条件付き項書換え系に基づく言語におけるメタ計算

沼澤政信[†] 栗原正仁[††] 大内東[†]

最近，プログラミング言語としての項書換え系に自己反映計算機能を付加した言語的枠組みとして，REPS（Reflective Equational Programming System）が提案された．本論文では，REPSを基礎として，それを条件付き項書換え系（CTRS）に拡張した言語CREPSを提案し，その処理系の実装と応用について述べる．はじめに，REPSの計算モデルをそのまま用いることによりメタ計算機能を持つCTRSを実現する場合に起きる問題点を指摘し，それを解決するためにCTRS仮想簡約マシンと呼ぶスタックマシンを導入する．次に，このマシンの計算を計算状態の集合上の簡約関係によって定義し，ベースレベルにおける言語の操作的意味を与える．さらに，このマシンに（ベースレベルとメタレベルでオブジェクトをやりとりする）メタ計算機能を付加することにより言語を拡張する．

Meta-computation in a Conditional Term Rewriting System-based Language

MASANOBU NUMAZAWA,[†] MASAHITO KURIHARA[††] and AZUMA OHUCHI[†]

Recently, a reflective programming language, called REPS, has been proposed as a basic reflective framework on top of term rewriting model of computation. In this paper, we extend REPS to a new model CREPS, which makes it possible to perform meta-computation while *conditional* rewriting. First, we discuss a problem with REPS when trying to straightforwardly extend it for conditional rewriting, and to solve this problem, introduce a stack machine called the CTRS virtual reduction machine. Then, we give an operational semantics of the basic CREPS in terms of a reduction relation on the set of computational states of the machine. Finally, we extend the language to full CREPS, by adding to the machine some meta-computational mechanisms in which computational objects are transformed from base-level to meta-level, and vice versa.

1. はじめに

近年，急速な計算機の処理能力の向上にともない，ソフトウェアに対する要求の多様化，システムの複雑化が進んでいる．特に，静的かつ既知な環境ばかりではなく，動的かつ未知な実環境における多様性に対応するシステム開発が望まれている．その要求に柔軟な対応を行うためには，より高度で複雑なプログラムを簡潔に記述できるプログラミング言語の開発および迅速かつ厳密なシステム仕様の確認が必要不可欠，かつ有用である．そこで我々は，自己反映計算の動的適応性および拡張性に注目し，それを仕様記述においてより自然な記述を提供する条件付き項書換え系へ導入することを試み，その基礎的な計算機構を構築する．

メタ計算（プログラミング）機能は，拡張性および動的適応性を持つ柔軟な計算メカニズムを提供するものであり，その機能を付加したプログラミング言語の提案が数多く行われている．中でも特に，計算システムが「自分自身」について感知したり自分自身を変更したりすることを含むような計算は自己反映計算（リフレクション）[1]と呼ばれ，これまでに，関数型[2]，論理型[3]，オブジェクト指向をベースとするもの[4]などが提案されている．

書換え系[5]は，関数プログラミング，等式論理の自動証明や抽象データ型に基づく仕様記述など，記号計算の研究分野における基礎理論として重要な役割を担っている．特に，計算の意味や性質の解析，プログラムの変換・検証といった技術を数学的にとらえられることから，より厳密な議論を展開することを可能としている．書換え系の中でも，広範囲にわたり応用さ

† 北海道大学大学院工学研究科システム情報工学専攻
Division of Systems and Information Engineering, Graduate School of Engineering, Hokkaido University
†† 北海道工業大学工学部電気工学科
Department of Electrical Engineering, Faculty of Engineering, Hokkaido Institute of Technology

れ，しかも性質がよく研究されているものの1つが項書換え系[5]である．これは，代数的仕様記述を用いた仕様においてはプロトタイピング法の試作プログラムとして有効であり，システム構築の際のシステムの機能，仕様の確認を厳密かつ簡単に行える．さらに項書換え系は，規則適用のための条件の明示的記述，プログラム作成の簡易化，代入操作の効率化などを目的として，書換え規則へ条件部を導入する拡張も行われており，条件付き項書換え系[6]，メンバシップ条件付き項書換え系[7]などが提案されている．

最近，書換え系をベースとした自己反映システムの基礎研究もなされてきた．渡部[8]は抽象書換え系を用いて，自己反映計算に対する形式的な定義を与え，山岡ら[9]は自己反映的な抽象書換え系の具体例として，値呼び λ 計算を取り上げ，その場合に得られるプログラミング言語の操作的意味論および処理系の実装を行っている．また，栗原ら[10]は項書換え系に自己反映計算機能を付加した言語的枠組みとしてREPSを提案している．

本論文では，条件付き項書換え系に自己反映計算機能を導入するための基礎的計算モデルの確立を目的として，自己反映計算機能を持つ条件付き項書換え系の基礎的な計算機構を構築する．具体的には，REPSの考え方を基礎として，それを条件付き項書換え系に拡張した言語 CREPS（Conditional REPS）を提案し，その操作的意味を与える．

文献10)では，今後の課題として，条件付き書換え規則を記述できるようにREPSの枠組みを拡張することが述べられている．この拡張のためには，書換え規則の条件部の評価過程を計算状態に適切に反映しなくてはいけない．しかし，これはREPSの定義する計算モデルのままでは実現できないため，新たな計算モデルを導入する必要がある．本論文では，条件評価中のメタ情報を格納する目的から，4つのフィールドを持つレコードを要素とするスタックを持つ仮想簡約マシンを導入し，その計算状態の推移を形式化した簡約関係により言語 CREPS の操作的意味を定義する．その結果，CREPS はREPSの機能を含んだより高機能の言語的枠組みとなる．

2. 準　　備

本章では，項書換え系および条件付き項書換え系の概念と記号について説明する．基本的には文献5), 10)に従うが，重要なものについてのみ以下で簡単に述べる．

2.1 項書換え系

関数記号の集合を $\mathcal{F}$，**変数**の集合を $\mathcal{V}$ とする．各関数記号 $F \in \mathcal{F}$ には，**項数**と呼ばれる非負整数が付随している．項数0の関数記号を**定数**と呼ぶ．$\mathcal{F}$ は，構成子の集合 $\mathcal{F}_C$ と演算子の集合 $\mathcal{F}_D$ の2つの素な集合に分割される．

