論　文

共有メモリ型並列計算機上での正則な項書換え系の一実装法

松石　航也[†*]　服部　哲[†]　岡野　浩三[†]　東野　輝夫[†]
谷口　健一[†]

An Implementation of Orthogonal TRS on Multi-Processor Machine with Shared Memory

Koya MATSUISHI[†*], Satoshi HATTORI[†], Kozo OKANO[†], Teruo HIGASHINO[†], and Kenichi TANIGUCHI[†]

あらまし　本論文では，共有メモリをもつマルチプロセッサ上での正則な項書換え系の効率の良い一実装法を提案する．提案する方法では，(1)「eventually outermost 書換え系列」に基づく書換え方法をとっていること，(2) 有限状態機械をマッチングの際に用い，そこで得られた情報を次回以降のマッチングの際にマルチプロセッサ環境上で再利用していること，(3) 多くの研究が特定のアーキテクチャに依存しているのに対し，特定のアーキテクチャに依存せず並列性を本質的に利用していること，等の特徴をもつ．提案する手法を評価するために並列項書換え過程を模倣するシミュレータを作成した．このシミュレータを用い，処理時間，CPU 利用率の計測を行った．与えられた項が十分大きいときには提案する方法で効率良く書換えが実行されることが確かめられた．

キーワード　項書換え系，正則，共有メモリ型マルチプロセッサ，マッチングオートマトン

1. まえがき

並列計算機上で項書換え系 [4] における正規形を速く求めるための方法がいくつか研究されている．例えば，RRM [1] の各プロセッサに項を表す木のノード 1 個を分担させる方法や，項を表す木を分割して，各プロセッサに木の部分を分担させる方法 [7], [12] 等が提案されている．文献 [1], [7] の方法においては，書換えの効率化は RRM のような特定のアーキテクチャに依存している．

本論文では，並列計算機のアーキテクチャに依存せず，項書換え系の本質的な並列性を利用した効率の良い書換え方法を提案する．このため汎用的な並列計算機のモデルとして，有限個のプロセッサからなる共有メモリ型並列計算機モデルを採用する．

項書換え系の部分クラスである正則な項書換え系を対象とする．正則な項書換え系においては，最外リデックスが有限ステップ内に書き換えられることが保証されるならば，正規形が（もしあれば）求められることが知られている [8]．このような書換え系列は「eventually outermost 書換え系列」と呼ばれる．

書換えを効率良く行うための工夫の一つとして，リデックスを発見するためのマッチングアルゴリズムが研究されている．その一つとして有限オートマトンを用いる判定アルゴリズムが知られている [2], [5], [6], [11]．特に [11] では，与えられた項に対して，すべての最外リデックスを見つける具体的なアルゴリズムを提案している．そして，並列最外戦略に基づき発見したすべての最外リデックスを一斉に書き換える，並列計算機上での実現法を提案している．

一般に，並列最外戦略では，サイズの異なる複数の部分項をすべて書き換えてから次のマッチングを行うので効率上問題になることがある．そこで，本論文では，「eventually outermost 書換え系列」に基づき，複数の部分項の書換えとマッチングが同期しない並列計算機環境で効率良く書換えを行うようにした．

本アルゴリズムの利点は以下の 2 点である．

まず，「eventually outermost 書換え系列」に基づいていること．この方法ではある程度独立した部分項は，独立に，それぞれのサイズに応じた時間をかけて書換えを進めていくことが可能である．次に，マッチン

[†] 大阪大学基礎工学部情報工学科，豊中市
Department of Information and Computer Sciences, Osaka University, Toyonaka-shi, 560 Japan
[*] 現在，富士通株式会社

グを効率良く行うためにマッチングオートマトンを用いていることである．この方針に基づく際，マッチングと書換えが独立して行われることによって生じ得る（定義上は）起こり得ない項の変換操作(注1)を防ぐこと（項の一貫性の保持）が必要になる．そこで，マッチング処理を含め変形処理中の部分項同士が常に重ならない書換え方法をとることとした．本手法では(1)部分項の書換えによって最外リデックスが新たに生じた場合，それらをもれなく効率良く見つけるため，また，(2)項の重なりが生じる状況において最外項の処理以外の処理を効率良く終了させるために，過去のマッチングオートマトンの動作履歴等を保持し，それらの情報と合わせてプロセス間通信を行うという工夫をした．

本実現法の評価を行うために，共有メモリ型並列計算機で起こる書換えの過程を模倣するシステムを逐次型計算機上で作成した．いくつかの例題に対し，プロセッサの個数が1，2，4，8，16台の場合のシミュレーションを行い，書換え実行時間の計測を行った．扱う項が大きく規則の個数が多い場合には，並列度が高く高速に正規形を得ることができた．これは，コストのかかるプロセス間通信を少なくし，プロセッサ間で適切に負荷分散できたこと等が理由であろう．これらの結果から本実現法が有用であるといえる．

以下，**2.**で項書換え系の定義，諸性質と項書換え系の実装とは何かについて説明し，**3.**で項書換え系の実現法を提案する．**4.**で本方法の評価結果を示す．

