プログラミング－言語・基礎・実践－　7－2
（1992. 5. 29）

応答型並行プログラムの部分評価法

Partial Evaluation of Reactive Concurrent Programs

村上 昌己

Masaki Murakami

岡山大学 工学部

Faculty of Engneering, Okayama University

〒700 岡山市 津島中 3-1-1

3-1-1 Tsushima-naka, Okayama 700 JAPAN

e-mail: murakami@momo.it.okayama-u.ac.jp

Abstract

本稿では、環境と通信を続けながら継続的に処理を実行する応答型の並行プログラムの、部分評価の方法について述べる．本稿で述べる方法は、特殊化されるプロセスのコンティニュエーションに、次々と従来の従来型の部分評価手続きを適用し、その不動点を求めるものである．この方法は、プロセス融合の拡張概念と考えることができる．この方法は、CSP, CCS, actor, Concurrent Constraint 等様々な並行計算モデルに、一般的に適用できるものである．本稿では並行論理型言語 Nested Guarded Horn Clause (NGHC) で記述した応答型プロセスの部分評価法を、展開/畳み込み規則を用いて実現した例を示す．本稿ではさらに、混合計算の実現例について示す．

Abstract: In this paper, a method of partial evaluation for reactive concurrent processes is proposed. Our procedure is applying a conventional partial evaluation procedure to continuations of a reactive process and compute the fixpoint. This method can be regard as an extension of the process fusion method. The method presented here works for various concurrent models such as CSP, CCS, actors and Concurrent Constraint generally. In this paper, an example of the method for a concurrent logic programming language called Nested Guarded Horn Clause (NGHC)[Fa90] using unfolding/folding rules is presented. We also give an example of mixed computation.

1 はじめに

プログラムの部分計算法はプログラムをその実行環境についての情報を用いて特殊化し実行効率を向上させる手法であり [Fut87]、関数型/論理型を含めた様々な分野で多くの結果が報告されている [Jo91]. 特に論理型プログラムにおいては、展開/畳み込みの規則を用いて部分評価をする方法によってメタプログラミング手法の高速化の方法などが報告されている [Tak86]. 並行論理型プログラムの分野においても、[Fj88] によって [Fr88] の展開/畳み込みの手法の応用として、初期ゴールの入力代入についての制約を用いて特殊化を行なう部分評価手法が報告されている.

[Fj88] の方法は例えば $append(X,Y,Z)$ というゴールを $Y=[1,3,4]$ という代入情報を用いて特殊化するようなものであった. この例からわかるように、この方法は通常の逐次型プログラムと同様な変換型 (transformational) の実行のパラダイムに基づいたものであった. このような変換型の実行パラダイムにおいては、プログラムは実行を開始する前に入力を受け取り、以後は停止して出力結果を返すまで環境との相互作用を一切行なわない閉じた計算を行なうものである. したがって部分評価のために用いることができる実行環境に関する情報は、$Y=[1,3,4]$ のような初期入力値についての制約のみである.

一方、並行論理型言語における最も興味深い話題は、勿論応答型 (reactive) の並行プロセスの記述に関する話題である [Sh89]. 応答型のパラダイムにおいてはプログラムは、計算の開始時に入力を受け取るだけでなく、実行の途中において環境との間でメッセージの送受信して通信を行ないながら、処理を続ける開いた計算系である. したがって応答型のプログラムにおいては、初期入力だけでなく、実行の途中で受け取るメッセージについての制約もプログラムの特殊化に用いることができる可能性があり、これによってさらに実行効率の向上が期待できる. そのためには初期入力についての制約だけを情報として用いる変換型のプログラムについての部分評価手法では、不十分であることがわかる.

本稿では、環境と通信を続けながら継続的に処理を実行する応答型のプロセスについて、実行の途中に受け取るメッセージについての情報も用いて特殊化するような部分評価及び混合計算の手法について述べる. 本稿で述べる方法は、特殊化されるプロセスのコンティニュエーションに、次々と従来の従来型の部分評価手続きを適用し、その不動点を求めるものである. この方法は、プロセス融合の一般化と考えることができる. この方法は、CSP, CCS, actor, Concurrent Constraint 等様々な並行計算モデルに、一般的に適用できるものである. 本稿では並行論理型言語 Nested Guarded Horn Clause (NGHC) で記述した応答型プロセスの部分評価法を、展開/畳み込み規則を用いて実現した例を示す.

