stateCon(pair(T1, T2), Sp)
   = statePair(stateCon(T1, Sp), stateCon(T2, Sp)) .
eq cod(stateCon((), Sp)) = 'unit .
eq reifyObj(objCon(S)) = reifySym(S) .
eq reifyObj(O1 x O2)
   = '_x_ ~ pair(reifyObj(O1), reifyObj(O2)) .
eq reify(termCon(S)) = reifySym(S) .
eq reify(p1(O1, O2))
    = 'p1 ~ pair(reifyObj(O1), reifyObj(O2)) .
       % その他，reify の仕様が続く

rl mp : stateCon(T1, spec(S1, O1, V1, E1, R1))
        => stateCon(T2, spec(S1, O1, V1, E1, R1))
  if (member(rlCon(R, [T1, T2 | RRest]), R1)
      and rewAll(RRest, spec(S1, O1, V1, E1, R1)))
          = true .
```

なお，最後の書換え規則において，if 部に出現する変数 **R**, **RRest** については，「変数に代入すると if 部を満足するものが存在すれば」という意味になるので，意味のある変数であることに注意する．また，**rlCon**, **RewAll** はそれぞれ書換え規則のメタ表現，および第一引数で示されるリストの要素となる項の対において，いずれも 1 つ目の項が 2 つ目の項に書換え可能であることを示す．

- 項 t と書換え理論 $\mathcal{R} = (S, \Sigma, V, E, L, R)$ に対し，それらのメタ表現である reification 項 $\uparrow(t, \mathcal{R})$, $\uparrow \mathcal{R}$ が，次のように定義される．
  - $s \in \Sigma \cap \mathcal{S}$ のとき $\uparrow(s, \mathcal{R}) = \mathbf{termCon}('s)$.
  - $s \in \Sigma \cap \mathcal{S}_+$ のとき $\uparrow(s, \mathcal{R}) = \mathbf{reifySym}(s)$.
  - $(v_i, s_i) \in V$ のとき，
    $\uparrow(v_i, \mathcal{R})$
    $= \mathbf{p1}(s_i \ \mathbf{x}\ (s_{i+1} \ \mathbf{x}\ \ldots \ \mathbf{x}\ s_n))$
    $\tilde{}\ \mathbf{p2}(s_{i-1} \ \mathbf{x}\ (s_i \ \mathbf{x}\ \ldots \ \mathbf{x}\ s_n))$
    $\tilde{}\ \ldots \tilde{}\ \mathbf{p2}(s_1 \ \mathbf{x}\ (s_2 \ \mathbf{x}\ \ldots \ \mathbf{x}\ s_n))$
  - $\uparrow(f(t), \mathcal{R}) = (\uparrow(f, \mathcal{R}))$ o $\uparrow(t, \mathcal{R})$
  - $\uparrow(f(t_1, \ldots, t_n), \mathcal{R}) = (\uparrow(f, \mathcal{R}))$ o **pair** $(\uparrow(t_1, \mathcal{R}), \mathbf{pair}(\ldots, \mathbf{pair}(\uparrow(t_{n-1}, \mathcal{R}), \uparrow(t_n, \mathcal{R}))\ldots)$
  - $\uparrow(S, \Sigma, V, E, L, R) = \mathbf{spec}(\uparrow S, \ldots, \uparrow R)$
    ただし，$\uparrow S$ などは，各集合の要素のメタ表現のリストである．たとえば，
    $t_1 = t_2$ if $t_3 = t_4, \ldots, t_{n_1 - 1} = t_{n_1}$;
    $\ldots; t_{n_2} = t_{n_2+1}$ if $t_{n_2+2} = t_{n_2+3}, \ldots,$
    $t_{n_3-1} = t_{n_3}$ in E
    のとき，
    $\uparrow E = [\mathbf{eqCon}([\uparrow(t_1, \mathcal{R}), \ldots, \uparrow(t_{n_1}, \mathcal{R})]),$
    $\ldots, \mathbf{eqCon}([\uparrow(t_{n_2}, \mathcal{R}), \ldots, \uparrow(t_{n_3}, \mathcal{R})])]$

4.2 推論規則

RL/R においては，システム状態を表す項の等式関係による同値類と，仕様を表す書換え理論の対の間の遷移関係により，システムの挙動を記述する．このような等式関係および遷移関係は，次のような推論規則により与えられる

- システム状態は，項の同値類と書換え理論の対 $([t]_{\mathcal{R}}$☆☆$, \mathcal{R})$ で表す．
- 等式関係に対する推論規則は以下のとおり．
  - 通常の等式論理の推論規則により，$\mathcal{R}$ において $t = t'$ が導出されるならば，$t = t'$ in $\mathcal{R}$.
