Vol. 34 No. 8 情報処理学会論文誌 Aug. 1993

共有データ空間モデル上での並列プログラムの誤り処理法

河本達哉† 井上謙藏††

よく知られているように，並列プログラムの誤りは，実行過程の再現性が乏しいため，その原因の究明が困難であり，したがって組織的に誤り原因を追及する方法にも乏しい．最近並列プログラムのモデルとして，各プロセス間のデータの授受を共有データ空間を通じてのみ行う共有データ空間モデルが考案され，それでほとんどすべての並列プログラムを記述できることが示されている．しかし，このモデルはその性質から並列プログラムの誤り原因の追及にも有力な手段を提供するものである．本論文は，このモデルの一つであるLinda-モデルの上で実現された我々のシステムによる，並列プログラムの誤り原因発見の組織的な方法を述べるものである．

A Debugging Method of Concurrent Programs on Shared Dataspace Model

Tatsuya Kawamoto† and Kenzo Inoue††

As generally known, debugging concurrent programs is very difficult because of race condition of their execution processes. Therefore we have almost no systematic method for debugging of programs of this type. Recently, a shared dataspace model has been proposed for concurrent programs, which permits data-interchanges between processes of them only through this space, and it has been shown that almost all concurrent programs could be described on this model. However, this model gives a powerful method for debugging of these programs because of data-concentration into a dataspace, also. In this paper, it is described a systematic debugging method for concurrent programs developed upon a system implemented on Linda-model being a type of shared dataspace model.

1. はじめに

並列プログラムは，複数のプロセスが役割を分担しながら作業を進める「プロセス協調」と哲学者の食事問題のようにデータの取得を競う「プロセス競合」の処理に大別される[1),2)]．これら並列プログラムの実行状況は一般にその内外のプロセスの速度に依存するため，誤りが発生すると，その症状は再現性に乏しく，原因の究明は極めて困難である．

並列プログラムの誤り原因を明らかにするには，基本的には，システム全体の包括的な知識が必要である．例えば，「プロセス協調」の場合には，プロセス間の関係と役割配分，誤り時点での全体の進行状況などの知識が誤りに関係するプロセスの割出しに必要である．「プロセス競合」では，あるプロセスが動くための必要物をほかのプロセスがどのように提供するかの認識が，問題のプロセスを特定するのに必要になる．既存の並列プログラムのモデルでは，このような情報の提供は困難であり，したがって既存の誤り処理の手法では，このような情報の利用は欠如している[3)]．

最近，プロセス間のデータ交換を一つの空間を介して行う方法で並列プログラムを実行する共有データ空間モデルと呼ぶ新しい方式が提案され，それでほとんどすべての型の並列システムを実現できることが示され，制約問題解決や黒板モデルにも応用可能なパラダイムとして注目を集めている[4)~9)]．

このモデルでは，共有データ空間の遷移記録は並列システムの完全な実行経歴であるから，それは誤り解析のための基本的，包括的な知識を与える．本論文は，このモデルの一つであるLinda-モデル上での[10),11)]，並列プログラムの誤り処理法を提案するものである．

第2章で本論文の誤り処理法を実現するために必要

† 東京工業大学理工学研究科情報工学専攻
Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology
†† 東京理科大学理工学部情報科学科
Department of Information Sciences, Faculty of Science and Technology, Science University of Tokyo

な Linda-モデルの拡張について論じ，第3章では共有データ空間の管理法を説明し，第4章は我々の開発したシステムによる誤り処理の方法について詳説する．

2. Linda-モデルとその拡張

Linda-モデルは，任意のプログラム言語に，共有データ空間に触る4種のオペレーションin, rd, out, eval を埋め，その実行によって空間とのデータの授受，プロセスの生成，消滅のすべてをつかさどる．そこでこのモデルはどんな言語の上でも，単純に並列プログラムを実現できる．この共有データ空間をタプル空間（以下空間と略称），その中のデータをタプル（以下 TP），上記のオペレーションを Linda-オペレーション（以下 OP）と呼ぶ．我々の実験ではC言語の上にモデルを実現した．それゆえ，プログラムは関数の集合であり，そのコードは OP 以外はCのコードである．

空間から見るとプロセス（以下 PR）はブラック・ボックスであるから，空間内の特定の TP をどの PR のいずれの OP が生成したかはわからない．これでは誤りに関する情報の収集は不可能である．そこで，我々は，次の機能的な拡張を行った．すなわち，OP の表面的な記述には触れず，C言語の処理系を改造し，各 OP に ID として対象並列プログラム全体を通じての一貫番号（OPID）を付加し，引数に型を付けた（通常の言語処理系では OPID は関数名とその中での OP 番号の対となる）．