本稿の構成は以下のようになっている. 第 2 章では応答型の並行プロセスの実行環境について考え、これらについての情報を用いた特殊化の効果について述べ. このような情報を有効に用いた特殊化を行なうには変換型のプログラムの部分評価手法を直接用いたのでは上手くゆかないことを示す. 第 3 章では、変換型プログラムの部分評価法の問題点を解決した、応答型プロセスの新たな部分評価法について、一般的な枠組みで述べる. 第 4 章では、具体的に NGHC の部分評価法の、展開/畳み込みを用いた実現を示す. 第 5 章では、まとめと関連した研究について述べる.

2 応答型プロセスの実行環境と従来の方法の限界

2.1 応答型プロセスの実行環境

応答型あるいは開放型の並行プロセスの記述方法は CSP[Ho85], CCS[Mil89], actor [Agh86] 等が知られている. このような様々なモデルによって記述される並行プロセスに共通した姿は、通信方式が同期・非同期であるにかかわらず、入出力のためのチャネルを持ち、入力チャネルへのメッセージを受理することにより、出力チャネルへ応答を返し、また次の世代のプロセス[1]を生成し、自らは消滅するものと考えることができる. 通信一対一のシェークハンドであると仮定した場合は CCS に対応するモデルとなる. 一方通信を非同期であるとし、次の世代のプロセスにひとつが親の address を継承する場合を考えれば、actor もこの枠組みの特殊な場合と考えることができる.

[1] 通常、並行論理型言語の世界ではプロセスとはテイル・リカージョン等を用いて同じ名前のゴールを呼び出して実現することが多い. すなわち、プロセスとは数世代にわたって同じ形を継承し続ける安定した形のゴールをさすことが多い. 一方 CSP ではプロセスとは、動作を行なうたびに消滅し、新たな別のプロセスが生成されるものと考える. すなわち CCS におけるエージェント、actor における behavior の概念に近いものである. 本稿ではプロセスとは、この意味で用いる. 実際に並行論理型言語によって記述した際には、プロセスの概念は殆どゴールの概念と同じものとなる. 一方、世代を越えて安定して存在するものとしてのプロセスを差す場合、本稿では応答型プロセス又は応答型の並行プロセスと呼んで区別する.

このような応答型の並行プロセスの実行環境とは、各チャネルを通じてやりとりをする通信相手のプロセス、又はその通信されるメッセージの系列の集合と考えることができる．このようなメッセージの系列の集合についての制約としては、様々なものが考えられる．従来の変換型のプログラムの部分評価における「複数の入力のうち一部が既知である」という仮定に対応するのは、あるチャネルを通じてやりとりされるメッセージの系列が既知である、という仮定に対応するものと考えることができる．本稿では、このような制約について考える．

2.2 応答型プロセスにおける部分評価の効果

CSP 風の定式化によれば、プロセスとはメッセージを受け取り、その出力チャネルへの応答メッセージとコンティニュエーションであるプロセスの組を返す関数と考えることができる[2]．以下ではプロセス Pのメッセージの組$\overline{m}$によるコンティニュエーションを $cont(P,\overline{m})$ と、またPの$\overline{m}$に対する応答メッセージを $resp(P,\overline{m})$ 表記する．すなわち:

$$P(\overline{m}) = \langle cont(P,\overline{m}), resp(P,\overline{m})\rangle$$

と表わされる．

このような個々のメッセージの組が与えられた後でのステップにおいて、プロセスが入力チャネルに与えられたメッセージを用いて、出力チャネルへの応答及び子孫のプロセスを計算する動作は、変換型のプログラムの実行と考えることができる．

したがって、$\overline{m}$を受け取って $cont(P,\overline{m})$ 及び $resp(P,\overline{m})$ を計算するステップは、変換型プログラムの場合と同様に$\overline{m}$の一部を固定することによって、その実行を効率化することができるものと考えられる．