  - （等式のリフレクション）
    $t = t'$ in $\mathcal{R} \Leftrightarrow \mathbf{stateCon}(\uparrow(t, \mathcal{R}), \uparrow \mathcal{R}) = \mathbf{stateCon}(\uparrow(t', \mathcal{R}), \uparrow \mathcal{R})$ in $\mathcal{R}$
- $\mathcal{R} = \mathcal{R}'$ のとき，$(t, \mathcal{R}) \to (t', \mathcal{R}')$ は，$t \to t'$ in $\mathcal{R}$ とも書く．

☆ S_* とは異なり，S の要素から成る有限列（空列を含む）の集合を表す．

☆☆ 等式関係は書換え理論に依存するため，添字を付ける．

- 状態遷移関係に対する推論規則は以下のとおり.
  - 通常の書換え論理の推論規則により，$\mathcal{R}$ において $[t]_{\mathcal{R}} \rightarrow [t']_{\mathcal{R}}$ が導出されるならば，$[t]_{\mathcal{R}} \rightarrow [t']_{\mathcal{R}}$ in $\mathcal{R}$.
  - （書換えのリフレクション）
    $([t]_{\mathcal{R}}, \mathcal{R}) \rightarrow ([t']_{\mathcal{R}}, \mathcal{R}') \Leftrightarrow [\mathtt{stateCon}(\uparrow (t, \mathcal{R}), \uparrow \mathcal{R})]_{\mathcal{R}} \rightarrow [\mathtt{stateCon}(\uparrow (t', \mathcal{R}'), \uparrow \mathcal{R}')]_{\mathcal{R}}$ in $\mathcal{R}$

4.3 代数的意味論

本節では，RL/Rの代数的意味論を検討する．これまで述べてきたように，RL/Rは書換え論理を基盤としているので，書換え論理と同様に，カテゴリ論に基づく代数的意味論を検討することができる．さらにRL/Rにおいてはメタレベルアーキテクチャを表すための特徴があるので，これを意味論に反映させている．

4.3.1 意味領域

RL/Rの意味領域についても，やはり書換え論理と同様に，(2-) カテゴリとして与えられる．ただし，RL/Rにおけるメタレベルアーキテクチャの特徴を次のように反映している．

- 前述のベースレベルを扱うためのプリミティブの意味に対応する要素を有する．
- リフレクションの推論規則に対応して，ベースとメタそれぞれの意味領域の間に同型関手が与えられる．

定義3 意味領域 $\mathcal{C}$ は，$\mathcal{R}_0$-システム☆であって，カルテジアンカテゴリとしての同型関手 $\sigma_{\mathcal{C}} : \mathcal{C} \rightarrow \mathcal{C}^r$, $\delta_{\mathcal{C}} : \mathcal{C}^r \rightarrow \mathcal{C}$ を持つもの．ただし，$\mathcal{C}^r = (C_0^r, C_1^r, C_2^r, \mathrm{dom}^r, \mathrm{cod}^r, 2\mathrm{dom}^r, 2\mathrm{cod}^r)$ は次のように定義されるカルテジアン2-カテゴリで，$\sigma_{\mathcal{C}}$ は下記の条件を満たす．

- $C_0^r = [\![\mathtt{Obj}]\!]_{\mathcal{C}0}$
- $C_1^r = \{f \in [\![\mathtt{State}]\!]_{\mathcal{C}0} | [\![\mathtt{state?}]\!]_{\mathcal{C}}(f) = [\![\mathtt{true}]\!]_{\mathcal{C}}\}$
- $\mathrm{dom}(f) = [\![\mathtt{dom}]\!]_{\mathcal{C}}(f), g \circ f = [\![\_\mathtt{o}\_]\!]_{\mathcal{C}}(g, f)$ など，カルテジアンカテゴリとしての構成要素は，$\mathcal{R}_0$ の対応する要素により与えられる．
- $C_2^r = \{\tau \in [\![\mathtt{State}]\!]_{\mathcal{C}1} | \mathrm{dom}_{[\![\mathtt{State}]\!]_{\mathcal{C}}}(\tau), \mathrm{cod}_{[\![\mathtt{State}]\!]_{\mathcal{C}}}(\tau) \in C_1^r\}$
- $2\mathrm{dom}(\tau) = \mathrm{dom}_{[\![\mathtt{State}]\!]_{\mathcal{C}}}(\tau)$, $2\mathrm{cod}(\tau) = \mathrm{cod}_{[\![\mathtt{State}]\!]_{\mathcal{C}}}(\tau)$
- $\mathcal{R}_0$ のソート s に対し，$\sigma_{\mathcal{C}}([\![s]\!]_{\mathcal{C}}) = [\![\mathtt{objCon}('s)]\!]_{\mathcal{C}}$