空間はタプル空間マネージャ（Tuple Space Manager, TSM）が管理し，そこには受動的データのタプル（Passive Tuple, PT）と，PR の生成とともに生成され，その消滅を待つライブタプル（LT）が存在する（**図1**）．このほか，我々は in-アンチタプル（IT），rd-アンチタプル（RT）を付加した（合わせて ATとする）．これらの役割は，OP の説明の中で明らかになる．先に導入した TP は PT, LT, IT, RT の任意のものを意味する．

本実験では，TSM はオペレーティング・システムの力を借りて実現した．

(1) in-OP

OP の形式は in（TP 記述）である．TP 記述は TP の形式を与えるもので，その内容はCの関数の実引数の組であるが，ポインタは含めない．in-OP はその TP 記述に一致する PT を TSM に求める．空間中にそれが存在すれば TSM はそれを空間から除いて，in-OP に渡す．in-OP は入力 PT に合わせて変数に値を代入し，次の演算に進む．存在しなければ，TSM はその in-OP から IT を作り，空間に挿入し，in-OP は停止する．

例えば，in("aaa", ?x1, 10) に対応する PT は，

PT (str "aaa", int 35, int 10)

である（下記参照）．第2引数以外の値は in-OP のものと一致している必要がある．このような PT があれば，TSM はそれを空間から除き，in-OP に与える．in-OP は x1 に値35を代入し，次の演算に進む．該当 PT がなければ TSM は IT を生成し，空間に挿入し，in-OP は停止する．上記 in-OP, IT, PT の正確な形式は，

in(n_{o1}): (str "aaa", int ?x1, int 10),

IT(n_{p1}, n_{o1}, An_1):

(str "aaa", int ?x1, int 10)

PT(n_{p2}, n_{o2}, Tn_2):

(str "aaa", int 35, int 10)

である．n_{o1}, n_{o2} はそれぞれ IT を生成した in-OP, PTを生成した out-OP の OPID である．n_{p1}, n_{p2} はそれぞれの OP を実行する PR の ID (PRID) で，PR 生成の際に TSM が割り当て，TP 生成の際に TP に付加する．

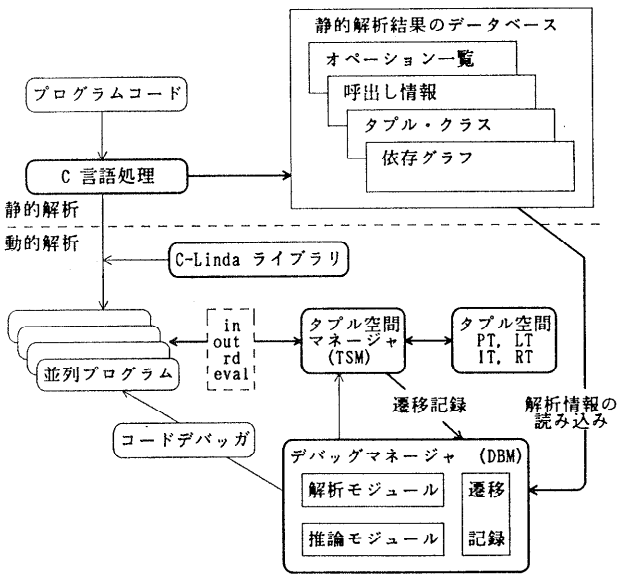


図1 並列プログラムの誤り処理システムの概略

Fig. 1 An outline of debugging system for concurrent programs.

An_1 ("A" n_1), Tn_2 ("T" n_2) は TP の ID で，TP が空間に保存される時に TSM が割り当てる（第3章参照）．

TP 記述中の記号？による引数の修飾は，Linda の規定により，x1 が入力引数であることを意味する．また TP 記述の最初の引数が記号列（例えば，上の "aaa"）であるときはそれをタグと呼び，TP の型を分類する重要な情報と見なす．

(2) rd-OP

OP の形式は rd (TP 記述) である．rd-OP は読み取りの際に空間から PT を取り除かないこと，PT が存在しない場合は rd-OP を RT とすることを除いては in-OP と同様である．RT の形式は IT に準ずる．

(3) out-OP

OP の形式は out (TP 記述) である．out-OP は，変数引数には値を入れて，TP 記述の形式の PT を生成し，TSM に委ね，次の演算に移る．TSM は，空間にこの形式の RT が存在すれば，そのすべてを取り除き，それらの生成した rd-OP に PT のコピーを渡し，rd-OP の実行は再開される．さらに，一致する形式の IT があれば，その1個に対して同様の処理をし，PT を空間から取り除く（(1)，(2)参照）．IT がなければ，PT に PTID を付して，空間に保存する（第3章参照）．

例えば，out ("bbb", x2, 10) を実行すると，x2 の値が5であれば，

$PT(n_{p1}, n_{01}, —)$: (str "bbb", int 5, int 10)