加えて、並行論理型プログラムのように$\overline{m}$の一部に対して出力を返す機能を持つモデルの場合、次のような効果も期待できる．$\overline{m}$ の一部について計算を前もって行なうことにより、$resp(P,\overline{m})$ の一部をあらかじめ作っておくことができる．このようにして作られた未完成メッセージは、そのメッセージのコンシューマとなるプロセスにとって、再び部分評価のために用いることができる．例えば多段のパイプラインのようにプロデューサ・コンシューマの関係で結ばれた多数の応答型プロセスが並列に走る場合、入力の一部について先に計算を進めておくことによって、通信先で使われる値を前もって送っておくことが可能となり、その結果実行時に行なわれる通信を節約でき、全体のスループットが向上する可能性も期待できる．

[2]本来の CSP では、出力動作も入力動作と同様に扱い、プロセスとは動作からプロセスへの関数である、という定式化をしているが、ここでは都合により、出力を別扱いする．

2.3 変換型部分評価手法の限界

本節では、応答型のプロセスを実行途中に受け取るメッセージについての情報を用いて部分評価しようとする際、従来の方法をそのまま適用したのでは上手くゆかないことを、並行論理型言語の例を用いて示す．

例えば次のような場合を考える．Pを二つの通信チャネル C_m及び C_nを持つ応答型のプロセスを考える．今 C_mを通じて送られてくるメッセージの列が $m_1, m_2, m_3, \ldots$ であることが既にわかっているものと仮定する．並行論理型プログラム場合、このようなメッセージを順に受け取る動作はプロセスを表現するゴール p(Cm, Cn,...) のチャネル変数 Cm を

```
Cm = [m1| Cm1]
Cm1 = [m2| Cm2]
Cm2 = [m3| Cm3]
...
```

のように具体化することによって実現される．

一般に、従来の変換型の部分評価手法を用いた場合、Pが起動された時点で得られている入力は m_1のみであり、Pは入力代入 Cm = [m1|Cm1] によってのみ特殊化され、m_2以降のメッセージは無視される．前節の述べたように、m_2以降のメッセージを利用してプロセスの特殊化をさらに進めることができれば、全体の実行効率はさらに向上することが期待される．しかしながら Pを $m_1, m_2, \ldots$ すべての情報を用いて特殊化することはそのままではできない．何故ならば、$m_1, m_2, \ldots$ というメッセージの無限な系列を与えることは、変数 C1 を [m1, m2, ...] という無限項に具体化するような代入を与えることに対応する．無限項を含むゴールについての部分評価は従来の方法では対応していない．

そればかりでなく、既知なメッセージの系列が有限である場合でも、従来の方法では問題が残る．例えば、Pを 2 つめのメッセージまでの情報を合成した代入

```
C1 = [m1, m2| C3]
```

によって部分評価することは一般には許されない．何故ならば応答型のプロセスにおいてはメッセージ同士の相対的な順序関係は動作の上で本質的だからである．次の例を考えよう．

```
p(Cm,Cn,O) :- Cm = [m1|C11], Cn = [n1|C21] |
          O = [o1|O1], p(C11,C21,O1).
p(C11,C21,O1) :-
             C11 = [m2|C12],C21 = [n2|C22] |
          O1 = [O|O2], p(C12,C22,O2).

p(Cm,Cn,O) :- Cm = [m1|C11], C11 = [m2|C12] |
        O = [o1|O1], O1= [1|O2], p(C1,C2,O2).
```

この例では、Cm 及び Cn に最初のメッセージ m_1, n_1 が到着した後、出力を待って次のメッセージ m_2, n_2 が到着すると仮定した場合、三番目の節が選択される可能性は無い．したがって入力メッセージの系列が $m_1, m_2, \ldots$ であった場合の O の第二要素は常に 0 となる．

このような例について [Fj88] で採用されている考え方に基づく方法、すなわち入力代入を与えた上で節の展開を繰り返す方法で Cm = [m1, m2| C3] を用いて部分評価した場合、これは m_1 の到着後、p が出力を出すのを待たずに m_2 が到着する可能性のある場合に対応する．すなわち:

```
p_1(C12,C2,O) :- C2 = [n1|C21] |
             O = [o1|O1], p_2(C12,C21,O1).
p_2(C12,C21,O1) :- C21 = [n2|C22] |
             O1 = [O|O2], p3(C12,C22,O2).

p_1(C12,C2,O) :- true |
             O = [o1|O1], O1= [1|O2],
             p_3(C12,C2,O2).
```