項の集合を $\mathcal{T}(\mathcal{F}, \mathcal{V})$（または単に，$\mathcal{T}$）で表す．構成子のみからなる（変数も含まない）項を**構成子項**と呼ぶ．$\equiv$ は2つの項が構文的に同一であることを表す．項 t を構成するために用いられた項（t 自身も含む）を t の**部分項**という．部分項の**出現位置**は文献5)に従い，正整数の列によって表現される．項 t におけるすべての出現位置の集合を $\Pi(t)$，出現位置 $\pi \in \Pi(t)$ における t の部分項を $t|_\pi$ と表す．項 t における $t|_\pi$ を項 u で置き換えてできる項を $t[u]_\pi$ と表す．

項書換え系（TRS）は**書換え規則**（**ルール**）の集合 $\mathcal{R}$ である．ルールは項の順序対 $e \to e'$ である．ただし，e は変数ではなく，e' に出現する変数は必ず e にも出現する．e のインスタンスを**リデックス**という．$\mathcal{R}$ によって定義される $\mathcal{T}$ 上の**簡約関係**を $\to_\mathcal{R}$ で表す．$s \to_\mathcal{R} t$ を**簡約ステップ**，その列 $t_0 \to_\mathcal{R} t_1 \to_\mathcal{R} \ldots t_i \to_\mathcal{R} \ldots$ を**簡約列**と呼ぶ．$\to_\mathcal{R}$ の反射推移閉包を $\to^*_\mathcal{R}$ で表す．与えられた項 s，t に対し，$s \to^*_\mathcal{R} u$ かつ $t \to^*_\mathcal{R} u$ なる項 u が存在するとき，s と t は**会同する**（*join*）と呼び $s \downarrow_\mathcal{R} t$ と書く．

2.2 条件付き項書換え系

条件付き項書換え系（CTRS）は，$e \to e'$ *if* $t_1 = t'_1, \ldots, t_n = t'_n$ の形で与えられる**条件付き書換え規則**（**条件付きルール**，あるいは単に**ルール**）の集合である．項 e を**左辺**，項 e' を**右辺**，項の対から作られる等式 $t_i = t'_i$ を**条件式**，その列 $t_1 = t'_1, \ldots, t_n = t'_n$ を**条件**という．条件を持たない（$n = 0$ の）ルールは $e \to e'$ と書く．また，左辺は変数ではなく，右辺または条件に出現する変数は必ず左辺にも出現する．

条件式の等号 $=$ の解釈（評価）の仕方により，異なるCTRSが定義されている[6]．本論文では，その中で最も実用的と考えられている会同CTRSを採用する．

定義 2.1　会同（join）CTRS $\mathcal{R}$ は条件中の等号 $=$ を会同性 $\downarrow_\mathcal{R}$ と解釈する．すなわち，項 s と s' が $\mathcal{R}$ に含まれるある条件付きルール

$$e \to e' \ \textit{if} \ t_1 = t'_1, \ldots, t_n = t'_n \in \mathcal{R}$$

文脈 $C[\,]$，代入 σ によって，$s \equiv C[e\sigma]$ かつ $s' \equiv C[e'\sigma]$ を満たし，かつ任意の $i \in \{1, \ldots, n\}$ について $t_i\sigma \downarrow_\mathcal{R} t'_i\sigma$ であるとき，(またそのときに限り)

$s \to_{\mathcal{R}} s'$ である（厳密に述べると，このような再帰的条件を満たす最小の関係を $\to_{\mathcal{R}}$ と定義する）．

定義 2.2 会同 CTRS $\mathcal{R}$ に対し，CTRS $\mathcal{R}_i$ $(i = 0, 1, 2, \ldots)$ を以下のように定義する．

$$\begin{aligned}\mathcal{R}_0 &= \{e \to e' \mid e \to e' \in \mathcal{R}\}\\ \mathcal{R}_{i+1} &= \{e\sigma \to e'\sigma \mid\\ &\quad e \to e' \ if\ t_1 = t_1', \ldots, t_n = t_n' \in \mathcal{R} \text{ かつ},\\ &\quad \forall j \in \{1, \ldots, n\} \text{ に対して}, t_j\sigma \downarrow_{\mathcal{R}_i} t_j'\sigma\}\end{aligned}$$

すべての i について $\mathcal{R}_i \subseteq \mathcal{R}_{i+1}$ である．さらに，$s \to_{\mathcal{R}} s'$ ならば，ある i について $s \to_{\mathcal{R}_i} s'$ であり，またその逆も成り立つ．簡約ステップ $s \to_{\mathcal{R}} s'$ の深さを，$s \to_{\mathcal{R}_i} s'$ を満たす最小の i として定義する．

例 2.1 次の CTRS $\mathcal{R}$ を考える．

$$\mathcal{R} = \begin{cases} even(0) \to true\\ even(S(x)) \to odd(x)\\ odd(x) \to true \quad if \quad even(x) = false\\ odd(x) \to false \quad if \quad even(x) = true \end{cases}$$

$\mathcal{R}$ に定義 2.2 を適用すると以下のようになる．

$$\begin{aligned}\mathcal{R}_0 &= \{even(0) \to true,\ enen(S(x)) \to odd(x)\}\\ \mathcal{R}_1 &= \mathcal{R}_0 \cup \{odd(0) \to false\}\\ \mathcal{R}_2 &= \mathcal{R}_1 \cup \{odd(S(0)) \to true\}\end{aligned}$$

以下に簡約列の例を示す．第 1 ステップの深さは 0，第 2 ステップの深さは 1 である．

$$even(S(0)) \to_{\mathcal{R}} odd(0) \to_{\mathcal{R}} false$$

3. CTRS 仮想簡約マシン

我々は，REPS のベースレベルの計算モデルを通常の項書換え系から条件付き項書換え系に拡張し．条件評価中の簡約ステップに対するメタ操作をも行えるようにしたい．本章では，その際生じる従来の REPS の枠組み上での問題点を示し，その解決策として新たな仮想簡約マシンと状態推移規則を導入する．