が生成し ('—' は空白を表す)，このとき空間に

$IT(n_{p2}, n_{02}, An_2)$:
(str "bbb", int ?x3, int 10),

$RT(n_{p3}, n_{03}, An_3)$:
(str "bbb", int ?x4, int ?x5)

が存在すれば，これらの IT, RT は空間から取り除かれ，変数，x3, x4, に値 5, 変数 x5, に 10 が入り，それぞれの OP は再開され，PT は消滅する．

以下の説明では，個々の PR や OP は，それらの ID で区別し，PR : n_p, OP : n_0 等と表す場合がある．

(4) eval-OP

OP の形式は eval (TP 記述) である．例えば eval ("ccc", f3(2), 11) の実行に際しては，TSM は下記の LT を作り，空間に挿入し，さらに f3(2) の PR を生成し，起動する．OP を実行した PR は次の演算に移る．生成された PR は，終了の際に値を TSM を通じて対応 LT に引き渡す．TSM は，LT の引数 f3(2) をその値で書き換え，LT に発生 PR がもはや存在しなければ，それを PT に変形し，PT としての処理を行う．

上の eval-OP に対応する LT の形式は

$LT(n_{p1}, n_{01}, Ln_1)$
: (str "ccc", $pr(n_{p2})$: int f3(int 2), int 11)

である．Ln_1("L" n_1) は LT の ID で，LT の生成の際に，TSM が割り当てる．ただし，LT は将来 PT となるので，番号 n_1 は PT の系統から採用される．n_{p2} は PR f3(2) の PRID である．PR f3(2) の関数値を 10 とすると，この LT は，次の PT に変化する．

$PT(n_p, n_0, Tn_1)$: (str "ccc", int 10, int 11)

3. タプル空間とその管理

空間は，TSM が管理する2個の環状リストと1個の線形リストから成る（**図2**）．PT は一つの環状リスト，PT-LIST，に記録し，RT と IT は他の環状リスト，AT-LIST，に記録する．いずれもリストに沿って動く記入口と取出口がある．線形リスト，PR-LIST，は生成された PR の記録および対応する LT を記録する．

TSM は PT を受けとると AT-LIST を索引し，対応する PT があればそれをすべてリストから取り除き，対応する IT があればその一つを取り除く．

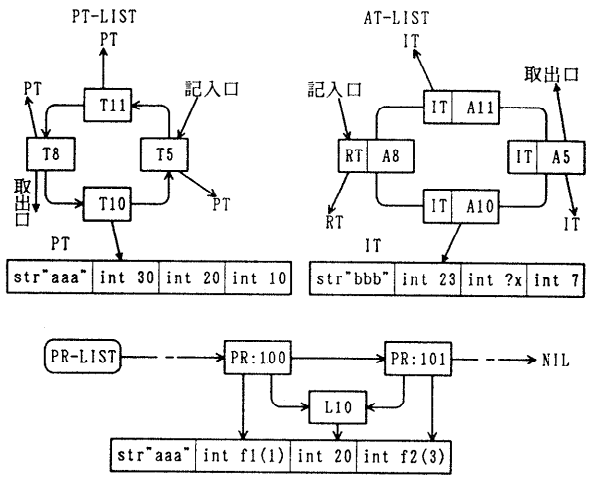


図2 タプル空間の構造
Fig. 2 The structure of tuple-space.

表 1　哲学者の食事問題の空間遷移記録
Table 1　A transition record of tuple-space for dining philosophers' problem.

遷移記録		注
時刻	発生データ記録と対応タプルの ID	
1	LT(0, 0, L0):(PR(561): int real_main("command args..")) :(-)	主プロセス生成用特殊 LT
2	PT(561, 1, T1):("chopstick", int 0):(-)	箸 0→空間
3	LT(561, 2, L2):("done",PR(562): int phil(int 1,int 5)):(-)	PR:562 生成
4	PT(561, 3, T3):("ticket", int 0) :(-)	切符 0→空間
5	PT(561, 1, T4):("chopstick", int 1):(-)	箸 1→空間
..	....	
12	LT(561, 2,L11):("done",pr(565): int phil(int 4,int 5)):(-)	PR:565 生成
..	....	(str は省略)
16	IT(561, 4, A1):("done",int ?m) :(-)	
17	IT(562, 5, -):("ticket", int ?m) :(T3)	←切符 0
18	IT(562, 6, -):("chopstick", int 1):(T4)	←箸 1
..	....	
28	IT(566, 5,A13):("ticket", int ?m) :(-)	
29	PT(562, 8, -):("chopstick", int 2):(A6)	→A6
30	PT(562, 9,T16):("chopstick", int 1):(-)	箸 1→空間
31	PT(562,10, -):("ticket", int 0) :(A13)	→A13
32	IT(566, 6,A14):("chopstick", int 5):(-)	箸 5→空間
..	....	
127	TM(565, 0, -):(int 4) :(L11,A1)	PR:565 消滅
128	IT(561, 4,A61):("done", int ?m) :(-)	
129	PT(562, 8,T66):("chopstick", int 2):(-)	
..	....	この間 PR:562/564 消滅
144	PT(563,10,T74):("ticket", int 2) :(-)	
145	TM(563, 0, -):(int 2) :(L5,A63)	PR:563 消滅
146	IT(561, 4,A67):("done",int ?m) :(-)	

```
プログラム                                  OPID    注

real_main( )
  {int i,loops,m;
   int phil( );
   loops=5;
   for(i=0;i<5;i++)
     {out("chopstick",i);                    1
      eval("done",phil(i+1,loops));          2     i+1 は i の誤り
      if(i<(5-1)) out("ticket",i);           3     切符は 4 に制限
     }                                             (デッドロックの防止)
   for(i=0;i<5;i++)
       {in("done",?m);}                      4
  }

int phil(i,loops)
  int i,loops;
    {int j,m;
     for(j=0;j<loops;j++)
       {in("ticket",?m);                     5
        in("chopstick",i);                   6
        in("chopstick",(i+1)%5);             7
        out("chopstick",(i+1)%5);            8
        out("chopstick",i);                  9
        out("ticket",m);                     10
       }
     return(i);
    }
```

