意味的な排他制御を可能とする排他制御プログラムの自動生成

田村 真司† 宝珍 輝尚‡ 野宮 浩揮‡

†‡京都工芸繊維大学大学院 〒606-8585 京都市左京区松ヶ崎御所海道町

E-mail: †s.tamura01@gmail.com, ‡{hochin, nomiya}@kit.ac.jp

あらまし 本論文では，意味的な排他制御を可能とする排他制御プログラムを遺伝的プログラミング（GP）を用いて自動生成するシステムの設計と実装について報告する．ここで，意味的な排他制御とは，新しい情報の書き込みしか行われないテーブルでは書き込み操作が競合することはないといった意味情報を用いて，より効率的な排他制御を行うことである．二相ロック法では制限が強すぎる場面での有効性を示すために，意味的に競合を緩和する関係を記述してもらい，これに基づいて，排他制御プログラムの自動生成を行う手法について，実現のためのアプローチを示した後，実装について述べる．

キーワード 排他制御プログラム，遺伝的プログラミング，意味的排他制御，自動生成

Automatic Generation of Concurrency Control Program Allowing Semantic Concurrency Control

Shinji TAMURA† Teruhisa HOCHIN‡ and Hiroki NOMIYA‡

†‡School of Science and Technology, Kyoto Institute of Technology

Goshokaido-cho, Matsugasaki, Sakyo-ku, Kyoto-shi, Kyoto 606-8585 Japan

E-mail: †s.tamura01@gmail.com, ‡{hochin, nomiya}@kit.ac.jp

1. はじめに

日進月歩で技術が進化している今日，様々なコンピュータが環境に合わせて発達し，それと同時にデータベースが多種多様な場面で必要となっている．しかし，様々な分野の要求に応えるには，単一の汎用的なデータベース管理システム（Database management system, DBMS）では対応できない部分が出てくる．このことから，各分野に対応した機能を持つ DBMS が望まれる[1]が，その構築は難しい．また，実現できたとしても特定の場面でしか使わない不要な機能が多くなる恐れがある．つまり，各分野に特化した DBMS が必要となるのであるが，その構築には多大なコストと時間が必要となる．このことから，DBMS の拡張性が強く求められている[2]．

ここで，データベースを利用する際には，分野によって異なるトランザクションが発生すると考えられる．それは例えば，データを読む操作がほぼ全てを占めていたり，ある一つのデータに対しての操作が集中したり，といったように様々な特徴があり，ある特徴に対しては適切な排他制御機構も別の特徴を持つ分野で運用すると著しく効率が落ちてしまうことがある．発生するトランザクションの特徴に応じて最適な排他制御機構を生成できるならば，利用分野に最適な DBMS の構築に繋がる．

本研究では，DBMS の性能や信頼性に関わる最も重要な機構の一つである排他制御機構の生成に関して検討を行ってきた[3][4][5]．ここでは，遺伝的プログラミングを用いてトランザクションの特徴に合った排他制御プログラムの自動生成を試みている．

ここで，一般的な排他制御法としては，正当性が保証された二相ロック法が用いられることが多い．しかし，二相ロック法は制限が強いため，排他制御法としては適さない分野が多く存在する．これに対処する方法として，SQL のトランザクション分離レベル[6]の利用や，競合が発生したときに指定した競合を無視するように設定できる排他制御言語が用いられている[7]．しかし，前者の問題としては，READ と WRITE の競合は許さないが，WRITE と WRITE の競合を許すような競合の緩和を行うことができない点が挙げられ，後者では，競合直列等価が発生したときに判定を行うという性質から，実行コストがかかると考えられる点が挙げられる．後者ではさらに，排他制御言語のような例では実際に排他制御のルールを記述する必要があり，これにもまた多大なコストがかかると考えられる．

そこで本研究では，あまりコストをかけずにそれぞれの応用分野に適した様々な排他制御を利用可能とすることを目的として，緩和すべき競合だけをユーザに

記述してもらい，実際の排他制御の根幹となるルールを環境に適するように排他制御プログラムを自動生成する手法を提案する．本論文では，アプリケーションのデータベースへの命令の意味特性を利用することにより，トランザクションの競合可能性を減少させることができると言われている意味的排他制御を利用可能にすることを目指す．