3.1 動 機

REPS においては，自己反映計算機能により，書換え規則の集合（プログラム）が計算中に動的に変化しうる．そのため，仮想簡約マシンの状態を，書換え対象である項 s とプログラム $\mathcal{R}$ の順序対 $(s, \mathcal{R})$ で定義している．計算は状態集合上の二項関係 $\Rightarrow$ によってモデル化され，簡約列 $(s_1, \mathcal{R}_1) \Rightarrow (s_2, \mathcal{R}_2) \Rightarrow \ldots$ として定義される．

このモデルは抽象度が高すぎ，条件評価中のメタ操作の定義にそのまま用いることはできない．なぜならば，トップレベルでの書換えのみを用いて簡約ステップを定義しているため，条件評価中のマシンのメタ情報を明示的に定義できないからである．たとえば，このモデルをそのまま用いると，例 2.1 の計算は $(even(S(0)), \mathcal{R}) \Rightarrow (odd(0), \mathcal{R}) \Rightarrow (false, \mathcal{R})$ となり，この簡約列は条件評価中の簡約を陽に表現してはいない．つまり，深さ 1 である第 2 ステップで行われる条件評価中の簡約 $even(0) \to_{\mathcal{R}} true$ が計算の定義に反映されていない．

すなわち，このモデルは計算の 1 ステップを定義しているという点では計算意味論[11]的だが，そのステップが大きな計算（条件評価）を内包しており，その計算過程を明示せずに最終的な評価結果のみに注目して計算を定義しているという点では評価意味論[11]的な側面がある．我々はこの点を考慮し，条件評価をさらに細かなステップに分け，その 1 ステップを陽に定義する方法で，より計算意味論的なモデルを構築する．

この新しいモデルにおいては，REPS と異なる新しい計算状態および状態推移規則（$\Rightarrow$）を導入する．その特徴は以下の点にある．

- 計算状態の定義にスタックを導入する．
- 状態推移の制御のために相と呼ぶ情報を導入する．

スタックは，簡約の深さとスタックの深さを対応させることにより，条件評価の簡約ステップのメタ情報を自然に格納できる．相は，マシンが現在，何を実行しようとしているかを表すために導入する制御情報であり，照合相と置換相がある．

条件部の評価を明示的にメタレベルで扱えるようなモデル化が実現されている先駆け的研究としては，文献 3) の一階述語論理に基づく並列論理プログラミング言語 GHC（Guarded Horn Clauses）での自己反映計算の実現（RGHC: Reflective GHC）があるが，これは我々の提案するスタックを用いたモデルとは異なるモデルを提案している．それは RGHC で RGHC を記述したメタインタプリタを構築し，その中に実行過程における様々な情報を段階的に顕在化させて，それらを扱うリフレクティブな命令を定義している．

3.2 計 算 状 態

仮想簡約マシンの計算状態を，スタック S，プログラム $\mathcal{R}$ の順序対 $(S, \mathcal{R})$ で表す．スタック S の要素は 4 つのフィールドを持つレコード $\langle T, E, M, P\rangle$ とする．**表 1** に各フィールドのデータ型および初期状態を示す．計算の初期状態においてはスタックの深さは 1 であり，表 1 に示す唯一のレコード要素が積まれていると仮定する．ただし，$top(T)$ はスタック S のトップ要素におけるフィールド T のデータを，{ }

表1 レコード要素における4つのフィールド
Table 1 The four fields of the record.

フィールド	データ型	初期状態
T: 項	項	$top(T)$ = 初期項 t
E: 条件	条件項の集合	$top(E) = \{\}$
M: 照合	(ルール, 位置) の集合	$top(M) = \mathcal{R} \times \Pi(t)$
P: 相	相	$top(P) = Match$

は空集合を表す.

フィールド T には, 各深さにおける (条件付き) 簡約ステップの簡約の対象となる項 (**対象項**) が積まれる. すなわち, スタックの底のフィールド T には, 初期項 s_0 またはその簡約項 s' ($s_0 \to_{\mathcal{R}}^{*} s'$ を満たす) が積まれる. 条件式 (たとえば, $s = t$) を評価する場合には, それを $eq(s, t)$ という形の項として表現したもの (**条件項**) がスタックのトップのフィールド T に積まれる. ただし, eq は項数2のあらかじめ決められている特定の関数記号である. $top(T)$ 要素を**現対象項**と呼ぶ.

フィールド E には, 現在の簡約の深さにおいて評価すべき条件項のうち, 上記の $top(T)$ 以外のものの集合が積まれる.

対象項の簡約において, マシンはどのルールをどの位置に適用できるかを探索する. 探索空間は, ルールの集合 $\mathcal{R}$ と対象項 t 中の位置の集合 $\Pi(t)$ の直積 $\mathcal{R} \times \Pi(t) = \{(\rho, \pi) \mid \rho \in \mathcal{R}, \pi \in \Pi(t)\}$ で表される. マシンは, この空間から非決定的に1つの要素 (ρ, π) を選択し, ルール ρ を $t|_\pi$ に適用できるか判定する. 適用できないと判定されたときは, この要素を削除することにより探索空間を縮小した後, 別の要素を選択する. フィールド M には, このような過程における現在の探索空間が積まれる. 処理系の実現の際には, 要素 (ρ, π) の具体的な選択方式が重要である. これを**簡約戦略**と呼ぶ.

フィールド P は計算状態を制御するための相と呼ばれる情報を持つ. 相には**照合相** ($Match$) と**置換相** ($Replace$) がある. ただし, 置換相は付加情報として評価中のルールの右辺 e', 代入 σ および位置 π を持ち, $Replace(e', \pi, \sigma)$ と記述する.

3.3 状態推移規則

仮想簡約マシンの計算状態は8種類の状態推移規則により推移する. **図1** にそれらを計算状態の集合上の簡約関係 $\Rightarrow$ に関しての公理として形式的に示し, 本節で順にその内容を説明する. 便宜上, $t \equiv top(T)$ とする. 図中では, スタック S に要素 s をプッシュした結果得られるスタックを Prolog 言語のリスト記法を借りて $[s \mid S]$ と表す. $[s', s \mid S]$ は $[s' \mid [s \mid S]]$ の略記である. _ は Don't Care を表す. 本節の範囲ではメタ計算機能を扱わないため, 計算途中でプログラム $\mathcal{R}$ が変化することはないので, 計算状態を単にスタック S として略記する.