2. 項書換え系と計算機モデル

この章では，項書換え系に関するいくつかの定義と諸性質および対象とする計算機モデルを述べる．

2.1 項書換え系

[定義 2.1] 項書換え系：項書換え系 Σ は，項集合 T と書換え規則の集合 R との2字組 $\langle T, R\rangle$ である．

項集合 $T(=T(F,V))$ は，以下の2条件を満たす最小の集合である．ここで，F は有限個の関数記号からなる集合，V は可算無限個の変数記号からなる集合である．各関数記号には一つの非負整数が付随し，これをアリティーと呼ぶ．項 $t \in T$ は，変数を含まない $(Var(t)=\emptyset)$ とき，基礎項と言う．

- $x \in V$ならば $x \in T$
- $f \in F, t_1, \cdots, t_n \in T, n$ は f のアリティーならば $f(t_1, \cdots t_n) \in T$

書換え規則 $(l \to r(l, r \in T))$ は，以下の2条件を満たす項の2字組である．ここで，$Var(t)$ は，項 t に含まれる変数の集合である．

(I)：$l \notin V$，(II)：$Var(l) \supseteq Var(r)$

書換え規則 $l \to r$ において，l を規則左辺，r を規則右辺と言う． □

[定義 2.2] 出現，部分項：項 $t \in T$ に対し，出現の集合 $Oc(t)$，出現 $u \in Oc(t)$ における部分項 $t/u \in T$，および，出現 $u \in Oc(t)$ における記号 $t(u \in F \cup V$ を，それぞれ以下のように定義する．

(I) $t \in V$ のとき，$Oc(t)=\{\varepsilon\}$，$t/\varepsilon = t$，$t(\varepsilon) = t$.

(II) $t = f(t_1, \cdots, t_n)$ のとき，$Oc(t) = \{\varepsilon\} \cup \{i \cdot v \mid 1 \leqq i \leqq n, v \in Oc(t_i)\}$，$t/\varepsilon = t$，$t(\varepsilon) = f$，$t/(i \cdot v) = t_i/v$，$t(i \cdot v) = t_i(v)$. □

項集合 $T(F,V)$ 上の代入 σ は V から T への写像であり，集合 $D(\sigma) = \{x \in V | \sigma(x) \neq x\}$ が有限なものを言う．

この σ は自然に T から T への写像 θ に拡張できる．

[定義 2.3] 書換え，リデックス，マッチング代入：項書換え系 $\Sigma = \langle T(F,V), R\rangle$ が与えられたとき，項 $s \in T$ が項 $t \in T$ に書き換えられるとは，以下の条件が成り立つことである．ここで，$s[u \leftarrow t']$ とは，s の部分項 s/u を t' で置き換えて得られる項を表す．

書換え規則 $l \to r \in R$ と s の出現 $u \in Oc(s)$ と代入 σ が存在して，$s/u = l\sigma$ かつ $t = s[u \leftarrow r\sigma]$ が成り立つ．

このとき，これを $s \to_R t$，若しくは単に $s \to t$ と表す．また，このとき部分項 s/u をリデックス，σ をマッチング代入と言い，s/u は，規則左辺 l にマッチすると言う． □

[定義 2.4] 書換え系列：項書換え系 $\Sigma = \langle T, R\rangle$ が与えられたとき，書換えによって得られる任意の（有限または無限の）列，$t_0 \to t_1 \to \cdots \to t_n \to \cdots (t_i \in T)$ を書換え系列と言う． □

[定義 2.5] 正規形：項書換え系 $\Sigma = \langle T, R\rangle$ において，項 $s \in T$ に対し $s \to t$ となる項 t が存在しないとき，s は正規形であると言う．

また，項 $s \in T$ に対し，s より始まる書換え系列で得られる正規形 $t \in T$ を s の正規形と言う． □

[定義 2.6] 最外リデックス：項書換え系 $\Sigma = \langle T, R\rangle$ において，項 $s \in T$ に対し，部分項 $s/u(u \in Oc(s))$ がリデックスであり，かつ，いかなるリデックスの部

(注1)：例えば書換え途中の部分項に対してマッチングを開始することなど．

分項でもないとき，s/u を s の最外リデックスと言う．一般に最外リデックスは複数存在する． □

以下では，本研究の対象とする正則な項書換え系について定義する．

[定義 2.7] 左線形：項書換え系は，いずれの規則の左辺についても同じ変数が2回以上現れることがないとき，左線形であると言う． □

[定義 2.8] 重なりがない：項書換え系 $\Sigma = \langle T, R\rangle$ において，任意の規則左辺 $s(s = l_1, l_1 \to r_1 \in R)$ と規則左辺の変数でない部分項 $t(t = l_2/u, l_2 \to r_2 \in R, t \notin V)$ に対して，s と t に共通の変数がないように変数の名前替えを行った後，いかなる代入 θ に対しても，$s\theta \neq t\theta$ が成り立つとき，重なりがないと言う． □

[定義 2.9] 正則な項書換え系：左線形でかつ重なりがない項書換え系を正則な項書換え系と言う． □

正則な項書換え系には次のような性質がある．この性質は計算の一意性を示しているものである．

[性質 2.1] 正則な項書換え系における正規形

正則な項書換え系においては，任意の基礎項に対して，その正規形は，たかだか一つである． □

以下では，書換えと正規形との関係において重要な書換え戦略とその性質について述べる．

[定義 2.10] 書換え戦略：書換えを行う際に，複数あるリデックスの中から実際に書換えを行うリデックスを選んで書き換える手順を書換え戦略と言う． □

[定義 2.11] 正規化戦略：入力項が正規形をもつならば必ず正規形を得られるような書換え戦略を正規化戦略と言う． □

[定義 2.12] 非同期最外書換え系列 (eventually outermost rewriting sequence)：次の条件を満たす書換え系列を非同期最外書換え系列と言う．