以降，2．ではこれまでの研究について概説し，3．で遺伝的プログラミングについて概説し，4．でこれまでのシステムについて述べる．そして，5．で意味的排他制御について述べ，6．で問題解決アプローチについて述べる．7．で実験を行い，最後に 8．でまとめる．

2. 遺伝的プログラミングを用いた排他制御プログラムの自動生成

水野らは，遺伝的プログラミングを用いてトランザクションの特徴に合った排他制御プログラムを生成する試みを行っている[3]．ここでは，二相ロック法を実現するプログラムと時刻印順序法を実現するプログラムをもとにして基本機能を抽出し，遺伝的プログラミングでプログラムを表現する木（プログラム木）のノードとしている．初期プログラム木として二相ロック法用のプログラム木を与えたところ，より効率的な処理を行うプログラム木を得るという結果が得られており，トランザクションの特徴を反映した排他制御プログラムの生成という目的は達成されていると考えられる．しかしながら，プログラム木のノードは二相ロック法と時刻印順序法における基本機能であるため，これらをもとに遺伝的操作を施しても，これらの方法の混合手法が得られる程度で，全く新しい制御方法が得られる可能性は低い．

そこで，トランザクションの特徴に応じた排他制御プログラムの自動生成を目的として，水野の排他制御プログラム自動生成システムにおけるプログラム木のノードを汎用性の高いものとし，様々な制御プログラムの生成をも可能とする手法を提案してきた[4]．提案してきた手法は，排他制御プログラムが各データならびに各トランザクションに対して変数を設定し操作していることを利用するもので，プログラム木のノードとして変数の設定操作を行うものを導入する方法である．その結果，今までにない形の排他制御プログラムを生成することはできなかったが，競合する条件で実験時には初期木を指定しなくても，ロックを使うプログラムを生成することができた．

3. 遺伝的プログラミング

3.1. 遺伝的プログラミング

遺伝的アルゴリズム（Genetic Algorithm， GA）とは進化論的な考え方に基づいてデータを操作し，最適化の問題や学習，推論を扱う手法である．GA のアルゴリズムを以下に示す．

(1) 現世代の集団を生成する．
(2) 現世代の集団内の各個体に対して適合度を計算する．
(3) 適合度をもとに現世代の集団から個体を選択する．
(4) (3)で選択した個体に対して，突然変異・交叉などの操作を行い，次世代を生成する．
(5) (2)～(4)を必要数繰り返し，全ての世代の中で最高の適合度を持つ個体を解とする．

遺伝的プログラミング（Genetic Programming，GP）は，GA を構造的な表現ができるように拡張したものである．GP では，個体を木構造で表し，突然変異・交差などの遺伝的操作を部分木の変更によって実現させる．また，木構造はＳ式で表すこともできる．このときの対応関係を図 1 に示す．図 1 の F1，F2 は非終端記号，T1，T2 は終端記号を表している．

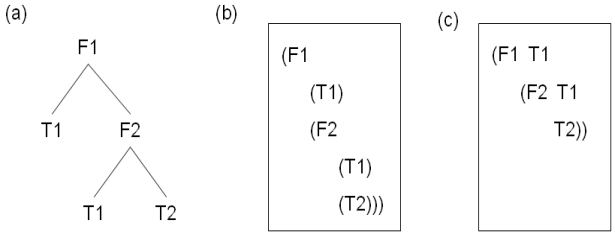

図 1．プログラムの(a)木構造表現(b)S 式表現と(c)Ｓ式の省略表現

3.2. 自動関数定義

人間のプログラマが作成したプログラムと GP で生成したプログラムの違いとして，関数(サブルーチン)の有無が挙げられる．関数を使用しない場合は，同じ処理をする部分がプログラム中に何度も登場してしまい，そのプログラムを格納するために，大量のリソースが必要となる．これに対して，関数を使用することにより，冗長性が低下し，結果として探索が効率化できる可能性がある．

また，自動関数化手法は GP の探索原理の根拠を示している，という見方もある．もともと，GA の探索原理の根拠としてスキーマ定理と積み木仮説があり，この仮説では各集団のもつスキーマと呼ばれる最適解の部分解を交叉・突然変異によって組み合わせて最適解を構成していくと説明されている．このスキーマが GP においては部分木であると考えられている．つまり，自動関数定義(Automatic Defining Function：ADF)とは，

有効な部分木を組み合わせることで，プログラムの短縮化による探索効率の向上や理論的な裏付けが実現できる手法である[8].