のようになる．このとき O の第二要素は 1 となる可能性がある．

このように従来の変換型の部分評価手法を応答型のプロセスに適用した場合、もともとのプロセスのセマンティクスを破壊する可能性もある．

3 応答型プロセスの部分評価法

3.1 応答型プロセス部分評価法の仕様

チャネル C_m への入力メッセージの系列 $M = m_1, m_2, m_3, \ldots m_i \ldots$ が既知であった場合、直観的には P の M による部分評価 $Part(P, M)$ とは、少なくとも C_n に与えられた任意のメッセージ系列に対して「応答が同じ」でなければならない．すなわち任意の i に対して:

$$resp(P[M_{i-1}, N_{i-1}], (m_i, n_i)) = resp(Part(P, M)[N_{i-1}], n_i) \quad \cdots *$$

とならなければならない．ここでメッセージの有限系列 $M_i = m_1, m_2, m_3, \ldots m_i, N_i = n_1, n_2, \ldots n_i$ について、P に M_i 及び N_i を与えた後のコンティニュエーション $cont(\ldots cont(P, (m_1, n_1)), (m_2, n_2)), \ldots, (m_i, n_i))$ を $P[M_i, N_i]$ と表記する．

3.2 メッセージの系列のよる部分評価

次に上の * を充たす関数を実際に求める方法の基本的な考え方について、以下に述べる．

二つのメッセージを受け取って 1 ステップ動作するプロセス P について、$part(P, m)$ によってプロセス P のメッセージ m による (従来から与えられた方法による) 部分評価を現わすものとする． すなわち

$$part(P, m)(n) = \langle cont(P, (m, n)), resp(P, (m, n)) \rangle$$

である．ここでメッセージの系列 M についての P の部分評価 $Part(P, M)$ は以下のように定義される．

$$Part(P, M) = part(\{(m, n) \rightarrow \langle Part(part(cont(P, (M_1, n)), resp(P, (M_1, n))), M'), resp(P, (m, n))\rangle\}, M_1) \quad \cdots\cdots **$$

ここで M_1, M' はそれぞれ系列 M の第一要素とそれを除いた残りの系列をあらわす[3]．

このように定められた部分評価が、任意の $m_1, m_2, \ldots$ 及び $n_1, n_2, \ldots$ の任意の長さ j の有限プレフィックスについて * を充たすことは、* の右辺の $Part$ を ** の右辺のよって j 回置き替えることによって示される．

** は、$Part(P, M)$ を実際に求める方法を与えている．すなわち、M が有限な系列であった場合、以下のような手順で求められる．

$Part(P, M)$:

Step 1 M が長さ 0 の系列ならば $Part(P, M) = P$ として終了する．そうでなければ **Step 2** へ．

Step 2 以下の結果を再帰的に求め、**Step 3** へ．

$$Part(part(cont(P, (M_1, n)), resp(P, (M_1, n))), M')$$

[3] $\{x \rightarrow t\}$ によって「x を t に写像する関数」を現わす．すなわち関数のグラフ表現である．

Step 3 Step 2 で求めた結果を、

$part(\{(m,n) \rightarrow \langle X, resp(P,(m,n))\rangle\}, M_1)$ の X に代入したプロセスを返し終了する.

以上の手続きは、従来の変換型の部分評価法 $part(P,m)$ 及び、Fをプロセス名とするとき $\{n \rightarrow F\}$ というプロセスを記述する構文を得る方法を用いていることに注意されたい. これらはCSP, 並行論理型言語等については、従来の結果を用いて得ることができる.

次に Mが有限でない場合を考える. 今 Mは任意の i 番目の要素を有限な手続きで求めることができると仮定する.

$Part(P,M)$:

以下の手続きで S(集合型) はグローバル変数で、初期値は$\emptyset$ とする.

Step 0 $\langle\langle call(P,M) : Part(P,M)\rangle\rangle$ という組を Sに加えて **Step 1** へ. ここで $call(P,M)$ は $Part(P,M)$ で定まるプロセスを呼び出す式.