その書換え系列におけるすべての項 s について，s のすべての最外リデックスが，s からの有限ステップ内に書き換えられるか，若しくは最外リデックスでなくなるかのどちらかが起こる [8]． □

本論文では，非同期最外書換え系列を生成する戦略を「非同期最外戦略」と呼ぶことにする．

[性質 2.2] その書換え戦略に従ってできる書換え系列が，必ず非同期書換え系列であるような書換え戦略は，正則な項書換え系に対して正規化戦略である [8]． □

書換え戦略として有名な並列最外戦略や GK 戦略に従ってできる書換え系列は，正則な項書換え系に対して正規化戦略であることが知られている [9]．

書換え対象項に出現する変数は規則に現れない定数とみなしても計算には全く影響を与えない．従って，以下では基礎項のみを対象として話を進める．

2.2 並列計算機のモデル

局所メモリをもつ複数のプロセッサと，共有メモリからなる，非同期式並列計算機 [3] を対象とする．

各プロセッサは，独立したプログラムカウンタをもち，共有メモリ上のプログラムを実行する．局所メモリは，それをもつプロセッサのみが読み書きできる．共有メモリは，どのプロセッサも読み書きできる．共有メモリのアクセス形態は CREW モデルとする．各プロセッサは，そのプロセッサに固有のプロセッサ番号をもっており，プログラムはプロセッサ番号に依存した動作の指定も含んでよいとする．

3. 提案する実現法

3.1 実現方法

ここでは，まず，非同期最外戦略に基づいた我々の手法の概要について述べる．次にその具体的な実現方法について述べ，実現方法が非同期最外戦略に基づいていることの略証を与える．

我々の目的は以下の仮定のもとで，非同期最外戦略に基づいて並列計算機上で対象項となる基礎項を効率良く書き換えること，および，正規形があれば常にそれを求められることである．

[仮定 1] 書換え規則の数と規則左辺のサイズがともに大きいこと．

[仮定 2] いつも対象項にリデックスが広く多く存在すること．

常に正規形を得るためには，並列最外戦略や GK 戦略が戦略の候補として考えられる．しかしながら，並列最外戦略による TRS のマルチプロセッサ上での実現は効率的とは言えない．並列戦略では，書換えのたびに同期をとらなければならないからである．一般に，書換えの対象となる部分項のサイズは大きく異なるのが普通であり，同期をとることによる実行効率の低下が予測される．GK 戦略も同様な問題をもつ．そこで我々は非同期最外戦略を用いる．

まずプロセッサ数を考慮に入れていない簡単な実現方針1を説明する．

ここでは（1）新たに最外リデックス候補が発生すれば，再び根ノードからマッチングを調べるのではなく，最外リデックスを見つけるのに十分なノードで，かつ，なるべく近いところからマッチングを調べる，

（2）項の一貫性の保持のため新たな最外リデックス候補の項 t の処理を始める前に t のすべての部分項の処理をあらかじめ終了させる，という方針をとる．

書換え対象の基礎項（“対象項”と呼ぶ）は木表現で共有メモリに置く．書換えは共有メモリ上の木の操作（新ノードの生成，ノードの削除，ポインタの変更など）を何回か行うことにより実行される．書換え対象項を共有メモリ上に置かず，局所メモリ上に分散配置する方法も考えられるが，随時変化する項を適切に負荷分散して配置する問題を別途解決する必要がある．今回は共有メモリ上に配置する問題を対象とする．

[実現方針 1]

（1） 最初は対象項に対するマッチングのみを行う．

（2） 部分項 t に対するマッチングの結果，(a) t はリデックスではないか，(b) t がリデックスであるか，がわかる．

（3） (a) であれば t が既約かどうかを判定し，既約であれば何もしない．既約でなければ t の部分項である複数の最外リデックスの候補に対し，マッチングを独立して行う（それぞれの部分項に対して（2）に戻る）．(b) であれば項 t の書換えを引き続き行う．

（4） 一般に，部分項の書換えの影響を受けて，最外リデックスの候補になる項 s が出現する．そのような s に対する処理は，まず s のすべての部分項に対する処理に対し強制終了通知を行う．一方強制終了通知を受けた処理は適当な処理を終えて最子孫プロセスから終了していく．項 s のすべての部分項に対する処理がこのように終了してから項 s のマッチングを開始する（s について（2）に戻る）． □

（4）においては部分項の処理をすべて終了させてからマッチングを行っているが，一方で，項の先祖子孫関係でない位置にある任意の部分項のマッチングや書換えの処理はお互いに影響を与えないので同期をとる必要はなく，仮定 2 のもとで非同期最外戦略の利点が生かされる．