ADF を用いた GP を ADFGP と呼び，図 2 のように通常の生成プログラムを右側に，もう一方の左側に DEFUN を用いて定義した ADF を存在させる．このとき，VALUES 以下の ADF0_body に ADF0 のプログラム本体が存在する．また，左側で定義した ADF0 は非終端記号名 ADF0 として右側の生成プログラムで自由に使用することができる．

また，GP の実行に際して，左側の ADF0 の定義部分でも右側の生成プログラムと同様に交叉や突然変異を行う．ただし，ADF0 の定義部分は ADF0 の定義部分同士で，右側の生成プログラムの定義部分は生成プログラムの定義部分同士で交叉を行う．

システムの仕様として ADF の数，引数の数，記号の種別などを決める必要がある．

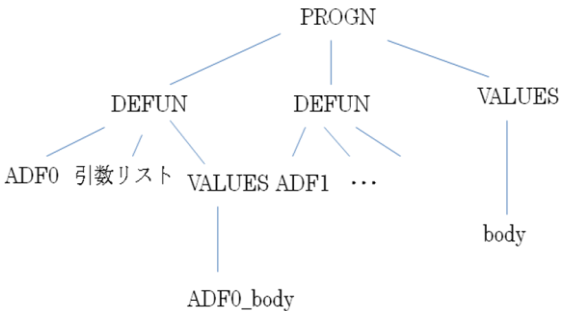

図 2. ADFGP のデータ構造

4. 排他制御プログラム生成システム

4.1. 排他制御プログラム生成システムの概要

本システムは大きく分けて二つのシステムから構成されている[4][5]. 大まかな流れとしては，まず「スケジュール生成システム」によりスケジュールを生成し，そのスケジュールを用いて「排他制御プログラム生成システム」で排他制御プログラムを生成する．

「スケジュール生成システム」は初期設定ファイルでトランザクションの数や操作するデータの数，含まれる書き込み(WRITE)命令の割合等を指定し，初期設定に基づいたスケジュールを生成する．

次に，「排他制御プログラム生成システム」の概要を図 3 に示す. このシステムは以下の順で実行される.

(1) 遺伝的操作により排他制御プログラムを得る.
(2) 排他制御プログラムで得たスケジュールを制御して適合度を得る.
(3) 最終世代に到っていなければ(2)で得た適合度をもとに(1)の操作に戻る．最終世代なら終了して最良個体を得る.

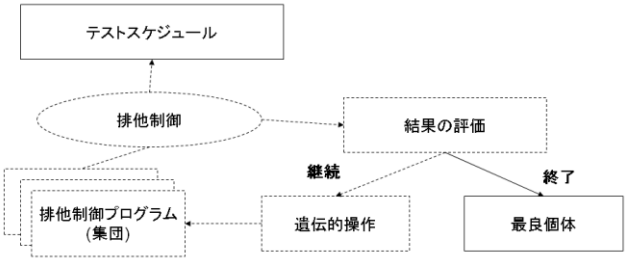

図 3. 排他制御プログラム生成システムの概要

4.2. 排他制御プログラム生成システムの設定

4.2.1. GP の設定

GP によるプログラム生成のための設定も必要である．以下の設定項目がある.

- s1：最大世代数
- s2：一世代における個体数
- s3：初期世代になるプログラムの指定の有無
- s4：プログラムの最大深さ
- s5：プログラムの成長方法
- s6：非終端記号における交叉確率
- s7：終端記号における交叉確率
- s8：コピーのみを行う確率
- s9：評価に用いるスケジュールの数
- s10：小さいプログラムを重視する割合

s3 において初期世代のプログラムがあるとする場合，プログラム（S 式表現）を記述した外部ファイルが必要である．初期世代になるプログラムを指定すると，初期世代の集団の半分程度が指定したプログラムとなり，以降の進化に影響を及ぼすこととなる．s5 では，プログラムをランダムに成長させるか，各枝を必ず s4 の最大深さまで成長させるかを設定できる．s6, s7, s8 は遺伝的操作の頻度の設定である．突然変異が起こるのは，交叉もコピーもしない場合である．s9 および s10 は適合度計算に関係する.