Step 1 以下の対が Sに含まれれば、**Step 2** へ. そうでなければ **Step3** へ.

$$\langle\langle Call : Part(part(cont(P,(M_1,n)), resp(P,(M_1,n))), M')\rangle\rangle$$

Step 2 $part(\{(m,n) \rightarrow \langle Call, resp(P,(m,n))\rangle\}, M_1)$ を返し終了する.

Step 3 以下の結果を再帰的に求め、**Step 4** へ.

$$Part(part(cont(P,(M_1,n)), resp(P,(M_1,n))), M')$$

Step 4 Step 3 で求めた結果を、

$part(\{(m,n) \rightarrow \langle X, resp(P,(m,n))\rangle, M_1)\}$ の Xに代入したプロセスを返し終了する.

M自身もプロセスとして記述された場合、与えられた応答型プロセスの Mによる特殊化は、プロセス融合と解釈することができる. 次章では展開/畳み込みを用いて並行論理型言語のプロセス融合の例を示す.

4 NGHC を用いた例

本節では並行論理型言語Nested Guarded Horn Clause (NGHC) [Fa90] の構文とセマンティクスについて簡単に述べた後、NGHC の展開/畳み込みについて述べる.

4.1 NGHC の概要

NGHC は GHC の拡張であり、多くの点で GHC の特徴を受けついでいる. NGHC のプログラムの任意の節は m 層のガードをもつ. 各ガードは $I|O$ という形の対であり、I,O はそれぞれ入力制約、出力制約である. このふたつはコミット記号 (|) によって区切られている. 入力制約 Iは一方向単一化 (one way unification ゴール $t < s$ の有限集合である. また出力制約は通常の単一化ゴールの有限集合である.

NGHC のプログラムは次のような節の有限集合とゴール節$\overline{G} = G_1, \ldots G_k$の組である.

$$H : -I_1|O_1, \ldots, I_n|O_n \rightarrow B_1, \ldots, B_m.(n,m \geq 0)$$

NGHC の計算規則は基本的には Flat GHC のそれと同様なものと考えられる. 詳しくは [Fa90] を参照されたい.

4.2 NGHC プログラムの展開/畳み込み

[Fa90] で述べられた展開規則は以下のようなものである.

定義

c を NGHC 節集合 Wに含まれる次のような節であるとする。

$$c = p(Y_1, \ldots Y_k) : - \\ I_1|O_1, I_2|O_2, \ldots, I_n|O_n \rightarrow M_1, M_2 \ldots M_m.$$

c の Wでの展開 $Unf(c,m)$ とは次のように定義される節の集合である。

$ONE-STEP(c,w) = OLD(c,w) \cup NEW(c,w)$ ここで

$OLD(c,w) =$
$\{ p(Y_1, \ldots, Y_k) : -$
$I_1|O_1, \ldots, I_n|O_n, true|O_{call\_j}, I_1^j|O_1^j \rightarrow$
$G, M_1, M_2, \ldots, M_{j-1}, M_{j+1}, M_m.$
ここで
i) $M_j = p_j(t_1^j, \ldots, t_{s(j)}^j), 1 \leq j \leq n$
ii) $c_j = p_j(X_1^j, \ldots X_{s(j)}^j) : -$
$I_1^j|O_1^j, \ldots, I_{n(j)}^j|O_{n(j)}^j \rightarrow G_1^j, \ldots, G_g^j$
を w に含まれる節とする.
iii) $O_{call\_j} = \{X_1^j = t_1^j, \ldots, X_{s(j)}^j = t_{s(j)}^j\}$
iv) $n(j) > 1$ の場合、 $G = p_i'(Z_1, \ldots, Z_n)$
かつ p' を以下のような節によって
定まる新しい述語とする.

$p_i'(Z_1,\ldots,Z_n) : -I_2^j|O_2^j,\ldots,I_{n(j)}^j|O_{n(j)}^j$
$\rightarrow G_1^j,\ldots,G_g^j$
ここで $\{Z_1,\ldots,Z_n\} =$
$Var(I_2^j|O_2^j,\ldots,I_{n(j)}^j|O_{n(j)}^j \rightarrow G_1^j,\ldots,G_g^j)$.
$n(j)=1$ のとき $G = G_1^j,\ldots,G_g^j\}$.