まえがきに述べたように，上記のマッチング処理を効率良く行うために仮定 1 のもとでは，マッチングオートマトンを利用するのが効率的だと考えられる．また，実際にはプロセッサ数が固定されるためプロセスという活動単位を設け，FIFO のプロセスキューによってプロセッサへの割当てを管理する．プロセス p は割り当てられた部分項（以降 $ST(p)$ と表す）に対するマッチングと書換えを実行する．プロセスは“活動”と“休止”という二つの状態をもつ．休止状態のプロセスはプロセスキューに入っている状態とそうでない状態にある．任意の時点および任意のノードにおいてそのノードに位置するプロセスはたかだか一つである．プロセッサは活動にあるプロセスにのみ割り当てられる．

プロセスの情報を保持したりプロセス間通信のために後述のプロセス情報を各プロセスに設ける．これらは共有メモリ上に置かれる．プロセス間通信のために主にプロセス情報内の再起動フラグ r と終了要請フラグ c を用いる．フラグ r は「子孫プロセスに対して処理の終了要請をせよ」ということを表し，フラグ c は「以降の処理の終了要請」を表す．この観点で実現方針 1 を詳細化したものが以下の［実現方針 2］である．

[実現方針 2–プロセス管理部]

（1） 計算システムが対象項に対する休止プロセスを一つ生成する．このプロセスのフラグ r，c はリセットしておきプロセスキューに入れる．

（2） プロセスキュー中のプロセスを可能な限り，アイドルなプロセッサに割り当てる．プロセッサを割り当てられたプロセスは活動状態となりプロセスキューから出る．これらのプロセスの任意の一つを p とする．各 p について［実現方針 2–プロセス内の処理］を実行する．（2）を繰り返す．

（3） プロセスがすべて終了すれば対象項の正規形が求まっている． □

[実現方針 2–プロセス内の処理]

（1） プロセス p は以下の条件のうち最初に合致するものに従って動作する．

（a） フラグ c が設定されている．：p は直ちに終了する．

（b） p の子孫プロセスは存在しない（すべて終了している）．：フラグ r，c をリセットして（2）に進む．

（c） フラグ r が設定されている．：p のすべての子孫プロセスに対しフラグ c を設定する．p は自身のフラグ r をリセットし休止状態に入る．

（d） その他：p は休止状態に入る．

（2） プロセス p は $ST(p)$ に対してマッチング処理をマッチングオートマトンを用いることにより実行する．この結果

（a） $ST(p)$ にマッチする規則がないことがわかる．かつ $ST(p)$ の部分項で最大リデックスである可能性のある（重ならない）候補部分項の空でない集合 C がわかる．

（b） $ST(p)$ が既約であることがわかる（C が空集合であることがわかる）(注2)．

（c） $ST(p)$ にマッチする規則がわかる．

の三つのいずれかの状況になる(注3)．

（3） （a）の場合 p にフラグ c が設定されていなければ C の各要素に対する子プロセスを複数生成し，これらのプロセスが休止プロセスとしてプロセスキューに入る．p は休止プロセスとなる．

フラグ c が設定されていれば子プロセスを生成することなく直ちに終了する．

（b）の場合 $ST(p)$ 中の各ノードの既約フラグ n（後述）を設定し p は終了する．

（c）の場合フラグ c が設定されていれば直ちに終了する．そうでなければ p は $ST(p)$ を書き換える(注4)．（リデックスになる可能性が生じたという意味で）影響を受ける可能性のあるプロセスがあればそれらのうち最祖先のプロセス（$F(p)$ と表す．求め方は後述）のフラグ r を設定する．$F(p)$ は休止プロセスとしてプロセスキューに入れられる．その後 p は終了する．

（4） プロセスが終了する上記のすべての状況において「自プロセスが終了することにより親プロセスの子プロセスがすべてなくなる」場合(注5)は，プロセッサは自プロセスを終了した後，該当親プロセスの処理を活動状態にして直ちに開始する．ここで開始したプロセスは［実現方針2–プロセス内の処理］に従う． □

次節において証明を行うが，(3)（c）においてフラグ r の働きと $F(p)$ の選び方により，最外リデックスである可能性が生じた部分項を再び処理することができる．アルゴリズムによって，フラグ c が設定されたプロセスは最子孫プロセスから順に終了するので有限時間内にフラグ r が設定されたプロセスが活動状態に入れる．

（2）（b）において部分項 s が既約であることがわかれば s のすべてのノードにフラグ n が設定される（s のいくつかの部分木のノードは既に n が設定されていることが多い）．ある部分項が既約かどうかはマッチングオートマトンを起動するより前に n を調べることにより効率良く判定できることが多い．

3.2 実現方法の正しさ

以下では，“重なりのない項変形機械”と重なりのない項変形機械が生成する項系列の定義を与え，これらの定義のもとで，提案アルゴリズムが重なりのない項変形機械であり，提案アルゴリズムの生成する項系列が非同期最外書換え系列であることの略証を与える．

［定義 3.1］ 重なりのない項変形機械：重なりのない項変形機械は（項を表す）木から別の木に繰り返し変形する．各変形は以下のように行う．部分木とそれに対する変形規則を指定すると，部分木をその変形規則で変形する．変形は有限時間で行い，変形開始時から変形終了までの間，変形中の木，あるいは，変形中の木に含まれる部分木，変形中の木を含む部分木に関して変形を行わない限りにおいて，別の部分木に関する変形を同時に行ってよい． □