3.3.1 照 合 相

照合相の役割は, (1) ルール照合と (2) 条件照合の2つである. (1) は通常の書換えのためのルールの左辺と対象項との照合である. (2) は特別なパターン $eq(x, x)$ と条件項との照合である.

(1) ルール照合においては, 現対象項 t, $top(M)$ から選択される位置 π, およびルール ρ の左辺 e によりパターン照合 ($\exists\sigma.\ t|_\pi \equiv e\sigma$) を試み, その結果に応じて図1 (1-1)~(1-3) のいずれか1つの推移規則が実行される. ただし, $\uplus$ は共通部分を持たない2つの集合の和 (直和) を表す. いずれのケースにも指示されている $e \in \mathcal{T}$ の条件は, ρ の左辺が項であることを要求しており, 本節の範囲ではつねに成立するが, 次章でメタ計算機能を付加するときに必要となる.

$t|_\pi \equiv e\sigma$ を満たす代入 σ が存在しないときには (1-1) が実行される. もし $t|_\pi$ が条件なしルールとの照合に成功したならば (1-2) が, 条件付きルールとの照合に成功したならば (1-3) が実行される. (1-3) の場合には, 書換えの深さが1だけ増えるのにともなって, スタックに新たなレコードが積まれる. 特に, 各条件式を条件項 $eq(t_i\sigma, t_i'\sigma)$ $(1 \le i \le n)$ とし, それらを要素とする集合を $E(\rho, \sigma)$ とするとき, 任意の要素 $e_i \equiv eq(t_i\sigma, t_i'\sigma)$ が非決定的に選ばれて $top(T)$ に, 残りの集合 $E(\rho, \sigma) - \{e_i\}$ は $top(E)$ に格納される.

(2) 条件照合においては, t が条件項 $eq(u, u')$ で, かつ $u \equiv u'$ であるかどうかを判定する.

$u \not\equiv u'$ かつ $top(M) = \{\}$ であるときには, u と u' は異なる正規形であり (2-1) が実行される. $u \equiv u'$ のときは, $top(E) = \{\}$ ならば (2-2) が, $top(E) \neq \{\}$ ならば (2-3) が実行される.

(1-1)~(1-3), (2-1)~(2-3) のどれにもあてはまらないときは, $t \equiv top(T)$ は条件項ではない, かつ $top(M)$ は空集合である. したがって, このとき t は正規形である.

3.3.2 置 換 相

置換相では, リデックス $t|_\pi$ を対応する右辺のインスタンス $e'\sigma$ に置換することにより, t を $t' \equiv t[e'\sigma]_\pi$ に書き換える. さらに, 次の対象項の決定と照合のための準備を行う. t がトップレベルの項のときは, (3-1) により t' が次の対象項となる. t が条件項のときは, (3-2) により t' を $top(E)$ に戻し, その中から非決

(1) ルール照合

(1-1) **不照合**：

$[\langle t, E, \{(\rho, \pi)\} \uplus M, Match\rangle \mid S']$
$\Rightarrow [\langle t, E, M, Match\rangle \mid S']$
ただし，$\nexists\sigma.\ t|_\pi \equiv e\sigma$.

(1-2) **照合成功（条件なし）**：

$[\langle t, E, \{(\rho, \pi)\} \uplus M, Match\rangle \mid S']$
$\Rightarrow [\langle t, E, M, Replace(e', \pi, \sigma)\rangle \mid S']$
ただし，$n = 0,\ t|_\pi \equiv e\sigma$.

(1-3) **照合成功（条件付き）**：

$[\langle t, E, \{(\rho, \pi)\} \uplus M, Match\rangle \mid S']$
$\Rightarrow [\langle e_i, E(\rho, \sigma) - \{e_i\}, \mathcal{R}\times\Pi(e_i), Match\rangle,$
$\langle t, E, M, Replace(e', \pi, \sigma)\rangle \mid S']$
ただし，$n > 0,\ t|_\pi \equiv e\sigma,\ e_i \equiv eq(t_i\sigma, t'_i\sigma),$
$E(\rho, \sigma) = \{e_1\ e_2\ \ldots\ e_n\}$.

ただし，(1-1)〜(1-3) において，
$\rho = e \to e'\ if\ t_1 = t'_1, \ldots, t_n = t'_n,$
$e \in \mathcal{T},\ 1 \le i \le n$

(2) 条件照合

(2-1) **条件不成立**：

$[\langle eq(u, u'), \_\_, \{\}, Match\rangle, \langle T, E, M, \_\_\rangle \mid S'']$
$\Rightarrow [\langle T, E, M, Match\rangle \mid S'']$
ただし，$u \not\equiv u'$.

(2-2) **条件成立**：

$[\langle eq(u, u), \{\}, \_\_, Match\rangle \mid S'] \Rightarrow S'$

(2-3) **条件式成立**：

$[\langle eq(u, u), \{e\} \uplus E, \_\_, Match\rangle \mid S']$
$\Rightarrow [\langle e, E, \mathcal{R}\times\Pi(e), Match\rangle \mid S']$

(3) 置換

(3-1) **置換**：

$[\langle t, E, \_\_, Replace(e', \pi, \sigma)\rangle \mid S']$
$\Rightarrow [\langle t', E, \mathcal{R}\times\Pi(t'), Match\rangle \mid S']$
ただし，$t \not\equiv eq(\_\_, \_\_),\ e' \in \mathcal{T},\ t' \equiv t[e'\sigma]_\pi$.

(3-2) **条件式置換**：

$[\langle t, E, \_\_, Replace(e', \pi, \sigma)\rangle \mid S']$
$\Rightarrow [\langle t'', E \cup \{t'\} - \{t''\}, \mathcal{R}\times\Pi(t''), Match\rangle \mid S']$
ただし，$t \equiv eq(\_\_, \_\_),\ e' \in \mathcal{T},$
$t' \equiv t[e'\sigma]_\pi,\ t'' \in E \cup \{t'\}$.

