コンティニュエーションのデータ化を用いた探索アルゴリズムの並列化

渡辺 一郎　　村上 昌己

岡山大学 工学部

e-mail: {ichiro, murakami}@momo.it.okayama-u.ac.jp

要約

本稿では Prolog で記述された探索プログラムの並列化手法について述べる．逐次型 Prolog で記述された実際的なプログラムの多くは、純粋な Horn 論理型プログラムとして並列導出を行なった場合必ずしも意図した解が求められる保証はなく、正しく並列に実行するには節の探索規則と導出規則を考慮して並列に実行する必要がある．ここで述べる手法は、与えられたプログラムに対してプログラム変換を適用し、実際の実行と将来実行されるコンティニュエーションの部分計算を並列に行なうプログラムを得るものである．ここでは、Prolog のプログラムからコンティニュエーションを部分計算のデータとして持つ並列プログラムに変換する手法と、これ実行するために並列導出の規則を与える．多くの場合、Prolog によって記述された探索プログラムは、バックトラックによって解の探索を行なうため、コンティニュエーションは and/or tree 状の形となる．本稿で与える手法は、この tree 状のコンティニュエーションを並列に部分計算を行なうプロセスの集まりに変換する．

1　はじめに

Horn 論理型言語 Prolog は、探索問題に向くプログラミング言語として、多くの応用分野で問題の解法や探索アルゴリズムの記述に用いられ、多くのプログラムが記述され残されている．これらのレガシー Prolog コードには、アルゴリズムの説明のため、あるいはラピッド・プロトタイピング的な目的のため記述されたものも多数あり、並列環境で効率的に実行することを考慮していないものもしばしばある．このため逐次型 Prolog で記述された実際的なプログラムの多くは、純粋な Horn 論理型のプログラムではなく、最左導出でかつプログラム節が上から順に探索されるという前提のもとで最初に発見された解を意図した解とする場合も多い．このようなプログラムは、純粋な Horn 論理型プログラムとして並列導出を行なった場合必ずしも意図した解が求められる保証はなく、正しく並列に実行するには節の探索規則とアトムの選択規則を考慮して並列に実行する必要がある．

本稿では、このような最左導出・逐次探索を前提として作成された Prolog プログラムを、正しく並列実行するための方法について述べる．ここで述べる手法は、与えられたプログラムに対してプログラム変換を適用し、実際の実行と並列に将来実行されるコンティニュエーションの部分計算を行なうプログラムを得るものである．すなわち直観的には

$$G_1, G_2, \ldots, G_n$$

のような Prolog のゴール節があったとき、G_1の実行途中に $G_2, \ldots, G_n$の部分計算を並列に進めるものであり、一種の投機的計算による並列化手法である．ここでコンティニュエーション部分を部分計算するためにデータとして扱う必要がある．このため本稿ではよく知られた Prolog のプログラム変換手法 [7] である以下ようなものに着目する．すなわち、

```
p(X, Z) :- q(X, Y), r(Y, Z).
```

という節があったとき、新たにコンティニュエーションを収容する引数 `W` を追加し

```
p(X, Z, W) :- q(X, Y, r(Y, Z, W)).
```

とする．またボディ部が空な節:

```
p(X).
```

については、

```
p(X, W) :- W.
```

のようにコンティニュエーションを呼び出す節に変換する．このような変換を行なっても、多くの実際的な Prolog 処理系で正しく動作し、大きく効率を損なうこともないことが知られている．

これをもとに、[8, 10] では以下のような並列化手法を提案した．すなわち

```
p(X, Z) :- q1(X, Y1), q2(Y1, Y2), q3(Y2, Z).
```

のような節に対して、これを以下のように変換する．

`p(X, Z, W) :- q1(X, Y1, W1) &` $unfold$`(q2(Y1, Y2, q3(Y2, Z, W)), W1).`

ここで G_1 `&` G_2 は、 G_1 と G_2 の導出を並列に行なうことを意味する．また $unfold$ は第一引数に与えられたゴールを部分計算した結果を第二引数に返す述語である．すなわち各述語に付加した引数にコンティニュエーションを直接書き込むのでなく、それを部分計算した結果を書き込むようにしている．

しかし [8, 10] で提案された手法には、部分計算されるゴールに出現する述語は決定性の述語でなければ効果が得られないという問題がある．すなわち非決定性の述語が出現し、部分計算の結果が一意に定まらなくなった時点で、部分計算は中断しなければならない．また $G_1, \ldots G_i, G_{i+1}, \ldots, G_n$ というゴールに非決定性の述語が含まれる場合、G_i のコンティニュエーションは必ずしも $G_{i+1}, \ldots, G_n$ だけでなく、バックトラックにより $G_1, \ldots, G_{i-1}$ の部分が再度実行される可能性もある．したがって $G_{i+1}, \ldots, G_n$ だけの部分計算を結果が一意な範囲だけで行なう [8, 10] の手法には制限が多い． 多くの Prolog プログラムがバックトラック機能を用いて探索を行なうことを前提にしていることを考慮すると、非決定性述語を含むプログラムを有効に並列化することは重要であると考えられる．