$NEW(c,w) =$
$\{p_i'(Z_1,\ldots,Z_n) : -I_2^j|O_2^j,\ldots,I_{n(j)}^j|O_{n(j)}^j$
$\rightarrow G_1^j,\ldots,G_g^j$
ここで $p_i'(Z_1,\ldots,Z_n)$ は
$OLD(c,w)$ $iv)$ で導入された
新しい述語.$\}$

先の場合と同様に
$Unf(c,w) = MIN\{c_nor|c_nor$は c_int, の強標準形であり、 $c\_int \in ONE-STEP(c,W)\}$.

節集合 Pの節 c に展開規則を適用した結果を $Unf(c,P)$ と表記する． P を $\{c_1,\ldots,c_n\}$ とする．P の展開 $Unf(P)$ とは次のように定まる節の集合である．

$$Unf(P) = MIN(\cup_{i=1\ldots n} Unf(c_i,P))$$

ここで MINはNGHC節の集合 Sに対する種の簡約化で、主に冗長な節を取り除く操作からなる．

また畳み込み操作は展開の逆操作として容易に定めることができる．
定義 c を先の定義と同様にNGHC節集合 Wに含まれる節であるとする。このとき新たな述語 new_Pを以下の節 C_{new}によって定義する．

$$new\_P(Z_1,\ldots Z_l) : -true|true \rightarrow M_1',M_2'\ldots M_m'.$$

ただしあるθについて $(M_1',M_2'\ldots M_m')\theta =$
$(M_1,M_2\ldots M_m)$、$Z_1,\ldots Z_l$は $M_1',M_2'\ldots M_m'$に出現する変数である．

c の C_{new}による畳み込み $Fold(W,c,C_{new})$ とは、W から c を次の節で置き換えて得られる節の集合である．

$p(Y_1,\ldots Y_k) : -$
$I_1|O_1,I_2|O_2,\ldots,I_n|O_n \rightarrow new\_P(Z_1,\ldots Z_l)\theta$.

4.3 $Part(P,M)$ の展開/畳み込みによる実現

NGHCについて前章で述べた $Part(P,M)$ は展開/畳み込みを用いて実現することができる．以下では

1. NGHC のプログラム $P = (W,\overline{G})$ のメッセージ m による部分評価とは、$\overline{G}$の入力変数 Xを $X = [m|X_1]$ で具体化して、展開/畳み込みによって特殊化することによって実現される．

2. プログラム $P = (W,\overline{G})$ についての、メッセージ m についてのコンティニュエーション $cont(P,m)$ は $P' = (W \cup W',\overline{G'})$ で与えられる．ここで$\overline{G'}$及びW'は、以下のように定まる．$\overline{G}$の入力変数Xを $X = [m|X_1]$ で具体化して実行した際、$\overline{G}$の i番目のゴール G_iが、$H : -I_1|O_1,I_2|O_2,\ldots,I_n|O_n \rightarrow B$という形の節の $j-1$番目のガードまで解いてサスペンドしているとする．そのとき
$H : -I_1|O_1,\cdots,I_{j-1}|O_{j-1}$という形のプレフィックスを持つすべての節について、j番目のゴールから以降のポストフィックス:$I_j|O_j,\cdots,I_l|O_l \rightarrow \overline{B}$ 及び新たな述語記号と $H : -I_1|O_1,\cdots,I_{j-1}|O_{j-1}$ に出現する変数を引数に持つヘッド H_i'から新たな節 $H_i' : -I_j|O_j,\cdots,I_l|O_l \rightarrow \overline{B}$を作り、$W'$に加える．また$\overline{G'}$については、$\overline{G}$の i番目のゴールを、$j \neq l$の場合は $H_i'\theta$で、$j = l$の場合は$\overline{B}\theta$で置きかえて得られるゴールとする．ここで $theta$ は $H : -I_1|O_1,\cdots,I_{j-1}|O_{j-1}$を解いて得られた解代入とする．

3. メッセージ m についての応答 $resp(P,m)$ は、$\overline{G}$の入力変数 Xを $X = [m|X_1]$ で具体化して実行したときの、各ゴールの出力ストリームの要素の列である．