図1 CTRS 仮想簡約マシンの状態推移規則
Fig. 1 Transition rules for CTRS virtual reduction machine.

定的に選ばれる条件項 t'' を新たな対象項とする．

4. メタ計算機能の導入

本章では，前述の仮想簡約マシンにメタ計算機能を付加する．はじめに，メタ計算のための基本機能およびその構文について説明する．次に，メタ計算機能を付加した言語 CREPS の簡約関係に基づく操作的意味を述べる．最後に，メタ計算機能の効率化および応用について述べる．

4.1 メタ変換とベース変換

CREPS は，項，プログラムおよびスタックをシステムがアクセスするメタレベル・オブジェクトとして扱い，一方，構成子項をユーザプログラムがデータとしてアクセスできるベースレベル・オブジェクトとして扱う．そして，メタ計算のための基本機能として，前者から後者への変換（**メタ変換**）およびその逆変換（**ベース変換**）を提供する．メタ変換とベース変換はそれぞれ，メタ情報の参照と更新に対応している．

メタ変換で得られたデータを**メタ表現**と呼ぶ．REPS[10] と同様に，$term^\wedge$ および $rules^\wedge$ をそれぞれ，項およびプログラムをメタ変換する関数とする．また新たに，スタック S をメタ変換する関数として $Stack^\wedge$ を導入する．同様に，ベース変換する関数として，$term^\vee$，$rules^\vee$ および $Stack^\vee$ を用意する．

各関数記号 $F \in \mathcal{F}$ および変数 $x \in \mathcal{V}$ に対して，構成子でかつ定数である $F' \in \mathcal{F}_C$ および $x' \in \mathcal{F}_C$ がそれぞれ，1対1に対応して存在していると仮定する．これはメタレベルにおいて，関数記号や変数を構成子項として表現することを可能とする．特に，もし F が構成子でかつ定数であれば，$F' \equiv F$ であるとする．

メタ変換を表す各関数については，REPS[10] と同様の考え方で定義できるので，本論文では後の例題で必要とする $term^\wedge$ のみを示す．なお，ここでは，リストを表すために，Lisp 言語の記法を用いている．たとえば，$(a\ \ b)$ は項 $cons(a, cons(b, nil))$ の略記である．ただし，var, $cons$, nil はあらかじめ決められている構成子である．

$term^\wedge(x) \equiv var(x'),$ when $x \in \mathcal{V},\ x' \in \mathcal{F}_C$
$term^\wedge(c) \equiv c,$ when $c \in \mathcal{F}_C, arity(c) = 0$
$term^\wedge(F(t_1, \ldots, t_n))$
$\equiv (F'\ term^\wedge(t_1)\ \ldots\ term^\wedge(t_n)),$
when $arity(F) > 0$
or $arity(F) = 0 \wedge F \in \mathcal{F}_D, F' \in \mathcal{F}_C$

4.2 メタルール

一般のCTRSではルールの左辺と右辺はともに項であるが，CREPSでは項ではない**メタ計算式**を導入する．メタ計算式とは，ルールの左辺のトップレベルに $\ell : s.r$，あるいは右辺のトップレベルに $s.r$（ただし，$\ell, s, r \in \mathcal{T}$）としてのみ現れる表現である．これらは，システムのメタ計算機能とのインタフェースの役割を果たすものであり，書換えの対象にはならない．項とメタ計算式を総称して**式**と呼ぶ．

左辺の ℓ，s，r のそれぞれと，部分項，スタック，プログラムの各メタ表現との照合により，メタレベルからベースレベルへの情報の伝達を行うことができる．これを**メタ照合**と呼び，以下で正式に定義する．

メタ照合では，以下の条件を満たす代入 σ が存在するかどうかを確かめる．

$$\begin{cases} \ell\sigma \equiv F(term^{\wedge}(t_1), \ldots, term^{\wedge}(t_m)), \\ s\sigma \equiv Stack^{\wedge}(S), \\ r\sigma \equiv rules^{\wedge}(\mathcal{R}). \end{cases}$$
ただし，$t \equiv F(t_1, \ldots, t_m)$

この条件を述語 $Meta\text{-}Match\,(t,\ \ell : s.r\ ,\ \sigma\ )$ で表す．ただし，この述語は暗黙に，照合の際のマシンの状態 $(S, \mathcal{R})$ にも依存することに注意する．メタ照合に成功したときは，マシンの状態のメタ表現が σ に保持されている．

同様に，右辺がメタ計算式 $s.r$ であるとき，そのインスタンスがベース変換され，それがシステムの新たな状態に反映される．これを**メタ置換**と呼ぶ．

次節の状態推移規則から明らかになるように，左辺がメタ計算式であるルール

$\ell : s.r \rightarrow$ (式) *if* (条件)

はメタ変換，右辺がメタ計算式であるルール

(式) $\rightarrow s.r$ *if* (条件)

はベース変換を引き起こす．これらのルールを**メタルール**と呼ぶ．

4.3 状態推移規則のメタ計算への拡張

すでに述べたベースレベルの状態推移規則に，新たにメタレベルに関係する4つの推移規則を追加して簡約関係 $\Rightarrow$ を拡張する（**図2**）．以下，$t \equiv top(T)$ として各ステップを説明する．

照合相における状態推移規則を3つ追加する．各推移規則は，ルール ρ の左辺がメタ計算式 $\ell : s.r$ であるときに限り適用可能である．どの推移規則を適用するかは，項 $t|_\pi$ と左辺 $\ell : s.r$ のメタ照合の結果に依存する．

メタ置換推移規則は，ルール ρ の右辺がメタ計算式 $s.r$ であるときに限り適用可能である．これはプログラム $\mathcal{R}$ を変えることができる唯一の規則なので，状態記述に $\mathcal{R}$ を陽に加えた．メタ置換では，s および r のインスタンスがベース変換されて，システムの新たな状態 $(\hat{S}, \hat{\mathcal{R}})$ となる．

(1) ルール照合

(M-1) メタ不照合：

$[\langle\ t,\ E,\ \{\ (\rho,\ \pi)\ \} \uplus M,\ Match\ \rangle \mid S']$
$\Rightarrow\ [\langle\ t,\ E,\ M,\ Match\ \rangle \mid S']$
ただし，$\nexists \sigma.Meta\text{-}Match\ (\ t|_\pi\ ,\ \ell : s.r\ ,\ \sigma\ ).$