本稿では、非決定性の述語が出現する場合も含めて、最左導出，節の線形探索及びバックトラックを前提にした逐次 Prolog のプログラムに対して、Fleng [6] (又は flat GHC [3] のサブセット) を拡張した導出規則を持つ並行論理型プログラムへの変換手法を提案する．

2 PCP 論理型プログラムへの変換

本稿では逐次型 Prolog のプログラムを、PCP 論理型プログラムと呼ぶ形式に変換する．PCP 論理型プログラムの節の構文は、Horn 節に以下に定義される $trans_C$ を適用した形と定義される．直観的には PCP 論理型プログラムの節は以下のような構文をもつ．

$$A :- B_1 \ \& \cdots \& \ B_n$$

ここで $B_1 \ \& \cdots \& \ B_n$ は、各 B_iを並列に導出するゴール節をあらわす．節の頭部 A は、Prolog のアトムに以下に定義する $trans_A$ を適用して得られるもののインスタンスである．各リテラル B_i は、頭部と同様な形か、 $true(t_1, t_2, t_3)$ の形、$unfold(B', W')$ の形、又は $W = B$という形の単一化ゴールである．ここで W, W'は変数、t_1, t_2, t_3を項， B' を Prolog のアトムに $trans_A$を適用したもののインスタンスとするとき、Bは $true(t_1, t_2, t_3)$ 又は $unfold(B', W')$ の形である．

定義 2.1 Pを Prolog プログラムとする． $trans_A, trans_B, trans_C, trans_{unfold}, trans_G$ を以下のように定義するとき、Prolog プログラムから PCP 論理型プログラムへの写像 $trans_P$ を以下のように定義する．

$$trans_P(P) \stackrel{\text{def}}{=} \{trans_C(C) | C \in P\} \cup \{trans_{unfold}(C) | C \in P\}$$

1. A をアトム $p(t_1, \ldots, t_m)$ とする場合:

   $$trans_A(A) \stackrel{\text{def}}{=} p(t_1, \ldots, t_m, U, V, W)$$

   ここで U, V, Wは新しい変数である．以降、U を p の AND 部の変数、V を OR 部の変数、 Wを rename 部の変数と呼ぶことにする．

2. W_1 を変数、$B_n, \ldots, B_1$ をゴール節とする場合:

   $$trans_B(B_n, \ldots, B_1, W_1) \stackrel{\text{def}}{=} \begin{cases} unfold(trans_A(B_n), W_1)\{V/nil, W/\emptyset\} & (n = 1 \text{ のとき}) \\ unfold(trans_A(B_n), W_1)\{V/nil, W/\emptyset\} \\ \quad \& \ unfold(trans_B(B_{n-1}, \ldots, B_1, W_1), W_2) & (n > 1 \text{ のとき}) \end{cases}$$

3. C をプログラム節 $A :- B_n, \ldots, B_1.$ とするとき:

   $$trans_C(C) \stackrel{\text{def}}{=} \begin{cases} trans_A(A) :- true(U, V, W). & (n = 0 \text{ のとき}) \\ trans_A(A) :- trans_A(B_n). & (n = 1 \text{ のとき}) \\ trans_A(A) :- trans_A(B_n) \ \& \ trans_B(B_{n-1}, \ldots, B_1, W). & (n > 1 \text{ のとき}) \end{cases}$$

   $$trans_{unfold}(C) \stackrel{\text{def}}{=} \begin{cases} unfold(trans_A(A), W') :- W' = true(U, V, W). & (n = 0 \text{ のとき}) \\ unfold(trans_A(A), W') :- W' = unfold(trans_A(B_n), W'). & (n = 1 \text{ のとき}) \\ unfold(trans_A(A), W') :- W' = unfold(trans_A(B_n), W') \\ \quad \& \ trans_B(B_{n-1}, \ldots, B_1, W) & (n > 1 \text{ のとき}) \end{cases}$$

ここで $n=0$ のとき $trans_A(A)$ に付け加えられる AND 部の変数は U、OR 部の変数は V、 rename 部の変数は W とする．また $n=1$ のとき、$trans_A(A)$ と $trans_A(B_n)$ に付け加えられる AND 部, OR 部 及び rename 部の変数は、それぞれ同じものとする．また $n>1$ のとき、$trans_A(B_n)$ に付け加えられる AND 部の変数は Wとし、 $trans_A(A)$ と $trans_A(B_n)$ に付け加えられる OR 部及び rename 部の変数はそれぞれ同じものとし、$trans_A(A)$ に付け加えられる AND 部の変数と $trans_B(B_{n-1},\ldots B_1)$ を求める際に $trans_A(B_1)$ で付け加えられる AND 部の変数は同じものとする．