図 3　哲学者の食事問題
Fig. 3　Dining philosophers' problem.

IT がなければ PT に PTID を付し，PT-LIST に記入する（2章(3)参照）．

in-OP，rd-OP の場合は，PT-LIST を索引し，対応する PT を求めて，それをリストより除き，それぞれの処理を行う．対応 PT がなければ，それぞれ IT，RT を作り ID を付し，AT-LIST に入れる（2章(1)，(2)参照）．

eval-OP の場合は，その引数の関数呼出しから PR を生成し，その記録を PR-LIST に記入し，LT を作り，LTID (PTID の系列の番号) を付し，PR-LIST 上の生成 PR 記録に結合する．PR が消滅したときは，TSM はその PR の記録を PR-LIST から除き，LT の対応引数を PR の値で書き換える．もし LT に，ほかに生成 PR がなければ，LT を PT に変える（2章(4)参照）．

タプル空間への入出力動作は排他的に制御されるので，空間が授受する情報は時系列を形成し空間の遷移記録となる．TSM はデバッグマネージャ (DeBug Manager，DBM) の要求によってこの記録を DBM へ送る．**表 1** は，並列プログラムの周知のモデルである哲学者の食事問題（**図 3**）から生ずる遷移記録である[12]．このプログラムは，その注に示す誤りを含む．

遷移記録の形式は

番号，発生データ記録：(対応 TPID)

である．番号は 1 で始まる一貫番号で，タプル空間時刻と呼ぶ．データ記録はその時刻に空間が授受した TP または TM の情報（下記）である．TP の情報は

TP (PRID，OPID，TPID)：(TP 記述)

で，その時点で発生した TP であるが，in-OP，rd-OP から AT が生成されないときも，それを IT，RT とし，その ID を空白（'—' で表記）として記録する（表 1，時刻 17，18 等）．TM は PR の消滅時点の記録のための情報で，TM (PRID, 0, —)：(値) の形を持つ．PRID と値はそれぞれ消滅した PR の ID と関数値である．TM は TP でないから，ID は存在しない（時刻 145）．

対応 TPID はこの TP や TM の発生によって，その時点で影響を受けた TP の ID 列である．すなわち PT の場合はそれを受け入れた IT，RT の ID（時刻 29），IT，RT の場合

はそれらが受け入れた TP の ID である（時刻17）．TM の場合は，消滅した PR を発行した LT の ID のほかに，その LT がその時 TP に変化し，IT，RT に受け入れられたら，それらの ID も列挙する（時刻145）．TP，TM の発生が，その時点でほかに影響を与えないときは対応 ID 列は空白である（時刻1，2等）．

プログラムの実行に異常が発見されたときは，システムの使用者は割り込みにより実行を停止させ，プロセスを強制的に消滅させることができる．表1の遷移記録は，異常停止に対して（PR：561，566 のロックアウト），割り込んで終了させたものである．この場合の遷移記録は割り込みの直前までの記録となる．

一般化の観点から空間を多重化する試みがあるが[13),8)]，複数の互いに独立な並列プログラムを動かすには多重化は避けられない．すなわち，リスト PT，AT，PR の一組ごとに空間時刻を設定し，並列プログラムには ID を付して，この一組に対応させる．空間が授受する TP にはこの ID を追加する．このようにして，同一 TP 記述の他並列プログラムの TP との混同を防ぎ，空間履歴のインタリーブを避ける必要がある．以下の議論はこの部分空間に対して，そのまま成立する．