(M-2) メタ照合成功（条件なし）：

$[\langle\ t,\ E,\ \{\ (\rho,\ \pi)\ \} \uplus M,\ Match\ \rangle \mid S']$
$\Rightarrow\ [\langle\ t,\ E,\ M,\ Replace(e',\ \pi,\ \sigma)\ \rangle \mid S']$
ただし，$n = 0$，$Meta\text{-}Match\ (\ t|_\pi\ ,\ \ell : s.r\ ,\ \sigma\ ).$

(M-3) メタ照合成功（条件付き）：

$[\langle\ t,\ E,\ \{\ (\rho,\ \pi)\ \} \uplus M,\ Match\ \rangle \mid S']$
$\Rightarrow\ [\langle\ e_i,\ E(\rho,\ \sigma) - \{\ e_i\ \},\ \mathcal{R} \times \Pi(e_i),\ Match\ \rangle,$
$\langle\ t,\ E,\ M,\ Replace(e',\ \pi,\ \sigma)\ \rangle \mid S']$
ただし，$n > 0$，$Meta\text{-}Match\ (\ t|_\pi\ ,\ \ell : s.r\ ,\ \sigma\ ),$
$c_i = eq(t_i\sigma, t'_i\sigma),\ E(\rho,\ \sigma) = \{e_1\ e_2\ \ldots\ e_n\}.$

ただし，(M-1)，(M-2)，(M-3) において，
$\rho = \ell : s.r \rightarrow e'$ *if* $t_1 = t'_1, \ldots, t_n = t'_n,$
$\ell : s.r \notin \mathcal{T},\ 1 \leq i \leq n$

(3) 置換

(R-1) メタ置換：

$([\langle\ \_,\ \_,\ \_,\ Replace(\ s.r\ ,\ \pi,\ \sigma)\ \rangle \mid S'],\ \mathcal{R})$
$\Rightarrow\ (\hat{S},\ \hat{\mathcal{R}})$
ただし，$\hat{S} = Stack^{\vee}(s\sigma)$，$\hat{\mathcal{R}} = rules^{\vee}(r\sigma).$

図2 照合相および置換相における追加推移規則
Fig. 2 Transition rules added for match and replace phase.

自己反映計算を用いる必要のない問題領域における計算では，これらの新たな状態推移規則の導入による実行効率の低下はほとんどない．つまり，そのような場合，CREPSと自己反映機能のないCTRS（本論文ではCTRS仮想簡約マシンであり，これは通常言語でのインタプリタに相当する処理系である）は書換え規則のまま実行を行う限りにおいてほぼ同様の実行効率である．なぜならば，CREPSは，計算中にメタ計算式との照合が行われない限りは，CTRS仮想簡約マシンの状態推移規則でのみ計算が実行されるからである．

4.4 処理系の実装の概略

本節では，CREPS処理系の実装上における構文，およびメタ計算機能の効率化についての概略を述べる．

構文は基本的にはREPS[10]と同様である．CREPSにおいても，$\mathtt{F} \in \mathcal{F}$，$\mathtt{F}' \in \mathcal{F}_C$ の両者とも F によって表記する．また，項の引数は[ ]で囲み，F が定数であるときは，もしそれが演算子であれば F[]，構成子であれば F と表記する．変数は？マークを変数名の前

につけて表記する．つまり，?x のメタ表現はvar[x]である．ルールの左辺と右辺を区切るには → ではなく = を用いる．

次に，CREPS 処理系が極端に非効率なものになることを避ける目的で，遅延評価の導入によるメタ計算機能の簡単な効率化を行う．

REPS においては，メタ計算において触るデータがシステム状態のごく一部であるときに，メタレベル・オブジェクトとベースレベル・オブジェクトとの間の変換が主要なオーバヘッドとなっていた．CREPS では，これを解決するために**遅延評価**（*delayed evaluation*）を導入した．遅延評価は我々が新たに提案する技法ではなく，このようなオーバヘッド問題を解決するために使用される一般的技法である．自己反映言語システムへの導入例としては文献 4) があり，遅延評価のしくみをコンパイル法に導入した部分計算コンパイル技法による本格的な効率化を試み，ベンチマークプログラムによる明確な実行評価を行いその効果を示している．しかし，CREPS の提案目的は前述のように基礎的な計算モデルの確立にあるので，本論文では，文献 4) のような本格的な効率化は今後の課題として，それに対する厳密な議論を行わない．ここでは，メタ変換における非効率な変換を避ける目的で遅延評価技術を CREPS へ組み込み，その実装の簡単な説明と，実験による遅延評価の確認を行う．

メタ変換は，メタレベル・オブジェクトを構成子項に変換する．このとき，メタレベル・オブジェクトのサイズに比例してその変換時間は長くなり，後にそのデータにアクセスがまったくない，もしくはその大部分にアクセスがない場合などには，その変換は非効率的である．この非効率な変換を防ぐために，CREPS では，変換要求があっても即座に変換せず，変換すべきオブジェクトから**遅延オブジェクト**と呼ばれるオブジェクトへの変換にとどめておき，その後，遅延オブジェクトの内部へのアクセス要求があった場合にのみ，それに応じて一部分を変換する遅延評価方法をとる．

以下では CREPS における遅延評価の実装を簡単に説明する．メタ変換すべき項 f[a,b] はクラス App-lication のインスタンスとして実現されている．これは，遅延評価をしない *term*^ によって，3 つの App-lication オブジェクトで実現されるリストに変換される（**図 3** (a)）．これに対して，遅延評価を行う *term*^（図 3 (b)）では，項 f[a,b] のクラスは Application から遅延オブジェクト Delayed に変換される．

次に，この遅延オブジェクトの第 1 要素（関数記号 f）へのアクセスを行うと，部分的なメタ変換が行われるが，引数部分の (a b) においては，依然アクセス要求がないのでメタ変換は遅延されている．