［定義 3.2］ 重なりのない項変形機械が生成する項系列と書換え系列：重なりのない項変形機械が部分木の変形を始めたらその変形は瞬時に行われるとみなして得られる木の系列を（その木を項とみなして）重なりのない項変形機械が生成する項系列と呼ぶ．その各変形規則が項書換え系の書換え規則であり，与えられる対象項が項書換え系の項であるとき生成される項系列を重なりのない項変形機械が生成する書換え系列と呼ぶ． □

重なりのない項変形機械では，項書換え系における各部分項の一変形操作は変形時間を0と考えても，ある時間がかかると考えても同じ結果になる．もちろん各部分項の書換え時間にばらつきが生じるので，項書換えの時間が0の場合とそうでない場合では生成される項系列は異なるおそれがある．ここでは非同期最外書換え系列であることのみ示せばよいのでこの違いは本質的ではない．

以下の命題で非同期最外書換え系列であることの略証を与える．主命題は命題4である．

これらの証明は次の事実に基づいている．

［事実］

（1） 任意の部分項に位置するプロセスはたかだか一つである．

（2） プロセス p がプロセス q の親であればそれぞれのプロセスが位置する部分項 $ST(p)$，$ST(q)$ について $ST(p)$ は $ST(q)$ の上位項である（以降項 t が項 s の部分項であるとき s は t の上位項と呼ぶこととする）．

（3） 任意のマッチング処理中あるいは書換え処理中のプロセス p について，プロセス p のすべての祖

(注2)：既約フラグ n も判定に用いている．
(注3)：マッチング処理に時間がかかるので一般にこの時点でフラグ c が再び設定されることがある．
(注4)：実際はフラグ c の設定にかかわらず書き換えてもよい．
(注5)：このときは子孫プロセスすべてがなくなる状態でもある．

先プロセスは休止中である．

（4） 最子孫のプロセスは活動中か，休止中でプロセスキューに入っているかのいずれかである．

（5） 活動中のプロセスしか終了できない，また，活動中のプロセスしか他プロセスを生成できない．

（6） 部分項 t に位置するマッチングオートマトンは t 中に（t ではない）リデックスが存在すれば，必ず空でないリデックス候補集合 C を返す． □

[補題] [事実] とアルゴリズムから以下のことが言える．マッチング処理中あるいは書換え処理中のプロセスは子孫プロセスをもたない(注6)． □

[命題1] 提案アルゴリズムが重なりのない項変形機械であること．

(略証) [補題] より，マッチング処理や書換え処理中はより部分項に位置する子孫プロセスが存在しない．よって特に任意の部分項の変形中に，その更なる部分項の変形処理が行われることはない．また，マッチングが成功し項変形を行う場合その変形は有限時間で終了する．この際，[事実]（3）より，項変形中により上位の部分項に対するマッチング処理や項変形処理が行われることもない． □

[命題2] マッチングオートマトンの働きと一規則による項変形操作が正しいという仮定のもとで提案アルゴリズムが生成する項系列は書換え系列であること．

(略証) [命題1] により，ある部分項のマッチング処理や書換え処理中に他の部分項の処理によって影響を受けることはない．マッチングオートマトンの働きにより，マッチング成功時に適用される規則はそのリデックスに対する規則であり，また変形処理は書換え規則に沿ったものである． □

ここで再起動フラグ r に関する重要な補題 [命題3] を証明する．

[命題3] 提案アルゴリズムにおいて，再起動フラグ r を設定されたプロセスあるいはそのある先祖プロセスがいつかはマッチング処理を開始する（開始したプロセスを p とする）．このときプロセス p のすべての子孫プロセスは終了している．

(略証) フラグ r を設定されたプロセス p はすべての子孫プロセスにフラグ c を設定し休止状態に入る．

[事実]（1），（2）よりフラグ c を設定すべきプロセス数は有限である．アルゴリズムでは，あるプロセスが子プロセスを生成するのはある部分項 u のマッチングに失敗したときだけであり，このとき部分項 u は変わらない．すなわち子プロセスが無限に増えることはない．よってすべての子孫プロセスにフラグ c を設定する処理は有限時間で終わる．

フラグ c を設定されたプロセスのうち最子孫プロセスは [事実]（4）より活動状態にあるかプロセスキューにある．活動プロセスはマッチング処理中であればマッチングが終了した時点で終了するし，一規則による書換え処理中であればその書換え処理が終了した時点で終了する．またプロセスキューにあるプロセスは，いつかは活動状態となり，活動状態になった直後終了する．アルゴリズムでは休止プロセスのうち直接の子プロセスがすべて終了したものは直ちに活動状態に入る．このようなプロセスにフラグ c が設定されていればアルゴリズムに従い直ちに終了する．このようにフラグ c を設定された子孫プロセスは最子孫プロセスから親に向かって終了していく．最終的に p が活動状態になる．ここで p にフラグ c が設定されていなければ，プロセス p はマッチング処理を開始する．一方 p にフラグ c が設定されていれば，プロセス p は終了する．このフラグはプロセス p のある祖先プロセス q が設定したものである．この q について p と同様に考えればよい．この繰返しは有限である． □