- 任意の $f : [\![\mathtt{unit}]\!]_{\mathcal{C}} \rightarrow [\![\mathtt{State}]\!]_{\mathcal{C}}, r : [\![\mathtt{unit}]\!]_{\mathcal{C}} \rightarrow [\![\mathtt{Spec}]\!]_{\mathcal{C}}$ に対し，

$\sigma_{\mathcal{C}}([\![\mathtt{stateCon}]\!]_{\mathcal{C}} \circ \langle f, r \rangle)$
$= [\![\mathtt{stateCon}]\!]_{\mathcal{C}}([\![\mathtt{'\_\tilde{}\_}]\!]_{\mathcal{C}}([\![\mathtt{'stateCon}]\!]_{\mathcal{C}}, [\![\mathtt{pair}]\!]_{\mathcal{C}}([\![\mathtt{reify}]\!]_{\mathcal{C}}(f([\!['()]\!]_{\mathcal{C}})), [\![\mathtt{reifySpec}]\!]_{\mathcal{C}}(r([\!['()]\!]_{\mathcal{C}})))), r([\!['()]\!]_{\mathcal{C}}))$

このように，意味領域をカテゴリとして定義することにより，RL/Rにおけるメタな操作（項の部分的な操作や書換え理論の動的な変更）を代数的に定式化できる．さらに，メタな操作がベースレベルに反映される，ということを，関手の概念により，やはり代数的に扱うことができる．

4.3.2 健全性と完全性

本意味論の健全性は，次の事実により示される．

補題4 $t = t'$ in $\mathcal{R} \Rightarrow [\![\mathtt{stateCon}(\uparrow (t, \mathcal{R}), \uparrow \mathcal{R})]\!]_{\mathcal{C}} = [\![\mathtt{stateCon}(\uparrow (t', \mathcal{R}), \uparrow \mathcal{R})]\!]_{\mathcal{C}}$ in $\mathcal{C}$

定理5 $[t]_{\mathcal{R}} \rightarrow [t']_{\mathcal{R}}$ in $\mathcal{R} \Rightarrow \exists r : [\![\mathtt{stateCon}(\uparrow (t, \mathcal{R}), \uparrow \mathcal{R})]\!]_{\mathcal{C}} \rightarrow [\![\mathtt{stateCon}(\uparrow (t', \mathcal{R}), \uparrow \mathcal{R})]\!]_{\mathcal{C}}$ in $\mathcal{C}$

系6 $(t, \mathcal{R}) \rightarrow (t', \mathcal{R}') \Rightarrow \exists r : [\![\mathtt{stateCon}(\uparrow (t, \mathcal{R}), \uparrow \mathcal{R})]\!]_{\mathcal{C}} \rightarrow [\![\mathtt{stateCon}(\uparrow (t', \mathcal{R}'), \uparrow \mathcal{R})]\!]_{\mathcal{C}}$ in $\mathcal{C}$

さらに，項モデルを構成することにより始モデルを与えることができる．始モデルに対しては，上記の各事実の逆を示すことができるので，これにより完全性が示される．

5. RL/Rにおけるメタオブジェクトプロトコル仕様の例

本章では，RL/Rにおけるメタオブジェクトプロトコルの仕様記述を，“addMethod” メッセージを例として説明する．

RL/Rでは，システム状態を，項の同値類と書換え理論の対で表す．通常の書換え論理と同様に，コンフィギュレーションを表すのに用いる項のソートは `Configuration` である．システムの挙動の仕様は次のとおりである．

- ベースレベルの挙動の仕様は，通常の書換え論理と同様に，コンフィギュレーションを表す項の書換え規則で表す．
- メタレベルの挙動の仕様は，`State` ソートの項の書換え規則で表す．書換え理論 $\mathcal{R}$ において，項 $\mathtt{stateCon}(\uparrow (t, \mathcal{R}), \uparrow \mathcal{R})$ が項 $\mathtt{stateCon}(\uparrow (t', \mathcal{R}'), \uparrow \mathcal{R}')$ に書き換わる場合，書換えのリフレクション規則により，対 $([t]_{\mathcal{R}}, \mathcal{R})$ で表される状態から，対 $([t']_{\mathcal{R}}, \mathcal{R}')$ で表される状態へと遷移するものと考える．

以下に，3章であげた “addMethod” メタプロトコルの仕様記述の例を示す．

“addMethod” メッセージは `Spec` ソートの項を1つ

☆ 書換え論理の意味論において，$\mathcal{R}_0$ に対する意味領域となるカテゴリ[10]．

引数としてとる．このメッセージを受け取ると，引数を構成するリストが，現在の仕様のメタ表現の項を構成するリストに追加（append）される．したがって，“addMethod” の実行に対応する挙動を表すことになる．“addMethod” の形式的仕様は次のとおりである．

```
op (addMethod_) : Spec -> Msg .