4. G をゴール節 : $-B_n,\ldots,B_1.$ とするとき:

$$trans_G(G) \stackrel{\text{def}}{=} \begin{cases} trans_A(B_n)\{U/nil, V/nil, W/\emptyset\} & (n=1\text{のとき}) \\ trans_A(B_n)\{V/nil, W/\emptyset\}\ \&\ trans_B(B_{n-1},\ldots,B_1,W_1) & (n>1\text{のとき}) \end{cases}$$

ここで $n=1$ のとき、 $trans_A(B_n)$ に付け加えられる AND 部の変数を U、OR 部の変数を V、 rename 部の変数を Wとする．$n>1$ のとき、 $trans_A(B_n)$ に付け加えられる AND 部の変数を W_1、 OR 部の変数を V、rename 部の変数を Wとする、 $trans_A(B_n)$ に付け加えられる AND 部の変数は Wである．

3 PCP 論理型プログラムの導出規則

本節では PCP 論理型プログラムの導出規則を定義する．

定義 3．1 ゴール節中のアトム G_j と プログラム節 C : A: $-B_n\ \&\cdots\&\ B_1$に対して、ある代入 θが存在し $G_j\theta = A\theta$ となるとき、G_jに Cが適用できるという．またこのとき

$$derive_1(G_j, C) \stackrel{\text{def}}{=} (B_n\ \&\cdots\&\ B_1)\theta'$$

とする．ここでθ' は A と G_jの最汎単一化代入である．
定義 3．2 ゴール節中のアトム $p(t_1,\ldots,t_m, Cnt_1, Cnt_2, Var)$ に対してプログラム P中で適用できる節を上から順に $C'_n,\ldots,C'_1$とするとき、

$$derive_2(p(t_1,\ldots t_m, Cnt_1, Cnt_2, Var), \langle C'_n,\ldots,C'_1\rangle)$$

$$\stackrel{\text{def}}{=} \begin{cases} derive_1(p(t_1,\ldots t_m, Cnt_1, Cnt_2, Var), C'_n) & (n=1\text{のとき}) \\ derive_1(p(t'_1,\ldots t'_m, Cnt_1, W, Var\cup Var'), C'_n) & \\ \ \&derive_2(unfold(p(t_1,\ldots t_m, Cnt_1, Cnt_2, Var), W), \langle C''_{n-1},\ldots,C''_1\rangle) & (n>1\text{のとき}) \end{cases}$$

ここで、 $t'_1,\ldots t'_n$ は $t_1,\ldots t_n$ に変数が出現する場合、それらの変数を新しい変数で置き換えたもの、また

$$Var' \stackrel{\text{def}}{=} \{X_i = X'_i | X_i\text{は } t_1,\ldots t_n\text{に出現する変数で、 } X'_i\text{はそれらと置き換えた } t'_i\text{の新しい変数.}\}$$

とする．また $C'_i(1\le i\le n-1)$ について、元の Prolog プログラム中のある節 C_i に対して $C'_i = trans_C(C_i)$ のとき、$C''_i = trans_{unfold}(C_i)$ とする．

同様にゴール節のアトム $unfold(p(t_1,\ldots t_m, Cnt_1, Cnt_2, Var), W)$ に対してプログラム P 中で適用できる節を上から順に $C''_n,\ldots,C''_1$とするとき、$derive_2(unfold(..))$ は以下のように定義する．

$$derive_2(unfold(p(t_1,\ldots t_m, Cnt_1, Cnt_2, Var), W), \langle C''_n,\ldots,C''_1\rangle)$$

$$\stackrel{\text{def}}{=} \begin{cases} derive_1(unfold(p(t_1,\ldots t_m, Cnt_1, Cnt_2, Var), W), C''_n) & (n=1\text{のとき}) \\ derive_1(unfold(p(t'_1,\ldots t'_m, Cnt_1, W_1, Var\cup Var'), W), C''_n) & \\ \ \&derive_2(unfold(p(t_1,\ldots t_m, Cnt_1, Cnt_2, Var), W_1), \langle C''_{n-1},\ldots,C''_1\rangle) & (n>1\text{のとき}) \end{cases}$$

定義 3．3　PCP プログラムの導出規則を以下のように定義する．規則は以下の順に試みられるものとする．ゴール節 G_i を

$$A_1\ \&\cdots\&\ A_{j-1}\ \&\ A_j\ \&\ A_{j+1}\ \&\cdots\&\ A_k$$

とするとき、計算規則でアトム A_jを選択したとき:

[組み込み述語 $true$ の導出] :

1. A_j が $true(nil, Cnt, Var)$ のとき、それまでに求めた解代入$\theta_1 \cdots \theta_i$ と Var との間で矛盾がないかを調べる．矛盾がなければ G_{i+1} は nil となり、解代入に Var を加える．矛盾があれば, Cntが nilでなければ G_{i+1} は

$$A_1 \ \& \cdots \& \ A_{j-1} \ \& \ Cnt \ \& \ A_{j+1} \ \& \cdots \& \ A_k$$

となる．Cntが nilならば G_{i+1} は $fail$ となる．

2. A_j が $true(fail, Cnt, Var)$ のとき、G_{i+1} は

$$A_1 \ \& \cdots \& \ A_{j-1} \ \& \ Cnt \ \& \ A_{j+1} \ \& \cdots \& \ A_k$$

となる．

3. A_j が $true(Cnt, nil, Var)$ 又は $true(Cnt, fail, Var)$ のとき、それまでに求めた解代入$\theta_1 \cdots \theta_i$ と Var との間で矛盾がないかを調べる．矛盾がなければ G_{i+1} は

$$(A_1 \ \& \cdots \& \ A_{j-1} \ \& \ Cnt \ \& \ A_{j+1} \ \& \cdots \& \ A_k)Var$$

となり、解代入に Var を加える． 矛盾があれば, G_{i+1} は $fail$ となる．

4. A_j が $true(true(Cnt_1, nil, Var_1), Cnt_2, Var_2)$ 又は $true(true(Cnt_1, fail, Var_1), Cnt_2, Var_2)$ のとき、G_{i+1} は

$$A_1 \ \& \cdots \& \ A_{j-1} \ \& \ true(Cnt_1, Cnt_2, Var_1 \cup Var_2) \ \& \ A_{j+1} \ \& \cdots \& \ A_k$$

となる．

5. A_jが $true(true(Cnt_1, Cnt_2, Var_1), Cnt_3, Var_2)$ のとき、G_{i+1} は

$$A_1 \ \& \cdots \& \ A_{j-1} \ \& \ true(Cnt_1, true(Cnt_2, Cnt_3, Var_2), Var_1 \cup Var_2) \ \& \ A_{j+1} \ \& \cdots \& \ A_k$$

となる．

[組み込み以外の述語の導出]:

A_j を $p(t_1, \ldots, t_m, Cnt_1, Cnt_2, Var)$ (p はプログラムで定義された述語) とするとき:

1. Cnt_1 が $fail$のとき、G_{i+1} は、

$$A_1 \ \& \cdots \& \ A_{j-1} \ \& \ Cnt_2 \ \& \ A_{j+1} \ \& \cdots \& \ A_k$$

となる．

2. A_j に適用できる節があるとき、それらの節を上から順に $C_n, \ldots, C_1$とおく．このとき G_{i+1} は、

$$(A_1 \ \& \cdots \& \ A_{j-1} \ \& \ derive_2(p(t_1, \ldots, t_m, Cnt_1, Cnt_2, Var), \langle C_n, \ldots, C_1 \rangle) \ \& \ A_{j+1} \ \& \cdots \& \ A_k)\theta_{i+1}$$

となる．

3. A_j に適用できる節が無いとき、Cnt_2 が nil 又は $fail$ ならば G_{i+1} は $fail$ となる．それ以外のときは

$$A_1 \ \& \cdots \& \ A_{j-1} \ \& \ Cnt_2 \ \& \ A_{j+1} \ \& \cdots \& \ A_k$$

となる．

[述語 $unfold$ の導出]:

A_j を $unfold(p(t_1, \ldots, t_m, Cnt_1, Cnt_2, Var), W)$ (p は任意の述語) とするとき:

1. Cnt_1 が $fail$のとき G_{i+1}は

$$(A_1 \ \& \cdots \& \ A_{j-1} \ \& \ \ A_{j+1} \ \& \cdots \& \ A_k)\{W = Cnt_2\}$$

となる．

2. A_j に適用できる節があるとき、それらの節を上から順に $C_n, \ldots, C_1$とおく．このとき G_{i+1} は、

$$(A_1 \ \& \cdots \& \ A_{j-1} \ \& \ derive_2(unfold(p(t_1, \ldots, t_m, Cnt_1, Cnt_2, Var), W), \langle C_n, \ldots, C_1 \rangle) \ \& \ A_{j+1} \ \& \cdots \& \ A_k)\theta_{i+1}$$