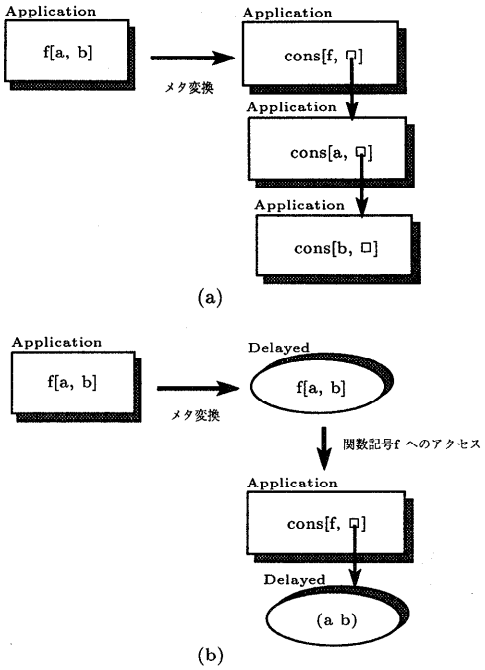


図 3 遅延評価
Fig. 3 Delayed evaluation.

最後に，実験により遅延評価の効果の確認を行う．メタ変換のオーバヘッドをルールの大きさとの関連でとらえられるように，50 本および 100 本のルールからなるプログラム（自身 $\mathcal{R}$）に新たに 1 本のルールを加えるプログラム $\mathcal{R}$ を実験プログラムとする．これは自己反映計算の目的の 1 つである動的適応性のあるプログラムが持つ基本的機能であると考えられる．

実験結果として，メタ変換，ベース変換に主に関係する関数の呼び出し回数を**表 2** に示す．ただし，各関数の機能は次のようになっている．グループ (a) の 3 つの関数により対象項およびルールのメタ変換要求が，グループ (b) の 3 つの関数によりベース変換要求が行われる．遅延評価の導入前におけるメタ変換は，グループ (c) の 4 つの関数により各項の種類（定数，変数，複合項，リスト）に応じたクラスのインスタンスを生成することで実現される．遅延評価における遅延オブジェクト（META-TERM クラスおよび META-EQNS クラスのインスタンス）生成はグループ (d) の関数により実現される．

遅延評価導入後のメタ変換要求において，図 3 (b) に示すような遅延オブジェクト生成をグループ (d) の関数が行うことにより，グループ (a) の関数の呼び出し回数が 1，2 回に減少している．また，この関数呼

表2 測定結果：関数呼び出し回数
Table 2 The result: The number of calls for each function.

	関数名	ルール50本		ルール100本	
		導入前	導入後	導入前	導入後
(a)	TERM-WEDGE	10354	2	21754	2
	EQNS-WEDGE	51	1	101	1
	EQN-WEDGE	50	0	100	0
(b)	TERM-VEE	10356	2	21756	2
	EQNS-VEE	52	2	102	2
	EQN-VEE	51	1	101	1
(c)	MK-CONST	19724	107	41474	207
	MK-VAR	2074	1037	4274	2137
	MK-APP	39049	9227	82149	19427
	MK-LIST	19083	18	40133	18
(d)	MK-META-TERM	-	2	-	2
	MK-META-EQNS	-	1	-	1

び出しは，遅延評価導入前にはルール本数に比例するが，導入後は遅延オブジェクト生成により無駄な変換を行わないため一定となっている．ベース変換要求に関わるグループ(b)の3つの関数についてもグループ(a)の関数とほぼ同様の結果を得ている．

グループ(c)の4つの関数の呼び出し回数については，導入前にはルール50本，100本でそれぞれ合計約80000回，約168000回であるのに対して，導入後はそれぞれ合計約10000回，約22000回に減少している．グループ(d)の関数の遅延オブジェクト生成により多くの無駄な変換を回避できたことが確認できる．

4.5 応　　用

項書換え系に基づく言語におけるメタ計算の応用事例は文献10)で議論されている．本節では，REPSでも実現可能ではあるが，今までに示されたことのない応用例を示す．

メンバシップ条件付き項書換え系（MCTRS）[7]においては，各ルールにメンバシップ条件と呼ばれる条件を付けることができる．MCTRSにおけるメンバシップ条件は，一般にメタレベルの情報を必要とするため，ルールでは仕様を記述できない．それに対して，メタレベルの情報を直接扱えるCREPSは，「構成子項である」あるいは「項の根の関数記号が構成子である」などのメンバシップ条件を記述できる．例として，$F=\{d,+,s,0\}$, $F'=\{+,s,0\}$ を関数記号の集合とし，次の系 $\mathcal{R}$ を考える[7]．

$$\mathcal{R}=\begin{cases} x+0\to x \\ x+s(y)\to s(x+y) \\ d(x)\to x+x \qquad if \quad x\in\mathcal{T}(F') \end{cases}$$

$\mathcal{R}$ に初期項 $d(d(0))$ を与えると，次の簡約列を得る．

$d(d(0))\to_{\mathcal{R}} d(0+0)\to_{\mathcal{R}} (0+0)+(0+0)\to^{*}_{\mathcal{R}}0$

メンバシップ条件により $d(0)\notin\mathcal{T}(F')$ なので，$d(d(0))\to_{\mathcal{R}} d(0)+d(0)$ という簡約は行われない．

この例をCREPSで記述すると以下のようになる．

```
(1) plus[?x,0] = ?x.
(2) plus[?x,s[?y]] = s[plus[?x,?y]].
(3) d[?x] = plus[?x,?x] if plus-expr[?x] = true.
(4) plus-expr[?x]:?s.?r = p-expr[?x].
(5) p-expr[0] = true.
(6) p-expr[(s ?x)] = p-expr[?x].
(7) p-expr[(plus ?x ?y)] = true
        if p-expr[?x] = true, p-expr[?y] = true.
```

このプログラムでは，関数 **plus-expr** においてメタ変換を行うことで，引数の項の構造を（メタレベルで）調べる．初期項 **d[d[0]]** を与えると，初期項全体に(3)を適用しようとする．その結果，**plus-expr[d[0]] = true** の条件評価に入る．左辺は(4)のメタ変換により，**p-expr[(d 0)]** に書き換えられるが，これはすでに正規形である．よって，条件は偽と判定され，初期項全体には(3)を適用できないことが分かる．次に，初期項の第1引数 **d[0]** に(3)を適用できるかどうかを調べるために，条件 **plus-expr[0] = true** の評価に入る．左辺は(4)のメタ変換により **p-expr[0]** に，さらに(5)により **true** へ書き換えられる．よって，条件は真と判定され，(3)により初期項は **d[plus[0,0]]** に書き換えられる．