4. 誤り処理の方法

並列プログラムは，PR の実行とそれが授受するデータの間のバランスが取れて，初めて正しく動く．Linda-モデルでは，これは PR による TP の確保や開放と PR の生成や消滅のバランスということになる．このバランスの乱れを見つけ，その原因を探ることが誤り処理の要点である．しかし，この乱れを発見するには，経験的な勘に頼らざるを得ない．ここで提案する処理法は経験的な手法を組織的に適用できるデータの収集とその上での処理手段の展開を骨子とする．

誤り処理は，DBM の管理下に行う静的解析と動的解析より成る（図1）．静的解析は，C 言語処理系からの出力を入力とし，動的解析のための予備情報の収集を行う．動的解析は静的解析の結果と遷移記録に基づき，プロセス主体解析，タプル主体解析，経験則解析，状態分布解析，プロセス実行の再現の順序で進む．

プロセス主体解析とタプル主体解析は経験則解析のためのデータの収集で，前者では PR の動的な関係を把握し，後者は TP の用途を明らかにする．

経験則解析では，経験を表現する手続きにより，それまでに作られた結果に対して推論処理を行い，異常な振舞いをする．したがって誤りの原因にかかわる TP，PR を抽出する．システムの使用者はこの結果から，プログラムの実行にあってはならないそれらの組み合わせ，空間パターン（後出），を作る．状態分布解析ではこの空間パターンが遷移記録上のどの時間帯に存在するかを検索する．それが発見されれば，空間のこの部分の形成に関与する1個または少数の PR とその状態を特定できる．これらの PR は誤りを含むものである．

以上の解析で不十分な場合には，特定された PR の実行を再現する必要がある．Linda-モデルでは，他の PR との関係はすべて空間を通じて行われ，我々のシステムでは空間の変遷を遷移記録として残す．したがって，特定の PR を動かす際に，空間との間で授受する TP は，それが必要となる時点で，DBM が遷移記録から供給することができる．そこで PR が授受する TP は，誤り発生状況での値そのものとなり，その PR の実行は誤り発生時の実行過程そのものの再現となる．

このようにして，誤りに関与する PR のひとつひとつの実行を，独立に再現する過程で，逐次的なプログラムの誤り処理手段を実施すれば，最終的に誤り要因を指摘できる．逐次的な処理に関しては省略する．

4.1 静的解析

静的解析ではC言語処理系の出力から次のデータの収集を行う．

(1)OP 一覧
(2)PR 間呼出し情報
(3)OP の TP クラス
(4)OP 間依存グラフ

OP一覧は関数ごとに，OPをテキスト上の順に従って列挙したもので，関数ごとの表はCの処理系が作る．静的解析では個々の関数の表を収集するだけである．

OP 一覧から(2)，(3)，および(4)が求められる．

OP 間呼出し情報は eval-OP から求められる．例えば，eval (“ccc”, f3 (int x), 11) から，これが関数 f1 の中に存在すれば，呼出し関係 f1→f3 が求まる．すべての eval-OP から求められた呼出し関係の集合が PR 間呼出し情報である．これは可能な呼出し関係であり，動的にその呼出しが生ずるとは限らない．

TP クラスとは，OP の TP 記述の型情報に基づ

く分類である．例えば in (“aaa”, ?x1, 10), out (“aaa”, x2, 10) の型情報は (“aaa”, int, int) で，eval (“ccc”, f3(x), 11) の場合は f3 の値を整数として (“ccc”, int, int) である．(1)で集められたすべての OP はこの型情報に基づいて，それぞれのクラスに分けられる．

同一の TP クラスの out-OP, eval-OP と in-OP, rd-OP 間では TP の授受が可能である．そこでこれらの OP 間で，前者から後者へ有向枝を引く．これがそのクラスの OP 間依存グラフである．この有向枝も，動的に TP の授受が生ずることを示すものではない．

4.2 動的解析

すでに述べたように動的解析の入力は静的解析の結果と空間の遷移記録である．

(1)プロセス主体解析

この解析では PR の親子関係とデータの出入情報を収集する．まず遷移記録を順次に検索し，各 PR に対して，PRID，関数名，PR の生成 LT，および生成時刻を集める．このデータからただちに PR の親子関係が作られる．静的解析からの依存グラフでは静的に TP の授受が可能であるが，動的には授受がない OP の対応を把握できる．表1の例ではこのずれは発見されない．

さらに，PT を授受する OP の対を特定し，PR 間の動的な依存状況を把握する．その結果は，

((発行側 PRID, OPID),
(受入側 PRID, OPID), 回数)

で表される．発行側は PT (またはその元になる LT) 発行側，受入側はこの PT を受け入れた側，回数は授受の生じた回数である．表1からは，**表2**の結果を得る．

表 2　プロセス主体解析によるプロセス間タプル授受回数
Table 2　Numbers of tuple-transfers between processes got by process-oriented analysis.