となる．

3. A_j に適用できる節が無いとき、Cnt_2 が nil 又は $fail$ ならば G_{i+1} は

$$(A_1\ \&\cdots\&\ A_{j-1}\ \&\ \ A_{j+1}\ \&\cdots\&\ A_k)\{W = fail\}$$

となる．それ以外のときは

$$A_1\ \&\cdots\&\ A_{j-1}\ \&\ Cnt_2\ \&\ A_{j+1}\ \&\cdots\&\ A_k$$

となる．

上の導出規則を用いてプログラム節の集合が与えられたときゴール節に対する導出は通常のように定義される．プログラム節の集合 P に対してゴール節 G_0 の導出の有限列 $G_0, G_1, \ldots, G_k$ を $P \cup \{G_0\}$ の演繹列という． $P \cup \{G_0\}$ の演繹列で最後のゴール節が nil であったとき、この演繹列を $P \cup \{G_0\}$ の反駁という．

Pを与えられた Prolog プログラム P_0を変換して得られた PCP 論理型プログラム、 G を P_0のゴール節を変換して得られた PCP 論理型プログラムのゴール節とする．

4 変換及び実行の例

ここでは、変換後のプログラムがもとのプログラムのセマンティクスをどのように保存し、どのように並列実行が行なわれるかについて、以下の例のプログラム P とゴール G について考える．

P : $p(1).$ <1>
$p(2).$ <2>
$q(2).$ <3>

G : :- $p(X), q(X).$

このプログラムを逐次に実行すると、以下のようなステップをたどる．

1. $p(X)$ に節 <1> が適用され、$X = 1$ となる．
2. $q(X)$ に節 <3> の適用を試み失敗する．
3. $q(X)$ について他の選択枝を捜し失敗するので、$X = 1$ を破棄し、 $p(X)$ にバックトラックする．
4. 節 <2> が適用され $X = 2$ となる．
5. $q(X)$ に節 <3> が適用され成功し、終了する．

一方このプログラムを PCP プログラムに変換すると以下のようになる．

P' : $p(1, Cnt_1, Cnt_2, Var)$:- $true(Cnt_1, Cnt_2, Var).$ <1'>
$unfold(p(1, Cnt_1, Cnt_2, Var), W')$:- $W' = true(Cnt_1, Cnt_2, Var).$ <1">
$p(2, Cnt_1, Cnt_2, Var)$:- $true(Cnt_1, Cnt_2, Var).$ <2'>
$unfold(p(2, Cnt_1, Cnt_2, Var), W')$:- $W' = true(Cnt_1, Cnt_2, Var).$ <2">
$q(2, Cnt_1, Cnt_2, Var)$:- $true(Cnt_1, Cnt_2, Var).$ <3'>
$unfold(q(2, V, Cnt_1, Cnt_2, Var), W')$:- $W' = true(Cnt_1, Cnt_2, Var).$ <3">

$$G' : \quad :- p(X, W, nil, \emptyset)\ \&\ unfold(q(X, nil, nil, \emptyset), W). \tag{1}$$

これ以降，ネスト構造を見やすくするため，

$$p(X, q(X, Y, r(Y, nil, nil, \emptyset), nil, \emptyset), nil, \emptyset)$$

のようなアトムを以下のように表記することにする．

$$p(X,\ \bullet,\ nil, \emptyset)$$
$$q(X, Y,\ \bullet,\ nil, \emptyset)$$
$$r(Y, nil, nil, \emptyset)$$

まず (1) の左側のアトム $p(X, W, nil, \emptyset)$ について考える． このアトムには 2 つの節 <1'>, <2'> が適用可能である．組み込み述語以外のアトムの導出規則の 2. より、(1) の左側のアトムは

$$derive_1(\ \overset{\bullet}{p(X',W,W_1,\{X=X'\})},<1'>) \quad \& \quad derive_2(\ \overset{\bullet}{unfold(p(X'',W,nil,\{X=X''\}),W_1)},<2">) \tag{2}$$

となる．ここで & の左側は、$derive_1$ の定義より

$$true(W,W_1,\{X=X'\})$$

であり、ここで節の頭部とアトムとの単一化の結果 $\{X'=1\}$ が仮の解代入として得られる．右側は $derive_2(unfold..$ の定義より、$derive_1(unfold(p(X'',W,nil,\{X=X''\}),W_1),\ <2">)$ であり、結果的に

$$W_1 = true(W,nil\{X=X''\})$$

となる．このとき仮の解代入として $\{X''=2\}$ が得られる．したがって (2) の導出結果は、

$$true(W,W_1,\{X=X'\})\ \&\ W_1 = true(W,nil,\{X=X''\}) \tag{3}$$

となる． (3) の & の右側の単一化を実行することにより 左側のアトム中の W_1 が具体化され、

$$true(W,\ \overset{\bullet}{true(W,nil,\{X=X''\})},\{X=X'\}) \tag{4}$$

が得られる．

ここまでのステップで、直観的には (2) の & の左側は節<1'>を選択した実行を行なっている．ここで変数 W_1 は、元の Prolog のプログラムで <1> を選択した実行が失敗してバックトラックした際に実行すべきコンティニュエーションを指している．一方 & の右側は、バックトラックした後に <2> を選んだ場合の実行を、<2">を用いて並列に行っているものである．いずれも解代入は、それぞれ $X'.X''$ というローカルな変数への束縛に留められており、節の選択が確定するまで他のアトムから参照されることはない．このようにして 選択可能な節が複数存在する場合の OR 並列性を引き出している．