次に，デバッガとのインタフェースへの応用例の概略を述べる．著者らの調べた範囲では，実際にデバッガをルールとして記述した例はみつからないが，CREPSにさらに入出力機能を付け加えれば，これは原理的に可能となる．

プログラムにエラーが生じ，デバッガに現在のプログラムとスタック情報を渡すと仮定する．たとえば，自然数上での除算を定義したプログラムに次の2本のルールを加える．

```
(1) div[?n, 0]:?s.?r = debug[?n, ?s, ?r] .
(2) continue[?s, ?r] = ?s.?r .
```

項 **div[s[s[0]], 0]** がリデックスの候補として選ばれると，(1)によりメタ変換が行われる．これにより，**?n**, **?s**, **?r** にはそれぞれ，項，スタックおよびプログラムのメタ表現が代入される．関数 **debug** は引数としてこれらの情報を受け取り，適当なデバッグ動作を行う．その後，ユーザとの対話等によりこれらの内容が更新され，計算を再開するために関数 **continue** が呼び出されるものと仮定する．そうすると，ルール(2)によりメタ表現がベース変換され，ここで得られた新しい計算状態から計算が再開される．

5. おわりに

本論文では，条件付き項書換え系に基づく言語にメ

タ計算機能を導入した言語CREPSを実現した．スタックを持つCTRS仮想簡約マシンに基づいて計算モデルを定義し，条件評価中の計算状態のメタ情報を得ることを可能とした．また，効率と応用例について述べた．

対象項に対してどの部分項を簡約するかということは，正規化戦略（正規形が存在するならば必ず求める）あるいはより効率の良い処理系を実現するためには不可欠な問題である．したがって，メタ計算機能により，簡約戦略を動的に変更しうるような仮想簡約マシンの実現が興味深い課題の1つとなる．この1つのアプローチとして，処理系自体を書換え規則の集合として記述することが考えられる．

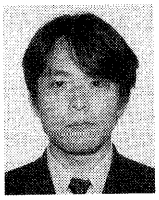
本論文では，動的なシステムの記述を目的にしてCREPSの操作的意味についてのみ議論した．したがって，メタ計算を導入した場合の数学的（代数的）意味については議論していない．このような仕様の数学的意味を厳密に定め，種々の解析手法により仕様の確認を厳密に行うことは，本論文の範囲を大きく超えるものであるが，今後の課題として非常に重要である．

謝辞 第1著者は本研究の一部に対し，文部省科学研究費補助金特別研究員奨励費による支援を受けている．第2著者は本研究の一部に対し，文部省科学研究費補助金（No.09650444）および北海道工業大学特別奨励研究助成金による支援を受けている．

参考文献

1) Maes, P.: Issues in Computational Reflection, *Meta-Level Architectures and Reflection*, Maes, P. and Nardi, D. (Eds.), pp.21–35, North-Holland (1988).
2) Smith, B.C.: Reflection and Semantics in Lisp, *Proc. ACM Symp. on Principle of Programming Languages*, pp.23–35 (1984).
3) Tanaka, J.: A Simple Programming System Written in GHC and Its Reflective Operations, *Proc. Logic Programming Conference'88*, pp.143–149, ICOT (1988).
4) Masuhara, H., Matsuoka, S., Watanabe, T. and Yonezawa, A.: Object-Oriented Concurrent Reflective Languages can be Implemented Efficiently, *Proc. OOPSLA'92*, pp.127–144 (1992).
5) Dershowitz, N. and Jouannaud, J.P.：書換え系，コンピュータ基礎理論ハンドブック II, pp.243–321, 丸善 (1994).
6) Bergstra, J.A. and Klop, J.W.: Conditional Rewrite Rules: Confluence and Termination, *Journal of Computer and System Sciences*, Vol.32, No.3, pp.323–362 (1986).
7) Toyama, Y.: Confluent Term Rewriting Systems with Membership Conditions, *Proc. Intern. Workshop on Conditional Term Rewriting Systems*, Lecture Notes in Computer Science, Vol.308, pp.228–241 (1987).
8) 渡部卓雄：書換えに基づく自己反映計算の定式化，信学技法，SS94-12, pp.1–7 (1994).
9) 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，信学技法，SS94-52, pp.49–56 (1995).
10) 栗原正仁，佐藤崇昭，大内 東：項書換えシステムにおける自己反映計算，コンピュータソフトウェア，Vol.12, No.4, pp.3–14 (1995).
11) ヘネシー，M.：プログラミング言語の意味論入門，サイエンス社 (1993).

(平成 9 年 12 月 18 日受付)
(平成 10 年 9 月 7 日採録)

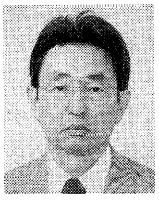


沼澤 政信（学生会員）

1969年生．1992年北海道大学工学部情報工学科卒業．1994年同大学院工学研究科情報工学専攻修士課程修了．現在，同博士後期課程在学中．システム情報工学，項書換え系などの研究に従事．電子情報通信学会，計測自動制御学会各会員．

栗原 正仁（正会員）

1955年生．1978年北海道大学工学部電気工学科卒業．1980年同大学院工学研究科情報工学専攻修士課程修了．現在，北海道工業大学電気工学科教授．工学博士．人工知能，項書換え系などの研究に従事．電子情報通信学会，日本ソフトウェア科学会，人工知能学会，EATCS（欧州理論計算機科学会）各会員．



大内 東（正会員）

1945年生．1974年北海道大学大学院工学研究科博士課程修了．現在，北海道大学工学部システム情報工学専攻教授．工学博士．システム情報工学，応用人工知能システム，医療システムなどの研究に従事．電子情報通信学会，日本ファジィ学会，計測自動制御学会，人工知能学会，電気学会，日本OR学会，医療情報学会，病院管理学会，IEEE-SMC各会員．