	タプル授受回数	注	
		発(LT, PT)受	対応時刻
1	((561, 2), (561, 4), 4)	M(P1, ··, 5)M	3/16/132, ·· (LT/TM/IT)
2	((561, 1), (562, 6), 1)	M(C1)P1	5/18 (PT/IT, 以下同)
3	((561, 1), (562, 7), 1)	M(C2)P1	8/19
4	((561, 3), (562, 5), 1)	M(T0)P1	4/17
5	((561, 3), (563, 5), 1)	M(T2)P2	10/20
6	((561, 1), (564, 6), 1)	M(C3)P3	11/23
7	((561, 3), (564, 5), 1)	M(T3)P3	13/22
8	((561, 1), (565, 6), 1)	M(C4)P3	14/25
9	((561, 1), (565, 7), 1)	M(C0)P4	2/26
10	((561, 3), (565, 5), 1)	M(T1)P4	7/24
11	((562, 9), (562, 6), 4)	P1(C1)P1	30/38, ··
12	((562, 8), (563, 6), 5)	P1(C2)P2	29/21, ··
13	((562, 10), (563, 5), 2)	P1(T2, 1)P2	83/88, ··
14	((562, 10), (565, 5), 1)	P1(T1)P4	57/59
15	((562, 10), (566, 5), 1)	P1(T0)P5	31/28
16	((563, 9), (562, 7), 4)	P2(C2)P1	47/39, ··
··	··	··	··
19	((563, 10), (565, 5), 2)	P2(T3, 2)P4	77/79, ··
20	((564, 10), (562, 5), 3)	P3(T2, 1, 3)P1	62/67, ··
··	··	··	··
23	((564, 10), (565, 5), 1)	P3(T3)P4	43/41
24	((565, 10), (562, 5), 1)	P4(T1)P1	36/37
··	··	··	··
28	((565, 8), (565, 7), 4)	P4(C0)P4	34/45, ··

注：M, P··主プログラム、哲学者のプロセス
T, C··切符、箸のタプル

(2)タプル主体解析

静的解析で作られる TP クラスは，それに属する OP から作られる TP の集合を自然に形成するが，この集合から，さらに PR 間に授受される TP の特定クラスが抽出される．このクラスに属する TP を特に資源と呼ぶことにすると，資源量の時間的推移を求めることが，タプル主体解析の目的である．

資源のクラスを生成するために，TP クラス型情報を用い，資源分類パターンを作る．具体例で説明する．表1の遷移記録は，型情報 (str “chopstick”, int), (str “ticket”, int), (str “done”, int) を含む．DBM はシステムの使用者に，この型情報を示して，それからどういう分類をしたいかを質問する．そこで当面の並列プログラムの意味的考察から，最初の型情報の第2引数は値ごとに分類するとして記号！を付し，残りのものの第2引数は任意の値を取り得るとして記号？を付け，次のパターンを作成したとする．

(str “chopstick”, int !),
(str “ticket”, int ?), (str “done”, int ?)

これが資源分類パターンである．これを用い表1の遷移記録を順次に検索する過程で，自然に第一のパターンには (str “chopstick”, int 0), ···, (str “chopstick”, int 5) が，第二のものには (str “ticket”, int ?) が，最後のものには (str “done”, int ?) が，それぞれ適応パターンとして選ばれ，かつそれらに属する TP が分類され，その量の時間的推移が求められる．これらのパターンが資源パターンで，それぞれの資源パターンに属する TP がそのパターンの資源である．

資源量の推移は，各資源パターンごとに，

時刻 TP (PRID, OPID) : (n_P, n_A, n_L ; b_R)

の形で表現する．TP はこの時刻に空間に挿入された TP の種類で，n_P, n_A, n_L は，それぞれこの TP の

表 3 タプル主体解析による資源量の推移
Table 3 Changes of resource-quantities got by tuple-oriented analysis.

資源パターン	時刻	資源量の推移
(str"chopstick", int 0)	2	PT(561, 1):(1,0,0;0)
	..	....
	119	PT(565, 8):(1,0,0;0)
(str"chopstick", int 1)	5	PT(561, 1):(1,0,0;0)
	..	....
	130	PT(562, 9):(1,0,0;0)
(str"chopstick", int 2)	8	PT(561, 1):(1,0,0;0)
	..	....
	143	PT(563, 9):(1,0,0;0)
(str"chopstick", int 3)	11	PT(561, 1):(1,0,0;0)
	..	....
	142	PT(563, 8):(1,0,0;0)
(str"chopstick", int 4)	14	PT(561, 1):(1,0,0;0)
	..	....
	134	PT(564, 8):(1,0,0;0)
(str"chopstick", int 5)	32	IT(566, 6):(0,1,0;0)
(str"done", int ?)	3	LT(561, 2):(0,0,1;0)
	..	....
	146	IT(561, 4):(0,1,1;0)
(str"ticket", int ?)	4	PT(561, 3):(1,0,0;0)
	..	....
	144	PT(563,10):(3,0,0;0)