var Bf : Buff .
vars Sl SltoAdd : SymList .
vars Ol OltoAdd : OpList .
vars Vl VltoAdd : VarList .
vars El EltoAdd : EqList .
vars Rl RltoAdd : RlList .
var Rest : Configuration .

rl stateCon(^(Bf (addMethod
                  spec(SltoAdd, OltoAdd, VltoAdd,
                       EltoAdd, RltoAdd)) Rest),
            spec(Sl, Ol, Vl, El, Rl))
   => stateCon(^(Bf Rest),
               spec(append(Sl, SltoAdd),
                    append(Ol, OltoAdd),
                    append(Vl, VltoAdd),
                    append(El, EltoAdd),
                    append(Rl, RltoAdd)))
  if member(varCon('Bf, 'Buff), Vl) = true
     and member(varCon('Sl, 'SymList), Vl) = true
     and ... and member(varCon('Rest, 'Configuration),
                        Vl) = true .
```

ただし，^ は，レイファイアを表す超越的な（つまり，メタレベルよりもさらに上位の）記号である．以下，バッファクラスの仕様と上記の仕様を合わせた仕様を **sp** で表す．

次に，3 章で示した “get2from” メソッドの仕様のメタ表現は次のようになる．

```
spec([], [opCon('get2from_replyto_,
                ['Id.Buff, 'OId], 'Msg.Buff),
          opCon('elt2-in_is_and_to_,
                ['Id.Buff, 'Elt, 'Elt, 'OId],
                 'Msg.Buff)],
     [varCon('E1, 'Elt), varCon('E2, 'Elt)],
     [], [rlCon('get2fromMethod,
                [^(get2from B replyto I)
                 <B : Buff | contents: E1 E2 Q>,
                 ^<B : Buff | contents: Q>
                 (elt2-in B is E1 and E2 to I)])])
```

以下，このメタ表現を **sp-to-add** と書く．

これらの仕様記述により，バッファオブジェクト **aBuff** は，メッセージ **(addMethod sp-to-add)** を受け取ることにより意図どおりの挙動を行うことが，次のようにして分かる．

(1) このような状況において初期のコンフィギュレーションは，仕様 **sp** において，次のような項で表される．

```
{aBuff | contents: aList}
(addMethod sp-to-add) rest
```

ただし，**rest** は，コンフィギュレーションの残りの部分を表す．

(2) 上述のコンフィギュレーションと初期の仕様との対のメタ表現は次のようになる．

```
stateCon(^({aBuff | contents: aList}
          (addMethod sp-to-add) rest), ^sp)
```

すると，このメタ表現が **sp** の書換え規則の左辺とマッチすることが，次のようにして分かる．すなわち，このメタ表現は，4 章で取りあげたベースレベルを扱うプリミティブの仕様 $\mathcal{R}_0$ における，レイファイアの仕様により，次の項と等しくなるからである．

```
stateCon('__ ~ pair(^{aBuff | contents: aList},
         '__ ~ pair(^(addMethod sp-to-add),
                    ^rest)), ^sp)
```

ただし '__ は，コンフィギュレーションを構成するための，見えない演算子のメタ表現である．したがって，変数の対応は次のようになる．

```
Bf : ^{aBuff | contents: aList}
SltoAdd, OltoAdd, ... : ^sp-to-add の構成要素
Rest : rest
Sl, Ol, ... : ^sp の構成要素
```

(3) 書換え規則により，上記の項は次のように書き換えられる．