4.2.2. ADF の設定

ADF の初期設定として各 ADF に対して以下の設定項目がある．ここで，ADF の数を n として，各 ADF を ADF_0，ADF_1，･･･，ADF_i，･･･，ADF_{n-1} と表わす.

- ADF_i の新規生成時の最大深さ
- ADF_i の交叉時の最大深さ
- ADF_i の突然変異時の最大深さ
- ADF_i の最低深さ
- ADF_i の成長方法

4.3. ノード

二相ロック法や時刻印順序法などの代表的な排他制御を実現するプログラムを分解して得たノードではなく，より汎用的な排他制御プログラム作成システム

とするために，ノードをより一般的な基本機能である変数の値の操作と定めた[4].

具体的には，データやトランザクションが個別に持つ変数を宣言・操作可能とし，この変数を不定変数と呼ぶことにした．また，排他制御ではデータとそれを操作するトランザクションを関係づけて管理することが多いため，データとトランザクション間で持つ特別な変数も宣言・操作可能とし，この変数を共有不定変数と呼ぶことにした.

提案した 2 つの変数に対する処理を行うノードを代表的な排他制御法である二相ロック法や時刻印順序法を表すことができるように設定し，それらのノードを組み合わせることで新しい排他制御法が得られることを期待している.

4.4. 適合度の計算

適合度の計算に関わるものは大きく分けて 6 つあり，以下の順で計算する.

(1) 直列可能性の検証

直列可能性の検証とは，複数の処理を並行して行った時に，それらの処理を順番に行った時と同じ結果が得られるかどうかを検証することである.

制御済みスケジュールにおいて，直列可能性が保証される場合のみ以降の適合度計算を行う．直列可能性が保証されない場合は適合度を最低に設定する．ここでは,直列可能性の検証法として先行グラフを採用し，競合直列可能性を調べる.

(2) 同時実行性の検証

この計算を行う時は，直列可能性が保証されているので，単純に一度に実行できるトランザクションの数が多ければ多いほど効率的になると考えてよい.

直列スケジュールの WRITE 命令の数を l_{write}，制御後スケジュールにおける同時実行オペレーションの最大同時実行数を $c_{\max}$，同時実行を考慮した総ステップ数でスケジュールの長さを割ったものを平均同時実行数 c_{ave}として，式(1)により得られる値を同時実行性の評価値とする.

$$E_1 = {l_{write}}^2 \times c_{\max} \times c_{ave} \qquad (1)$$

(3) トランザクションの応答時間

トランザクションの集合{$T_1,T_2,\cdots,T_n$}があるとき，T_nの直列スケジュールにおける応答時間を ts_n, 制御済みスケジュールにおける応答時間を tc_nとする. また，n_{abort}はアボートしたトランザクションの数，tsは直列スケジュールにおけるスケジュール開始から終了までの応答時間，tcは制御済みスケジュールにおけるスケジュール開始から終了までの応答時間とする．式(2)の計算を行い，トランザクションの応答時間の評価値とする.

$$E_2 = \frac{1}{2n} \cdot \{(n - n_{abort}) \cdot \frac{ts}{tc} + \sum_{k=1}^{n} \frac{ts_k}{tc_k}\} \qquad (2)$$

ここでの応答時間は,オペレーション READ, WRITE の時間とそのスケジュールを制御するときの不定変数を宣言する時間を足したものとする．実機を用いて評価を行い，READ・WRITE・不定変数の宣言を 10 : 20 : 1 とすることにした.

(4) トランザクションのアボート率

全トランザクションのうちアボートされた割合 p_{abort} を求め，式(3)の計算を行い，トランザクションのアボート率の評価値とする.

アボート率が 0.5 以上の時，性能が悪いことが容易に分かるため，2 乗している.

$$E_3 = \begin{cases} 1 - p_{abort} & if \quad p_{abort} < 0.5 \\ (1 - p_{abort})^2 & otherwise \end{cases} \qquad (3)$$

(5) プログラムサイズによる補正

プログラムサイズを考慮する計算を行う.