挿入の結果としての，同一資源パターンに属する PT, AT, LT の個数を示す．b_R は挿入されたタプルが RT で，それに対応する PT が存在して rd-OP が完了するか，挿入された TP が PT で，存在していた RT を消滅させた場合に 1，それ以外の場合は 0 とする．表 1 から求められる資源量の推移を，**表 3** に示す．

資源量の変化から，それぞれの資源パターンに対し，プロセスを資源の生産者(PT の生成 PR)，利用者(IT, RT による PT の引用 PR)，参照者 (RT のみによる PT の引用 PR)，消費者 (IT のみによる PT の引用 PR) に分類できる．表 3 では資源パターン (str "chopstick", int 5) に対しては PR : 566 が消費者である．

表 4 経験則解析の結果
Table 4 A result from analysis by empirical rules.

	[関連 PR] [関連 TP] [関連 OP][使用規則] 状態
..	.....
i_1	([566], [(str "chopstick", int 5)], [6], [r1], op_mismatch)
i_2	([561], [(str "done", int ?)], [2, 4], [r2], err_use_res)
i_3	([562], [(str "chopstick", int 1)], [9, 6], [r2], err_use_res)
i_4	([565], [(str "chopstick", int 0)], [8, 7], [r2], err_use_res)
i_5	([562, 563, 564, 565, 566], [], [], [r3], worker(phil))
i_6	([566], [("ticket", int ?)], [], [r3, r4], non_released_res)
i_7	([562, 563, 564, 565], [], [8, 6], [r3, r5], charac_op_pair)
i_8	([562, 563, 564, 565], [], [9, 7], [r3, r5], charac_op_pair)
i_9	([562, 563, 564, 565], [], [10, 5], [r3, r5], charac_op_pair)
i_{10}	([562], [], [9], [r3, r5, r6], abnormal_op)
i_{11}	([565], [], [8], [r3, r5, r6], abnormal_op)
i_{12}	([566], [], [8, 9, 10], [r3, r5, r6], abnormal_op)
..	.....

(3)経験則解析

経験則解析とはプロセス主体解析，タプル主体解析の結果を入力する，経験則の集合を用いて行われる前向き推論である．経験則は処理手続きによって実現されている．その若干の規則を示すと，次のようになる．

同一資源分類パターン中の授受過少資源の存在
→TP 授受誤りの可能性 r1

同一 PR の OP 間での資源授受が複数回
→資源の誤まり使用の可能性 r2

同一 PR の同一関数で生成された複数 PR
→同一クラス WORKER-PR r3

Worker 中の消費者 PR の存在
→開放されない資源の存在可能性 r4

同一クラスの半数以上の WORKER から出発する資源
授受の特定 OP 対→WORKER 特徴 OP 対 r5

特徴 OP 対間で TP 授受のない WORKER
→特徴 OP 使用誤りの可能性 r6

経験則解析は次のように進行する．上記の規則を，順次，プロセス主体解析，タプル主体解析の結果を蓄えたデータベース，および推論によって獲得された知識のデータベース (FDB) に適用し，規則の左辺に該当する対象を発見するとき，右辺の状態を，事実として，

([関連 PRID], [関連資源パターン],
[関連 OPID], [使用規則番号列], 状態)

の形で，FDB に記録する．使用規則番号列は，この状態に到達するまでに使用されたすべての規則の番号である．FDB は最初は空である．

規則を適用する際に，どのデータベースを索引するかは，それぞれの規則の処理手続きの責任とされる．

推論はデータベース FDB に変化がなくなるまで進行するから，経験則解析は，時として，かなり時間を必要とする作業となる．

例で示す．まず r1 の規則では，同一資源分類パターン (str "chopstick", int !) に対して，(str "chopstick", int 0), (str "chopstick", int 1), … の資源量変化を検索し (表 3)，そのすべての資源の授受回数の平均を求め，授受

回数がその半分以下の資源があれば，それは「TP 授受の誤りがある」ものと想定し，FDB に記録する．資源 (str "chopstick", int 5) がこれに該当し，**表4** の行 i_1 の記録が得られる．

次に r2 の規則では，プロセス主体解析の結果を検索し，それぞれ PR : 561 の OP : 2, 4, PR : 562 の OP : 9, 6, PR : 565 の OP : 8, 7 の間で複数回 PT の授受があるから，表4の i_2 i_3 i_4 の記録を得る．

r3では，プロセス主体解析の結果（PR 間親子関係）を検索し，PR : 561 で関数 phil から生成される PR 群に関して，表4の行 i_5 の記録を作る．

r4では，FDB に記録された WORKER-PR について，タプル主体解析の結果から，WORKER PR : 566 が消費者であることを発見し，表4の行 i_6 の記録を作る．