4. 関数 $\{m \rightarrow \langle F,o\rangle\}$ を実現するプログラムは、次のような節とゴール $new\_F(I,O)$ によって実現される．

$$new\_F(I,O) : -I < [m|I_1]|O = [o|O_1] \rightarrow F$$

4.4 例題

ここでは三次元の動画の例を考える．平面上を自由に動きまわるロボットを考える．ロボットの位置は $X-Y$ 平面上の座標によって与えられる．このロボットを真上でなく、斜めから観測した様子をディスプレイに表示するプロセスを考えよう．

このような状況は二次元空間上の座標に一次変換をほどこすプロセスによって記述することができる．一次変換は以下のようなプログラムで記述できる．よく知られたシストリック・アレイによる行列の掛け算である．

```
unpack(Instream, X, Y)  :-
  Instream < [(Xin, Yin) | In] |
  X =  [Xin|X_rest], Y = [Yin|Y_rest]
  -> unpack(In, X_rest, Y_rest).
```

```
node_11(Value_11, In_X,
      Channel_11_12, Channel_11_21) :-
        In_X < [X|In_rest] |
        Channel_11_12 = [X|C_r],
        Channel_11_21 = [X*Value_11|C_d]
        -> node_11(Value_11,
                   In_rest, C_r, C_d).

node_12(Value_12, Channel_11_12,
        Channel_12_22) :-
          Channel_11_12 < [X|In] |
          Channel_12_22 = [X*Value_12|Out]
          -> node_12(Value_12,  In, Out).

node_21(Value_21, In_Y, Channel_11_21,
        Out_X, Chanel_21_22) :-
          In_Y < [Y|In_rest],
          Channel_11_21 < [S|C] |
          Out _X= [Y*Value_21+ S |O],
          Chanel_21_22 = [Y|C_r] ->
          node_21(Value_21,
                  In_rest, C, O, C_r).

node_22(Value_22, Chanel_21_22,
        Channel_12_22, Out_Y) :-
             Channel_21_22 < [X|In],
             Channel_12_22 < [S|C] |
             Out_Y = [X*Value_22 + S|O] ->
             node_22(Value_22, In, C, O)

pack(Out_X, Out_Y, Outstream) :-
           Out_X < [X|Ox], Out_Y < [Y|Oy] |
           Outstream = [(X, Y)|O_rest] ->
           pack(Ox, Oy, O_rest)

trans(Instream, Outstream, A11,
      A12, A21, A22) :- true |true ->
         unpack(Instream, X, Y),
         node_11(A11, X, Channel_11_12,
                 Channel_11_21),
         node_12(A12, Channel_11_12,
                 Channel_12_22),
         node_21(A21, Y, Channel_11_21,
                 Out_X, Chanel_21_22),
         node_22(A22, Chanel_21_22,
                 Channel_12_22, Out_Y),
         pack(Out_X, Out_Y, Outstrream).
```

今、ロボットの軌跡が完全にプログラムされていて既にわかっている状況を考える．ロボットが仮に$X = Y$という直線上を一定速度で移動するとする場合、そのような軌跡を記述するプロセスは以下のようになる．

```
practice_move(Instream) :-
    true | true -> move_NE((0,0), Instream).

move_NE(Start, M) : - Start < (X, Y)  |
M = [(X+1, Y+1) |M_rest] ->
move_NE((X+1, Y+1), M_rest).
```

このとき斜めから観測した座標を出力するプロセスは以下のようなものになる．

```
p_new(Outstream,
      A11, A12, A21, A22) :- true| true
      ->  practice_move(Instream),
          trans(Instream, Outstream,
                A11, A12, A21, A22) .
```

これを展開/畳み込みによって変換した結果は以下のようになる．

```
p_new(Outstream, A11, A12, A21, A22) :-
  true  | true -> r_new(0, 0, A11, A12,
                   A21, A22, Outstream).

r_new(X1, Y1, A11, A12,
      A21, A22, Outstream) :- true |
          Outstream =
          [(Y1*A21+ X1*A11,
            Y1*A22 + X1*A12)|O_rest] ->
           r_new(X1+1, Y1+1, A11,
                 A12, A21, A22, O_rest)
```