具体的には S 式プログラムの記号(非終端記号と終端記号)を数えて N とし，プログラム起動前に設定しておいた，どれだけプログラムの大きさを重視するかの値 α を乗算して得られた値をプログラムサイズの評価値とする.

$$E_4 = N \times \alpha \qquad (4)$$

(6) 全テストスケジュールに対する結果の評価

(1)～(5)までは各テストスケジュールに対しての評価であったが，排他制御プログラムで競合等価な制御済みスケジュールが得られないという結果があっては問題なので，全てのテストスケジュールに対して制御済みスケジュールが競合等価であれば適合度を大きくすることとして，式（5）を採用した.

$$E_5 = \begin{cases} 1.5 & if \quad p_{not\_conflict\_equal} = 0 \\ 1 - p_{not\_conflict\_equal} & else \quad if \quad p_{not\_conflict\_equalt} < 0.5 \\ (1 - p_{not\_conflict\_equal})^2 & otherwise \end{cases} \qquad (5)$$

ここで，$p_{not_conflict_equal}$はテストスケジュールを操作して得た制御済みスケジュールが競合等価ではない割合である．

(7) 最終的な適合度

(1)～(6)までの計算を行い，最終的な適合度Eを求める．今までの結果をまとめると式(6)となる．ここでmax_scheduleはテストスケジュール数，$E_1(s)$はs番目のテストスケジュールに対するE_1である．

$$E = \left(\frac{\left(\sum_{s=0}^{max\_schedule-1} E_1(s) \times E_2(s) \times E_3(s) \right)}{max\_schedule} - E_4 \right) \times E_5 \qquad (6)$$

5. 意味的排他制御

アプリケーションのデータベースへの命令の意味特性を利用することにより，トランザクションの競合可能性を減少させることができると言われている[9]．この特性を利用した排他制御が意味的排他制御である．

例えば，データのログを格納するファイルでは通常挿入のみが実行され，データが追加されていく．ここで注意すべきは，ログを格納したファイルについて変更を行うのか，という点である．ログという性質上格納したデータの変更が行われることはなく，また挿入操作だけであるため，それぞれの WRITE 命令が競合することはない．このとき，ログを格納するファイルに関して WRITE 命令と WRITE 命令は競合しない，と設定することができる．これにより，通常の二相ロック法では大量の競合が発生してしまう状況で，より効率良く命令を捌ける排他制御を行うことができる．このほかにも，キュー，スタックやテーブルについても同様の排他制御が考えられている[9]．このように，オペレーションの使われ方や組み合わせの可否を考慮して同時実効性を向上させるのが意味特性を利用した排他制御である．

Hasse と Weikum は，意味特性を利用し，さらに内部トランザクションの並列化をサポートしたアーキテクチャ PLENTY について論述している[10]．

Barga と Pu は，reflection という考え方に基づいて拡張トランザクションモデルと意味的な排他制御を可能とする枠組みを提案している[11]．また，CORD を用いることで意味的な排他制御を可能にすることもできる[7]．

6. 意味的排他制御の導入

6.1. アプローチ

これまで筆者らが開発してきたプログラムに意味的排他制御を組み込むにあたり，当初の目的である排他制御の自動生成との両立が可能であるか検討を行った．先に示した通り，各環境で意味的に緩和する競合は当然環境によって異なるため，それを自動で判別する方法は存在しなかった．また，仮にある程度の精度で自動判別できたとしても，今度は判別した緩和すべき競合が本当に正しいのか，という問題が発生する．

二相ロック法で問題となる点，つまり，二相ロック法では競合判定が厳しすぎる点について緩和により解決を試みている他の研究[7][10][11]では，既存の DBMS に上乗せする形で，競合直列可能でない状態が発生した時に外部に記述した緩和する競合かどうかを判別し，緩和すべき競合であればその競合を許し，そうでなければ通常の競合と同様に操作させるアーキテクチャを採用している．そこで，本研究も緩和すべき関係についてはユーザに記述してもらう形式を取ることとする．

また，最終的には複数の READ,WRITE を一つのオペレーション(例えば，商品を販売した時の操作 SELL()など)として見ることで，より幅の広い競合緩和を目指すこととする．今回はその前段階として，READ と WRITE の競合を緩和する関係のみを記述して，その関係についてのみ緩和を行うこととする．