[命題4] 提案アルゴリズムによって得られる書換え系列中の任意の項における最外リデックスは，(a) いつかは書き換えられるか，(b) 最外リデックスでなくなるかのいずれかである．

提案アルゴリズムによって得られる書換え系列中の項 t について，以下の場合分けをする．(A) t が書換え系列の最初の項である．(B) t が書換え系列の最初の項でない．

t における任意の最外リデックス s に着目する．(A) の場合は，s が t そのものの場合は，最初のプロセスによって書き換えられる．それ以外の場合は，[事実]（6）と [命題1] 等により s に位置する子プロセスがやがて生成される．そして，s の書換えが行われる ((a) が満たされる)．

(B) の場合はそのような s は（1）s を含むより上位の項の書換えによって最外リデックスになったか，（2）s のある部分項 u の書換えによって最外リデックスになったかのいずれかである．いずれの場合でも s あるいは s のある上位項にプロセス p が位置し p はフラグ r を設定されたことがある．[命題3] よりやがてプ

(注6)：厳密には子プロセスを生成する瞬間は子プロセスをもつが，生成直後に親プロセスは休止状態になる．

ロセス p あるいは p のある祖先プロセスがマッチング処理を始める．今マッチング処理を始めたプロセスを p とみなす．プロセス p の位置する部分項が実際に最外リデックスであればその部分項が書き換えられる((b) が満たされる)．そうでない場合は，[事実] (6) と [命題 1] 等により子プロセスが生成され，(A) s のある上位項がリデックスとして書き換えられるか (B) s に位置するプロセスがやがて生成されるかいずれかである．(A) の場合は (b) を満たし，(B) の場合は s の書換えが行われる ((a) が満たされる)． □

3.3 マッチングオートマトン

ここではマッチングオートマトンとその失敗関数 [11] について述べる．

マッチングオートマトンは有限状態オートマトンであり入力は検査対象の部分項の関数記号列である．関数記号は幅優先探索順序で根ノードの関数記号から入力される．最終状態はマッチングする規則があるか否かを与える．またマッチする場合は最終状態からマッチする規則が具体的に何であるかがわかる．

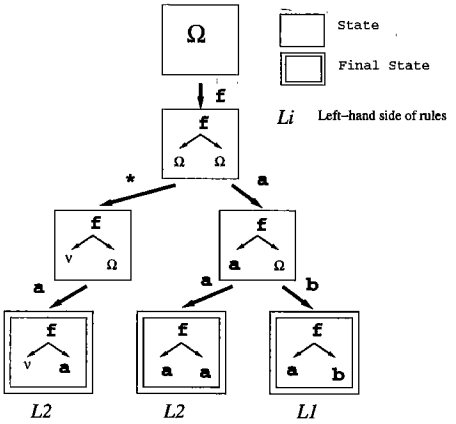
[例 3.1] [マッチングオートマトン]

図 1 のマッチングオートマトンは $L1 = f(a, b)$ あるいは $L2 = f(X, a)$ ($L1$, $L2$ は規則の左辺) に対する任意の項のマッチングを見つける．

入力項 $f(f(c, c), a)$ に対して，このオートマトンへの入力系列と遷移は以下のようになる．f, f, a という入力に対し，オートマトンは状態を Ω から始めて，f, $*$, a という遷移に添って，$f(\Omega, \Omega)$, $f(\nu, \Omega)$, $f(\nu, a)$ と変えていく．状態 $f(\nu, a)$ は終了状態の一つであり，$L2$ がマッチすることを表す． □

失敗関数を使うと以下のように子プロセスを効率良く決定できる．

例えば，項 $(f(g(\ldots), c))$ がリデックスであるかどうかを前述のマッチングオートマトン (図 1) を用いて判定することを考える．マッチングオートマトンは入力系列 f, g, c を入力していく．この時点でオートマトンはマッチする入力がないことを判定する．この時点で，失敗時の情報を利用すると，左部分項の根ノードは g であり右部分項はリデックスになり得ないことがわかる．よって，生成すべき子プロセスは一つだけであり，また，そのプロセスはオートマトンを初期状態ではなく状態 $g(\Omega)$ とすべきことがわかる．このようにマッチング失敗時における子プロセスとオートマン状態情報を失敗関数は与える．



If there is no corresponding symbol in transitions but there is a transition labeled $*$, then the transition labeled $*$ is selected. For other cases, the automaton fails.

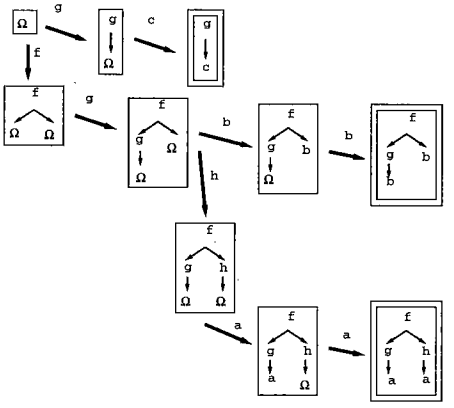


図 1 マッチングオートマトンの例
Fig. 1 Example of matching automaton.