以下の例はプロセスに与えられる入力メッセージが常に部分的に既知であるが、必ずしも決定性のプロセスによって記述できない場合について扱った例である．メッセージの未知の部分のみを入力とするプロセスに変換している混合計算の例といえる．

先の例と同様に二次元平面上を移動するロボットを考える．今、ロボットの故障でＹ成分の移動が全く不可

能となってしまったが、X方向には操縦者の意志のままに自由に動くことができるものとする．このとき、ディスプレイに表示するためのプロセスは、X座標のみを受けとってY成分に定数を与えて一次変換してやれば十分である．

```
move(Xin, Out) :- Xin < [X|Xrest] |
                  Out = [(X, 0) |O1] ->
                  move(Xrest, O1).
```

このような状況は以下の節で定義されるプロセスを求めることになる．

```
new_P(Xin, Outstream,
      A11, A12, A21, A22) :- true | true ->
          move(Xin, XY),
          trans(XY, Outstream,
                A11, A12, A21, A22).
```

展開/畳み込みによってプロセス融合した結果は以下のようになる．

```
new_P(Xin, Outstream, A11, A12, A21, A22)
     :- Xin < [X|Xrest] |
     Outstream = [(X*A11, X*A12)|O_rest]  ->
     new_P(Xrest, O_rest, A11, A12, A21, A22).
```

5 まとめ

本稿では応答型の並行プロセスを、実行途中に届けられる入力メッセージについての情報を用いて特殊化し実行効率を改善する方法について述べ、並行論理型言語NGHCの場合について展開/畳み込みを用いた実現について述べた．部分評価法の応用としてインタプリタの部分評価によるプログラムのコンパイル手法および部分評価プログラム自身の部分評価によるコンパイラの作成等が知られている [Fut87]．[Mur92] では本稿で述べた部分評価手法に適したメタインタプリタの記述について述べている．

謝辞: 岡山大学山崎進教授以下知能情報処理講座の皆様には、有益な議論と研究上の御支援をいただきました．記して感謝します．

参考文献

[Agh86] G. Agha, Actors, A model of Concurrent Computation in Distributed Systems, MIT Press (1986)

[Jo91] N. D. Jones, Partial Evaluation and the Generation of Program Generators, submitted to CACM. (Workshop on Partial Evaluation. at Kyoto University, Nov. 1991)

[Fa90] M. Falaschi, M. Gabbrielli, G. Levi, M. Murakami, Nested Guarded Horn Clauses: a language provided with a complete set of Unfolding Rules, Int. Jour. of Foundation of Comp. Sci., Vol. 1, No. 3, pp 249-263 (1990)

[Fj88] H. Fujita, A. Okumura, K. Furukawa, Partial Evaluation of GHC Programs based on The UR-set with Constraints, Proc. of the 5th Logic Programming Symp. (1988)

[Fr88] K.Furukawa, A. Okumura and M. Murakami, Unfolding Rules for GHC Programs, Proc. of the Workshop on Partial and Mixed Computation, New Generation Computing, vol 6, No. 2,3, pp 143-157 (1988)

[Fut87] 二村:部分計算, プログラム変換, 第4章, 知識情報処理シリーズ第7巻, 共立出版 (1987)

[Ho85] C.A. Hoare, Communicating Sequential Processes, Prentice Hall (1985)

[Mil89] R. Milner, Communication and Concurrency, Prentice Hall (1989)

[Mur92] 村上, 並行論理型言語におけるメタインタプリタの一構成法, 情報処理学会, プログラミング - 言語・基礎・実践 - 研究会, (1992)

[Sh89] E. Shapiro, The Family of Concurrent Logic Programming Languages, ACM Computing Surveys, vol. 21, no. 3, pp 413-510, (1989)

[Tak86] A. Takeuchi, K. Furukawa, Partial Evaluation of Prolog Programs and its Application to Meta Programming, Proc. of IFIP '86 Congress, North-Holland, pp.415-420 (1986)

[Fu85] K. Furukawa, K. Ueda, GHC Process Fusion by Program Transformation, 日本ソフトウェア科学会第2回大会論文集, 3B-4 (1985)