6.2. 実装

実装にあたり，まず，図 4 のように競合緩和関係を記述してもらい，これを読み込むこととした．また，この競合緩和関係の設定から，適合度計算の直列可能性の検証に使用している先行グラフの有向辺の生成法を変更することとした．

loosen_conflict.txt

```
#'#'で開始する行はコメント
#競合を許すなら true,競合を許さないなら false を記述
#デフォルトでは
#rr true
#rw false
#wr false
#ww false
rr true
rw false
wr false
ww true
```

図 4．競合緩和関係を記述したファイル

先行グラフにより，競合直列可能であるか否かの判定をそのグラフが非巡回であるかどうかで簡単に判別できる．先行グラフの作成アルゴリズムを次に示す．

スケジュール S を構成するトランザクションを{T1, T2,…,Tn}とし，あるトランザクション Ti によるデータ項目 A に対する読み出しを Ti:READ(A)，書き込みを Ti: WRITE(A)と書くこととする．このとき，スケジュール S において，次の条件を満たす時ノード Ti とノード Tj に有向辺を引く．結果のグラフを先行グラフといい，これが非巡回であれば競合直列可能であり，そうでなければ競合直列可能ではない．

(1) Ti:READ(A)が Tj:WRITE(A)に先行する．
(2) Ti:WRITE(A)が Tj:READ(A)に先行する．
(3) Ti:WRITE(A)が Tj:WRITE(A)に先行する．

ここに，先ほど示した競合緩和関係を考慮することで，緩和した状態での競合直列可能を判定することができる．

図 4 に示した記述をもとに有向辺を追加する条件を以下に示す．

(1) Ti:READ と Tj:READ の競合を許さない，かつ Ti:READ(A)が Tj:READ(A)に先行する．
(2) Ti:READ と Tj:WRITE の競合を許さない，かつ Ti:READ(A)が Tj:WRITE(A)に先行する．
(3) Ti:WRITE と Tj:READ の競合を許さない，かつ Ti:WRITE(A)が Tj:READ(A)に先行する．
(4) Ti:WRITE と Tj:WRITE の競合を許さない，かつ Ti:WRITE(A)が Tj:WRITE(A)に先行する．

これは例えば，図 4 では WRITE と WRITE で競合を許すとしている（ww true）ので，(4)では先行グラフに有向辺を追加しない．一方，READ と WRITE では競合を許さないとしている（rw false）ので，(2)で有向辺を追加することになる．

7. 実験

7.1. 初期設定

排他制御プログラムの生成実験で使用するトランザクションの初期設定と遺伝的操作の初期設定をそれぞれ，表 1 と表 2 に示す．また，意味的排他制御として図 4 で示した例を用いる．

ここでは，操作すべきデータ数 p2 をトランザクションの数 p1 より小さくし，かつ意味的排他制御が結果に影響しやすいように WRITE の発生確率を高く設定して意図的に多数の競合が発生するようにして実験を行った．初期世代になるプログラムを指定した場合には，初期世代の半分が指定したプログラムで構成される．

表 1　トランザクションの特徴設定

設定項目	内容	値
p1	同時に実行するトランザクションの数	10
p2	トランザクションが操作するデータの最大数	3
p3	命令（オペレーション）に WRITE が含まれる確率	0.8
p4	トランザクションが発するオペレーションの最大数	10
p5	トランザクションが途中で終了する確率	0.1

表 2　GP の初期設定

設定項目	内容	値
s1	世代数	5000
s2	一世代における個体数（集団サイズ）	1000
s3	初期世代になるプログラム指定の有無	実験により異なる
s4	プログラムの最大深さ	8
s5	プログラムの成長方法	ランダム
s6	非終端記号における交叉確率	0.1
s7	終端記号における交叉確率	0.6
s8	コピーのみを行う確率	0.1
s9	評価に用いるスケジュール数	100
s10	小さいプログラムを重視する割合	0.005

7.2. 実験結果

ここでは，初期木のない設定と初期木として二相ロック法を用いた設定の二つについて実験を行った．その結果得られたプログラムと二相ロック法のプログラムに対して，同じ設定で生成した別のスケジュール 100 個について 4.4 節で示した適合度の計算を行った．その結果を表 3 に示す．