```
stateCon(^(Bf Rest), spec(append(Sl, SltoAdd),
                          append(Ol, OltoAdd),
                          append(Vl, VltoAdd),
                          append(El, EltoAdd),
                          append(Rl, RltoAdd)))
```

この時点で，この状態からの意味のある状態遷移は，第 2 引数に現れているような変更後の仕様に従うもののみとなるので，書換えのリフレクション規則により，“get2from” メソッドの仕様が元の仕様に追加された，と見なすことができる．

6. 関連研究

メタレベルアーキテクチャやリフレクションの表示的意味論および代数的定式化について提案されている[1),2),4),12),13),15),18)]が，いずれもベースレベルの計算の状態に対するメタの操作を扱うものであり，本研究のような動的な記述の変更を扱ってはいない．

また，書換え規則の動的な変更が可能な等式論理体系が提案されている[14)]が，操作的意味論しか与えてい

ないので，厳密な検証を検討するのが困難である．

さらに，リフレクションの別の定式化についても提案されており[19]，値呼びラムダ計算に適用され[20]，具体的な操作的意味論が与えられている．しかし，構文的操作の意味しか検討されておらず，本論文で展開したような表示的意味論のための枠組みの提案には至っていない．

一方本研究は，Flageで実現しているような，動的な記述の変更に対し，代数的な意味を与えている．さらに，始代数モデルを与えることが可能な程度の具体的な定式化であるため，Flageの厳密な検証の基盤を提供するものとなっている．

7. おわりに

本論文では，書換え論理と呼ばれる代数的仕様記述言語にリフレクションを導入した体系RL/Rを提案した．書換え論理では，並行オブジェクト指向システムをモデル化できるので，RL/Rによりメタオブジェクトおよびメタオブジェクトプロトコルを代数的にモデル化することが可能となった．さらにRL/Rは代数的意味論，すなわち代数モデルに基づく宣言的意味論を持つ．このため，帰納的定理証明などの項書換え技術に基づく厳密な検証の基礎を与えることができる．

現在筆者らは，具体的な検証手法の開発を行っており，本論文の手法をより現実的なものとするために，様々な検証事例の蓄積を目指している．

謝辞 本研究は，産業科学技術研究開発制度「新ソフトウェア構造化モデルの研究開発」の一環として情報処理振興事業協会（IPA）が新エネルギー・産業技術総合開発機構から委託をうけて実施したものである．

参考文献

1) Danvy, O. and Malmkjær, K.: Intensions and Extensions in a Reflective Tower, *Proc. ACM Sympo. on LISP and Functional Programming*, pp.327–341 (1988).
2) Friedman, D. and Wand, M.: Reification: Reflection without Metaphysics, *Proc. ACM Sympo. on LISP and Functional Programming*, pp.348–355 (1984).
3) 本位田真一ほか：エージェント指向言語 Flage (1)–(5)，第47回情報処理学会全国大会論文集，pp.5-27–5-36 (1993).
4) 石川 洋，二木厚吉，渡部卓雄：書換え論理に基づく並行自己反映計算のモデル化，日本ソフトウェア科学会第11回大会，pp.313–316 (1994).
5) Kiczales, G., des Rivières, J. and Bobrow, D.G.: *The Art of the Metaobject Protocol*, MIT Press (1991).
6) 粂野文洋，田原康之，大須賀昭彦，本位田真一：フィールド指向言語Flage，ソフトウェア工学の基礎，日本ソフトウェア科学会FOSE'94, pp.31–40，近代科学社 (1994).
7) Kumeno, F., Tahara, Y., Ohsuga, A. and Honiden, S.: Evolutional Agents: Field Oriented Programming Language, Flage, *Proc. Asia Pacific Software Engineering Conference* pp.189–198 (1995).
8) Kurihara, M. and Ohuchi, A.: An Algebraic Specification and an Object-oriented Implementation of a Reflective Language, *Proc. Int. Workshop on New Models for Software Architecture*, pp.137–142 (1992).