一方ここまでの導出と並列に、(1) の右側のアトム

$$unfold(q(X,nil,nil,\emptyset),W) \tag{5}$$

の導出を以下のように行なうことができる． このことは、ゴール節中の AND 並列性を引き出すことに対応する．

このアトムには <3"> が適用可能なので、導出規則の述語 $unfold$ の場合の 2 より (5) の導出結果

$$derive_2(unfold(q(X,nil,nil,\emptyset),W),\ <3">)$$

は、

$$W \ = \ true(nil,nil,\emptyset) \tag{6}$$

となる．(6) の単一化を実行することにより、(4) 中の変数 W が具体化され、ゴール節全体は

$$true(\ \overset{\bullet}{true(nil,nil,\emptyset)},\ \overset{\bullet}{true(\ \overset{\bullet}{true(nil,nil,\emptyset)},\ nil\ \{X=X''\})},\ \{X=X'\})$$

となる．このとき解代入として $\{X=2\}$ が得られる．続いて $true$ についての規則 4 を適用すると、以下のようになる．

$$true(nil,\ \overset{\bullet}{true(\ \overset{\bullet}{true(nil,nil,\emptyset)},nil,\{X=X''\})},\ \{X=X'\}) \tag{7}$$

以下では、ここまでに求められた解代入

$$\theta = \{X'=1,X''=2,X=2\}$$

と、(4) の rename 部に格納されたローカルな変数の情報 $\{X = X'\}$ 及び $\{X = X''\}$ との間で矛盾が無いかどうかが順次チェックされる． 解代入θ中の束縛 $X' = 1, X = 2$ と $\{X = X'\}$ は矛盾するので、このローカル変数への束縛を求めた計算は解ではないものとして破棄される． この過程は、 (7) に $true$ についての規則 1. を適用して以下のゴールを得ることにより実行される．

$$\begin{array}{l} true(\quad \bullet \downarrow , nil, \{X = X''\}) \\ \quad\quad true(nil, nil, \emptyset) \end{array} \tag{8}$$

一方、$X'' = 2, X = 2$ と $\{X = X''\}$ は矛盾しない．すなわち (8) に $true$ についての規則 3. を適用することにより

$$true(nil, nil, \emptyset)$$

となり解代入に $\{X = X''\}$ が加えられる．最後に $true$ についての規則 1. によりゴールは nil となり計算は終了する．最終的に得られる解は $X = 2$ となる．

上記の例は、変換及び導出規則の動作を説明するためのものであり、バックトラックが発生する最小規模の例となっている．そのため並列化によるステップ数の削減よりオーバヘッドの方が勝っていたため、有効性を示す例とはなっていない．しかし元の逐次 Prolog プログラムとの実行ステップ数の大小関係は、節の数及びゴールに出現するアトムの数が数個程度で逆転する．

5 議論

5.1 提案手法の正当性について

本稿で述べた変換手法の適用例として、いくつかの典型的な例 (文字列が回文であることを判定するプログラム等) について、元の Prolog プログラムと変換後の PCP 論理型プログラムの実行結果が等しくなることが確認済みである．しかしの正当性についての形式的証明は、現段階では確立されていない．

正当性の証明は、変換後の任意のプログラムについて以下の二つの事を示すことにより得られる．

- 元の Prolog プログラムの実行によって得られた解が、変換後の PCP プログラムのある実行によって得られること．
- 上に定義した導出規則を用いて求められた任意の反駁について、最終的に求められた解代入が、元の Prolog プログラムの節の探索戦略を上から下への順序とし最左導出を用いて求められた解代入に等しいこと．

先に提案したバックトラックを考慮しない場合の変換手法 [8] については、条件付きで上記の結果に対応する結果が証明されている． そこでは導出の長さに関する数学的帰納法を用いている．本稿で述べた場合についても、同様な論法により証明が可能であろうと予想される．ここで証明の際に注意すべきこととして、ここで対象とする実行はバックトラックをともなうため、長さ n の導出が 長さ $n-1$ の導出とひとつのプログラム節から帰納的に定義されるとは限らないことがある．