表 3　各プログラムに対しての適合度

プログラム	適合度
二相ロック法	2.97×10^{4}
生成されたプログラム 初期木設定：なし	1.42×10^{4}
生成されたプログラム 初期木設定：二相ロック法	2.97×10^{4}

この結果から分かるように，二相ロック法よりも優秀なプログラムを生成することができなかった．原因としては，初期木の指定なしではプログラムとしての基本的な形が作れなかったからではないかと考えられる．また，初期木を二相ロック法にした場合でも，初期収束が発生し二相ロック法よりも効率的なプログラムを生成することができなかったと考えられる．

初期木を指定せずに生成したプログラムはそのトランザクションの最初のオペレーションが読み込み命令(READ)ならアボート，書き込み命令(WRITE)の後に読み込み命令がきたなら動作待ち(同じデータに対して処理を行う他のトランザクションのオペレーションを先に実行させる)を起こすアルゴリズムであった．当然のことながら，アボート率が二相ロック法に比べて上昇し，優秀なプログラムを生成することができなかった．

ここで，表 1 の p1 を 1，p3 を 1.0 として実験を行ったところ，何もしないというプログラムが生成されたので，意味的な排他制御への対応自体は適正に行われている．

また，生成されたプログラムのノード数が 9，二相ロック法のプログラムのノード数が 177 であったため，現状のノードでは，うまく組み合わせることが困難なことが分かった．そのため，ノードについて再考し，より性能のよいプログラムが生成しやすいようにしてゆく予定である．

8. おわりに

本論文では，単純な排他制御だけではなく意味特性を利用した意味的排他制御プログラムを自動生成する手法について述べた．実験を行い，その結果得られたプログラムと既存の二相ロック法のプログラムを比較したが，よりよいプログラムが得られたとはいえなかった．

これからは，現在のノードで有効に動作させる方法はないかを検証し，ないのであればノード自体の再構成を行い，今回導入できなかった他の競合緩和手法や複数のオペレーションを 1 オペレーションとして組み合わせた意味的排他制御を行っていきたい．また，生成したプログラムの正当性の検証も今後の課題である．

参 考 文 献

[1] M. Stonebraker and U. Cetintemel : “One Size Fits All : An Idea Whose Time Has Come and Gone”, Proc. of the 21st International Conference on Data Engineering, pp. 2-11 (2005).

[2] M. Seltzer : “Beyond Relational Databases”, Commun. ACM, Vol. 51, No. 7, pp. 52-58 (2008).

[3] 水野正義，宝珍輝尚，野宮浩揮：“遺伝的プログラミングによる排他制御プログラムの生成について”, 情報処理学会研究報告, 2009-DBS-148(27), 2009-FI-95 (27) (2009).

[4] 田村真司，宝珍輝尚，野宮浩揮：“遺伝的プログラミングによる排他制御プログラムの生成”, DEIM Forum 2010 E8-3 (2010).

[5] 田村真司，宝珍輝尚，野宮浩揮：“自動関数定義を用いた遺伝的プログラミングによる排他制御プログラムの生成”, 平成 22 年度 情報処理学会関西支部 支部大会 E-09 (2011).

[6] ジム・グレイ，アンドレアス・ロイター(喜連川優監訳)：“トランザクション処理(上)”, 日経 BP 社 (2001)

[7] G. T. Heineman, and G. E. Kaiser, "The CORD approach to Extensible Concurrency Control," Proc. of the 13th International Conference on Data Engineering, pp.562-571 (1997).

[8] 伊庭斉志：“遺伝的プログラミング”, 東京電機大学出版(1996).

[9] B. R. Badrinath and K. Ramamritham, “Semantics-Based Concurrency Control: Beyond Commutativity”, ACM Transactions on Database Systems, Vol. 17, No.1, pp.163-199 (1992).

[10] C. Hasse and G. Weikum, “Inter- And Intra-Transaction Parallelism For Combined Oltp/Olap Workloads”, Advanced Trnsaction Models And Architectures (Sushil Jajodia and Larry Kerschberg eds.), Kluwer Academic Publishers, pp.279-299 (1997).

[11] R. S. Barga and C. Pu, “A Reflective Framework For Implementing Extended Transactions”, Advanced Trnsaction Models And Architectures(Sushil Jajodia and Larry Kerschberg eds.), Kluwer Academic Publishers, pp.65-89 (1997).