3.4 プロセス情報

ここでは前述の $F(p)$ を求める問題，すなわち，「どこまで上位に位置するプロセッサに関してマッチングを再開すべきか」という問題を解決する．

プロセス情報を用いると，現時点のすべての休止または活動プロセスの生起関係や，各プロセスのマッチングに関する情報を得ることができる．プロセスの状態の変化に伴い，随時，プロセス情報は生成消滅したり，更新される．

一つのプロセス p に対するプロセス情報は次の項目をもつ．$ST(p)$ の根ノードへのポインタ，p が再開するときのマッチングオートマトンの状態 $S(p)$，p がマッチング時に参照していたノードへのポインタ群

$$x \times 0 \to 0$$
$$x \times s(y) \to x + x \times y$$
$$x + 0 \to x$$
$$x + s(y) \to s(x + y)$$

図2　比較用項書換え系
Fig. 2　Example TRS.

$IN(p)$，親プロセスからの深さ $D(p)$，子プロセスの数，親プロセスへのポインタ，フラグ c，フラグ r．ここで，プロセス p のプロセス q からの“深さ”はそれぞれのプロセスの位置するノード間の距離とする．

プロセス p が $ST(p)$ を書き換え終わったとする．このとき，p は $F(p)$ を以下のようにプロセス情報を用いて決定する．

$F(p)$ は書換えのあったノードからたかだか規則左辺項の“深さ”M まで対象項を遡ったノードに位置している．(p の q からの深さを表す) $\sum_{i=p}^{q} D(i)$，M から q が $F(p)$ か否かを決定できる．例えば $\sum_{i=p}^{q} D(i)$ が M よりも大きければ，$ST(p)$ が変化しても q はマッチングを見つけることは決してない．

また，$S(p)$，$IN(p)$ 等を用いることにより再活動時のマッチング作業が効率良く行われる．

4. 評　　価

提案する方法に従って起こる書換えの過程を模倣するシステムを作成し，本実現法の評価を行った．

4.1 評価用シミュレータ

C++用の並列プログラムとシミュレーション用ライブラリー AWESIME(注7)を用いて，我々の実現法の評価用シミュレータを作成した．

ノードの作成，参照，除去，プロセス情報の作成，除去，参照，マッチングオートマトン情報の作成，参照など共有メモリに対する操作とプロセスキューによるプロセッサ割当てに「時間」を割り当てて処理時間を計測した．正確には単位メモリへのアクセスを1ステップとし，作成するメモリ領域の大きさによって最大3ステップまでの重みを設けている．また，実際のメモリアクセスのスピード差を考慮に入れ，プロセスキューによるプロセッサ割当てに要する待ち時間は単位ステップの100倍とし，通信フラグの書換えに要する時間は単位ステップの10倍とした．

4.2 評価方法

まず，純粋な並列最外戦略と比較するために，乗算を行う標準的な書換え規則（図2）で $30 \times 10 + 2 \times 100$ の計算を純粋な並列最外戦略と本方法とで実行した．

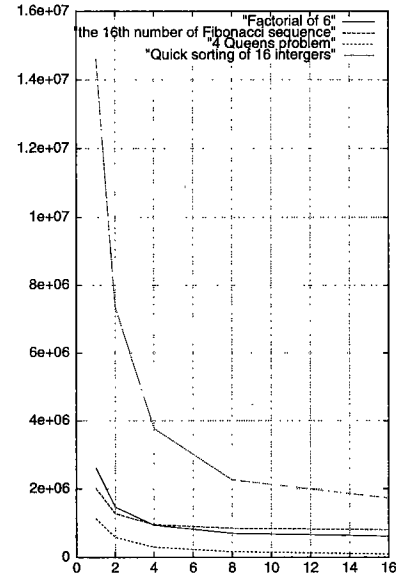


図3　例題 TRS の実行時間/縦軸ステップ数横軸台数
Fig. 3　Simulation time of TRSs Y axis: Steps, X axis: The number of processors.

図2の第2規則は変数 x のコピーを要する．与項（実際には s，0 によって数は表現されている）の＋の左項は大きな項 (30) のコピーが10回必要なのに対し，右項は小さな項のコピーをかなりの回数繰り返す．これは同期式並列書換えでは不利になる実例である．

次に，各例題（階乗，フィボナッチ数列，n クイーン問題，クイックソーティング）について，プロセッサ数が1，2，4，8，16台の計算機で実行したときの時間をシミュレータを用いて計測した．またプロセッサ使用率を時間経過に従って計測した．

4.3 評価結果

乗算の例では同期式並列最外戦略より，本方法で30%ほど実行ステップ数で有利になることがわかった．このような構造をもつ例はほかにも，「if $p(x,y)$ then $q(x)$ else $r(y)$ のような項に対して，$p(x,y)$ の書換えが単純，$q(x)$，$r(y)$ の書換えはコピー規則等により複雑」というようによく現れる．よって，このような項のサイズの差が顕著な場合は同期式並列最外戦略に比べ（本実現法がプロセス間通信のオーバヘッドをもつのにもかかわらず）本戦略が有効であることが確かめられた．

(注7)：“A Widely Extensible SIMulation Environment.”[10]