5.2 提案手法の有効性について

ここで扱う問題は、リテラルの選択規則を最左規則として各枝をプログラム節と同じ順序に左から配置した SLD 木を考えたとき、最も左にある成功枝を探索する問題となっている．この問題では、最も左の解に至るパスの左側のすべてパスの終点が解でないことを確認しなければならない．したがって、意図した解に至るパスより左側の各パスに対応する計算を OR 並列に実行するとしたとき、ひとつのパスを実行するコストのうち最も大きいものが、この問題のコストのひとつの下界となる． [1] ここでは、この下界に対してどの程度のオーバヘッドがあるかを考察することにより、提案手法の評価を与える．

前節の例からわかるように、提案した手法では各パスについて葉に至るまで導出を実行した後に、葉が解であるかを逐次的にチェックしている．したがって最悪の場合、上述の下界に対して解の左側にある葉の数に比例するオーバーヘッドが生じている．これは、先に述べた下界に比べて、無視できない大きな値である．すなわちパスの導出のコストがパスの長さだけかかったとしても[2]、せいぜい SLD 木の高さ程度である．一方その後に実行される葉の逐次的チェックの部分は、最悪の場合葉の数、すなわち SLD 木の高さに比べて相当に大きな値をとることになる．した

[1] この手法では AND 並列性を抽出しているため、個々のパスの実行もパスをいくつかの部分に分割して並列に行なっているが、この場合も以下の議論は同様である．

[2] これは、導出を逐次に行なっている場合で、$true(..)$ の形となるまでの導出のコストが最も大きくなると考えられる．

がって提案手法は、このままでは問題に対して有効な手法とはいえない．

提案手法が大きなオーバヘッドを持つ可能性がある原因は主に、各リテラルをプログラム節を用いて展開するフェーズがすべてのパスで終了した後に、$true$ に至った後に各葉をチェックするフェーズを開始していること．また葉のチェックが、各パスについて逐次に行なわれていることによる． このような問題は、 $true$ に関する導出規則を改良することにより、比較的容易に解決できるものと考える．

すなわち $true$ に関する規則の 3 において解代入と rename 部の無矛盾性をチェックしている．これはその $true$ に対応する葉が解であるかどうかの確認となっている．この規則は、OR コンティニュエーションが nil になった後に、ローカルな変数への束縛を解として公開するかどうかの判断をするために初めて適用される．したがってこのパス自体の計算が終了しても、自分より右側のパスの計算が終了するまで待たされることになる． しかしながら、あるパスの葉が解でないことの判定は、他のパスの導出の終了を待つ必要はない．これら規則を、OR コンティニュエーションが nil になる前に矛盾を検出できるよう改良することで、このパスが解でないことの確認が他のパスの導出と並列に実行できると考える．

また前節の例において、 (3) のような形のゴールの& の右側の $W = true(..)$ のようなアトムは、既に導出が終わったパスが自分より左のパスの終了を待っている状態と考えることができる．この場合も、このパスの結果が解でないこと (他のアトムの導出の結果得られた束縛と矛盾があること) の検出は、他のパスの導出の終了を待つ必要がない．したがって $W = true(..)$ の形のゴールの右辺に対して $true$ についての導出規則と同様な実行規則を追加することにより、この問題を回避することが考えられる．

5.3 実装について

本稿で提案した手法は、PCP 論理型プログラムという形式に変換するものである．PCP 論理型プログラムの実装方法としては、Fleng[6] や KLIC [1] などの実装済みの committed choice 型の並行論理型言語へのコンパイルによる方法が有力であると考える．PCP 論理型プログラムの committed chioce 型言語へのコンパイル技法の開発にあたって、特に注意を要することが予想されるのは、以下の 2 点である．

まず PCP 論理型プログラムの操作的意味論は 3 節に示した通り、Fleng や GHC のような典型的な committed chioce 型言語と異なり、ひとつのアトムの導出に同じ述語を定義する複数の節を同時に適用している．このため PCP 論理型プログラムの committed chioce 型言語コンパイルは、節の対応がソース・プログラムとターゲット言語でのプログラムでは一対一とはならない．

また第 2 点目として、データ化されたコンティニュエーションを直接アトムとして呼び出す機能は、多くの Prolog 処理系においては問題無く動作するが、現在の実際的な committed chioce 型言語の処理系においてどの程度保証されるかも注意が必要と考える．

また提案方式では、リテラルの導出のスケジューリングが大きく効率を作用することが判明している．例えば 4 節で述べた例では、バックトラックをともなう動作を説明したため、実行ステップ数がこの例としては比較的多いステップ数を要した場合となっている．同じプログラムと同じゴール (1) に対して、もし (1) の右側のアトム $unfold(q(X, nil, nil, \emptyset), W)$ が先に導出された場合、$unfold$ に関する規則の 2. と節 <3"> より、次のゴール節は

$$:- \quad p(2, true(nil, nil, \emptyset), nil, \emptyset).$$

となる．この時点で変数 X に対する解代入は $X = 2$ と確定する．<2'> より

$$:- \quad true(true(nil, nil, \emptyset), nil, \emptyset).$$

が次のゴール節として得られる．さらに $true$ についての導出規則を 2 回適用することにより、ゴールは成功し $X = 2$ が実際に解であったことが判かる．