9) Meseguer, J.: A Logical Theory of Concurrent Objects, *ECOOP/OOPSLA'90*, pp.101–115 (1990).
10) Meseguer, J.: Conditional Rewriting Logic as a Unified Model of Concurrency, *Theor. Comput. Sci.*, Vol.96, pp.73–155 (1992).
11) Meseguer, J., Futatsugi, K. and Winkler, T.: Using Rewriting Logic to Specify, Program, Integrate, and Reuse Open Concurrent Systems of Cooperating Agents, *Proc. Int. Sympo. on New Models for Software Architecture*, pp.61–106 (1992).
12) Meseguer, J.: A Logical Theory of Concurrent Objects and Its Realization in the Maude Language, *Research Directions in Concurrent Object-oriented Programming*, Chap.12, pp.314–390, MIT Press (1993).
13) Nakajima, S.: What Makes a Language Reflective and How?, *Proc. Int. Workshop on New Models for Software Architecture*, pp.125–136 (1992).
14) 佐藤崇昭，栗原正仁，大内 東：自己反映計算機能をもつ等式プログラム処理系の実現，情報処理学会プログラミング研究会報告，PRG-11-8, pp.69–76 (1994).
15) Smith, B.C.: Reflection and Semantics in Lisp, *Proc. POPL'84*, pp.23–35 (1984).
16) 田原康之，粂野文洋，大須賀昭彦，本位田真一：エージェントモデルにおけるメタレベルアーキテクチャの代数的意味論，ソフトウェア工学の基礎，日本ソフトウェア科学会FOSE'94, pp.121–128，近代科学社 (1994).
17) Tahara, Y., Kumeno, F., Ohsuga, A. and Honiden, S.: Formal Semantics of Agent Evolution in Language Flage Object-based Parallel and Distributed Computing, *LNCS*, Vol.1107, pp.329–348, Springer-Verlag (1995).
18) Wand, M. and Friedman, D.: The Mystery of the Tower Revealed: A Non-reflective Descrip-

tion of the Reflective Tower, *Meta-Level Architectures and Reflection*, pp.111–134, North-Holland (1988).
19) 渡部卓雄：抽象書換えに基づく自己反映計算の定式化手法，日本ソフトウェア科学会第11回大会，pp.309–312 (1994).
20) 山岡順一，渡部卓雄：自己反映的な値呼びλ計算の操作的意味論，情報処理学会プログラミング研究会報告，PRG-21-7, pp.49–56 (1995).

(平成7年5月25日受付)
(平成9年2月 5 日採録)

田原 康之（正会員）

1966年生．1991年東京大学大学院理学系研究科数学専攻修士課程修了．同年（株）東芝入社．1993年より1996年にかけて情報処理振興事業協会に出向．現在（株）東芝研究開発センターシステム・ソフトウェア生産技術研究所より，英国City大学に客員研究員として赴任中．エージェント技術，およびソフトウェア工学などの研究に従事．日本ソフトウェア科学会会員．

粂野 文洋（正会員）

1965年生．1988年早稲田大学理工学部数学科卒業．1990年同大学院理工学研究科修士課程修了．同年（株）三菱総合研究所入社．現在，情報処理振興事業協会（IPA）に出向し，協調アーキテクチャの研究に従事．日本ソフトウェア科学会会員．

大須賀昭彦（正会員）

1958年生．1981年上智大学理工学部数学科卒業．同年（株）東芝入社．1985～1989年（財）新世代コンピュータ技術開発機構に出向．現在（株）東芝研究開発センターシステム・ソフトウェア生産技術研究所に所属．工学博士．主としてソフトウェアのための形式手法，エージェント技術などの研究に従事．1986年度情報処理学会論文賞受賞．電子情報通信学会，日本ソフトウェア科学会，EATCS各会員．

本位田真一（正会員）

1953年生．1976年早稲田大学理工学部電気工学科卒業．1978年同大学院理工学研究科電気工学専攻修士課程修了．工学博士．同年（株）東芝入社．現在，同社研究開発センターシステム・ソフトウェア生産技術研究所に所属．1989年より早稲田大学非常勤講師を兼任．1996年度より大阪大学大学院ならびに筑波大学非常勤講師兼任．主としてエージェント技術，オブジェクト指向技術の研究に従事．1986年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事．著訳書10点．