例題のシミュレーション結果では，フィボナッチ数や階乗を求める問題で台数効果があまりない一方で，nクイーン問題とクイックソーティングでは台数効果が顕著に表れた（図3）．これらの違いは，特にnクイーン問題やクイックソーティングでは規則数が多く，また，各時点において並列に処理できるリデックスが多いためと思われる．フィボナッチ数を求める問題で台数効果がない理由は次のとおりである．漸化式により＋と数値からなる2分木を作るところまでは効率良く本アルゴリズムが機能するが，できた2分木はかなりアンバランスになっており，項のサイズのわりには＋の書換え規則が適応できるリデックスが少ないことによる．これらの結果から，ある程度規則等の多い例題においてはオーバヘッド（例題によって異なる）に見合うのに十分な効率が見込まれることがわかった．

プロセッサ使用率については，フィボナッチ数を求める問題以外では台数を増やしてもアイドル台数はあまり増えていなかった．負荷分散が適切に行われていると判断できる．

以上より，項の書換えの処理に十分なばらつきがあるならば，本手法が純粋な並列最外戦略に比べ有効であることが確認できた．

5. むすび

本論文では，正則な項書換え系を対象とし，有限個のプロセッサからなる共有メモリ型並列計算機上で，「非同期的に」最外リデックスを書き換えて正規形を求めるための方法を提案した．本実現法では，扱う項が大きく規則の個数が大きい書換え系で最外リデックスが常に多いと期待できる場合には，高速に正規形を得ることがわかった．

謝辞 有益なコメントを下さいました査読者に感謝致します．

文献

[1] H. Aida, J. Goguen, and J. Messeguer, "Compiling concurrent rewriting onto the rewrite rule machine," Lecture Notes in Computer Science 516, pp.320–332, 1990.

[2] J. Burghardt, "A tree pattern matching algorithm with reasonable space requirements," Lecture Notes in Computer Science 299, pp.1–15, 1988.

[3] L.G. Valiant, Handbook of Theoretical Computer Science Vol.A (Algorithm and Complexity), General Purpose Parallel Architecture, ed. J. van Leeuwen, Chapter 18, Elsevier Science Publishers, 1990.

[4] N. Dershowitz and J.-P. Jouannaud, Handbook of Theoretical Computer Science Vol.B (Formal Models and Semantics), Rewrite Systems, ed., J. van Leeuwen, Chapter 6, Elsevier Science Publishers, 1990.

[5] G. Huet and J.-J. Levy, "Call by need computations in non-ambiguous linear term rewriting systems," report in Institute de Recherche d'Informatique et d'Automatique, 1979.

[6] C.M. Hoffmann and M.J. O'donnell, "Pattern matching in trees," Journal of ACM, vol.29, no.1, 1982.

[7] J. Goguen, C. Kirchner, and J. Meseguer, "Concurrent term rewriting as a model of computation," Lecture Notes in Computer Science 279, pp.53–93 1987.

[8] M.J. O'donnell, "Computing in systems described by equatioins," Lecture Notes in Computer Science 58, 1977.

[9] J.W. Klop, "Term rewriting systems" in Handbook of Logic in Computer Science, Vol.II, eds., S. Abramsky, D. Gabbay, and T. Maibaum, Oxford University Press, pp.1–116, 1992.

[10] D. Grunwald, "A users guide to AWESIME: An object oriented parallel programming and simulation system," Univ. of Colorado Technical Report CU-CS-552-91, 1991.

[11] 山本晋一郎，坂部俊樹，稲垣康善，"項書換え系における並列最外戦略の効率的な実現法，" 信学論（D-I），vol.J77-D-I, no.10, pp.693–702, Oct. 1994.

[12] 山本晋一郎，石川 亮，酒井正彦，阿草清滋，"共有メモリ型並列計算機における項書換え系の実現，" 信学論（D-I），vol.J78-D-I, no.6, pp.559–562, June 1995.

[13] 岡野浩三，松石航也，東野輝夫，谷口健一，"正則な項書換え系のマルチプロセッサ上での実現，" 第58回情処全大，1-173-174, 1996.

（平成8年12月11日受付，9年8月14日再受付）

松石 航也

平6年3月阪大・基礎工・情報中退，同年4月同大大学院博士前期課程入学．平8同大学院博士前期課程了．在学中は，項書換え系の研究に従事．

服部 哲

平4阪大・基礎工・情報卒．現在同大大学院博士後期課程在学中．項書換え系，特に，条件付き項書換え系の研究等に従事．

岡野　浩三　（正員）

平2阪大・基礎工・情報卒．平5同大大学院博士後期課程中退．同年同大情報工学科助手．工博．代数的手法によるプログラム開発，分散システムなどの研究に従事．情報処理学会会員．

東野　輝夫　（正員）

昭54阪大・基礎工・情報卒．昭59同大大学院博士課程了．同年同大助手．平2，6モントリオール大学客員研究員．現在，阪大・大学院基礎工学研究科・情報数理系専攻助教授．工博．分散システム，通信プロトコル等の研究に従事．情報処理学会，IEEE，ACM各会員．

谷口　健一　（正員）

昭40阪大・工・電子卒．昭45同大大学院博士課程了．同年同大・基礎工・助手．現在，同大大学院基礎工学研究科・情報数理系専攻教授．工博．この間，計算理論，ソフトウェアやハードウェアの仕様記述・実現・検証の代数的手法および支援システム，関数型言語の処理系，分散システムや通信プロトコルの設計・検証法などに関する研究に従事．