このスケジューリングの場合、OR 分岐の無い決定性の述語の導出を先に行なうことにより、変数 X に対する束縛が早い時点で確定するため、この値が解とした実行が成功するかどうかだけを確認するだけで実行が終了する．したがって、解に至るパスの左のパスについて解でないことを確認する手数が大幅に削減されることにより、実行ステップ数が減少している．

このようにスケジューリングによって実行ステップ数が変化することを考慮して、実装を行なうことにより、5.3 節で述べたオーバヘッドはある程度回避することが可能であることも期待できる．

5.4 cut: "!" の導入について

従来の論理型プログラムによる探索アルゴリズムの並列実行に関する研究の代表的なものは、Horn 節集合で記述された問題の全解探索の問題であろう．例えば、committed choice 型言語による全解探索アルゴリズムの記述の技法 [9, 2] の研究、又は演繹データベースにおける仮想リレーションの検索手法 [4] などが報告されている．これらの技法

は、純粋な Horn 論理型言語の並列導出を、それぞれ固有のセマンティクスを持つ言語で効率的に行うための技法といえる。

一方本稿で扱っている問題は、Horn 論理型プログラムのセマンティクスの上の解を全解探索するものではなく、ある導出手続きで最初に求められる一部のみを意図した解とする問題といえる．すなわち、純粋な Horn 論理型プログラムのセマンティクスではその正当性が示せない Prolog プログラムを正しく並列導出することといえる．このように純粋でない Prolog プログラムの代表が、cut: “!” を含むものであろう．特にプログラムの正当性が探索順序に依存する Prolog コードの多くは、cut が多用されているのが実状である．このような事態を考慮すると、cut を含むプログラムの扱いを可能とすることは、本研究にとって重要な課題であると考えられる．

提案手法に cut の扱いを取り入れる手法としては、以下のような規則の拡張が考えられる． Prolog のソース文では cut は 0 引数のアトムと考えられる．したがって変換後は、コンティニュエーションと rename 部だけを引数とするアトムとなる． 例えばこのような cut が PCP 論理型プログラムのゴール中に $..p(.....X,V,W)\&unfold(!(..),X)$ という形で出現したとき、この cut がもとの Prolog プログラム中での cut と同等な機能を持つということは、Vに格納された OR コンティニュエーションを破棄する効果を持つということになる．したがって $unfold(!(..),X)$ が実行された際には、$p(.....X,V,W)$ に対してそのことを通知する必要がある．この通信のために両方のアトムに新たな引数を追加して、$p(.....X,V,W)$ 中のこの引数が具体化されたとき、OR 部の内容を nil とする規則を追加する． この拡張についての詳細は、現在検討中である．

References

[1] `http://www.klic.org/software/klic/` (1999)

[2] 古川，藤田，藤田，長谷川，淵，質問コンパイル法- KL1 による全解探索の新しいパラダイム - ，日本ソフトウェア科学会 第 9 回大会論文集, pp. 97-100 (1992)

[3] 古川， 溝口 編， 並列論理型言語とその応用，共立出版 (1987)

[4] 宮崎，世木，演繹データベースの問合わせ処理，情報処理 Vol. 31, No. 2, pp.216-224 (1990)

[5] 村上， 応答型逐次プロセスの部分計算を用いた並列実行， 日本ソフトウェア科学会 第 11 回大会論文集, pp. 145-148 (1994)

[6] Nilsson M. and Tanaka H., Fleng Prolog - The LAnguage which turns Supercomputer into Parallel Prolog Machine, Logic Programming ’86 (LNCS 264) Springer-Verlag, pp 170-179 (1989)

[7] Tarau P., WAM-optimization in Bin Prolog: towards a realistic contination passing prolog engine, Tech. Rep. 92-3 Dept. d’Infomatique, Universite de Moncton, 1992

[8] 高木，村上，コンティニュエーションのデータ化を用いた論理型プログラムの並列化，村上 編、平成 7 年度〜 9 年度 科学研究費補助金 (基盤 C) 研究成果報告書， 課題番号 07680359, pp.48-84 (1998)

[9] Ueda K., Making Exhaustive Search Program Deterministic, Proc. of ICLP’86 (1986)

[10] 渡辺， 村上， コンティニュエーションのデータ化を用いた論理型プログラムの並列化手法，日本ソフトウェア科学会第 16 回大会論文集 pp. 77-80 (1999)