さらに，r5 からは，FDB に登録されている WORKER PR : 562, …, 566 について，プロセス主体解析の結果を検索し，それぞれ OP : 8, 6, OP : 9, 7, OP : 10, 5 の対が WORKER から出発する OP 対として，WORKER の数5の半数を越えるので，これら WORKER を特徴付けるものとして表4の行 i_7, i_8, i_9 の記録を作る．

最後に r6 では FDB およびプロセス主体解析の結果から，PR : 562 の OP : 9, PR : 565 の OP : 8, PR : 566 の OP : 8, 9, 10 が特徴 OP 対の出発側であるにかかわらず，特徴 OP 対の対応する受け取り側を持たないので，行 i_{10}, i_{11}, i_{12} の記録を作る．

既述のように，この解析は FDB が変化しなくなるまで繰返し実行され，終了すると，FDB の内容を出力し，DBM はいったん停止する．

(4) 状態分布解析

経験則解析によって，誤りに関係する PR や資源を予想できた．次には，これらのあってはならない組み合わせの発生時点とその持続時間を調べる．この解析から，使用者は誤りの PR とその状態を特定できる．

状態分布解析の入力は，経験則解析の結果に基づいて使用者が作る空間パターンの遷移記録である．

当面必要な範囲の空間パターン構成の規則を，プログラム言語の構文の記述に使用する BNF の形で示す．

〈空間パターン〉::="["〈パターン式〉"]"
〈パターン式〉::=
　〈パターン項〉{"**or**"〈パターン項〉}
〈パターン項〉::=
　〈パターン因子〉{"**and**"〈パターン因子〉}
〈パターン因子〉::=
　〈TP パターン〉|"**not**" 〈TP パターン〉
〈TP パターン〉::=
　"PT" 〈資源パターン〉|"IT" 〈資源パターン〉
　|"RT" 〈資源パターン〉|"LT" 〈LT パターン〉
〈LT パターン〉::=
　〈資源パターン〉|〈PR パターン〉

上式で，括孤 { } は0回以上繰り返しである．PR パターンについては，後に説明する．

〈空間パターン〉は，それに適合する TP の集合が空間に発見されるときに真または偽となる論理式を表す．すなわち **not** を含まない〈パターン因子〉は，〈TP パターン〉に適合する TP が空間に存在するとき真，存在しなければ偽となる．**not** を含む場合はその逆である．〈パターン項〉は，**and** の左の項部分が真である場合に，右の〈パターン因子〉が真ならばそこまでの項の値を真とする．**and** の左が偽ならば，右の〈パターン因子〉の評価はしないで全体を偽とする．〈パターン式〉は，**or** の左の式部分が真か，右の〈パターン項〉が真である場合にそこまでの式の値を真とする．

それぞれの資源パターンには，経験則解析で発見された問題となる資源パターンを当てはめる．PR パターンは，問題となる PR を生成した LT から作るパターンである．例では，経験則解析の結果，PR : 566 がおかしな振舞をしていることがわかった（表4，i_1, i_6, i_{12}）．表4では，PR : 561, PR : 562, PR : 565 も自己の OP 間で複数の TP の授受がある（i_2, i_3, i_4）．しかし，PR : 561 は主プログラムの PR で，哲学者 PR の生成，消滅を行うのであるから，これは当然である．また，PR : 562, PR : 565 は i_{10}, i_{11}, i_{12} を合わせて見ると症状が軽い．そこで，まずは，PR : 566 を考える．PR : 566 を生成する OP は out ("done", phil(5, 5)) である（プログラムおよび i_5）．そこで，この場合には，PR パターンを ("done", int pr (566)) とする．

さらに経験則解析からは，誤りに関係する資源パターンとして，次のものが発見された（表4，i_1, i_3, i_4, i_6）．

("chopstick", int 5), ("chopstick", int 1),
("chopstick", int 0), ("ticket", int ?).

そこで，プログラムの意味を考えながら，誤り原因に見合うと想定される空間パターンを，これらの PR パターンや資源パターンより，次のように合成する．

PR：566 が箸5を待ち続ける空間は異常 (i_1)：
[LT("done", int pr(566)) **and** IT ("chopstick", int 5)] p1
一つの PR 内の OP 間での資源の授受は資源の重複に起因するかも知れない (i_3, i_4)：
[PT ("chopstick", int 1) **and** PT ("chopstick", int 1) **or** PT ("chopsitck", int 2) **and** PT ("chopsitck", int 2) **or**…] p2
PR：566が消費者なのは，他の切符があるに拘わらず，特定の切符を待っているためでないか (i_6)：
[IT ("ticket", int ?) **and** PT ("ticket", int ?)] p3
……

適切な空間パターンを作成したら，それを入力として DBM を起動する．DBM は，この入力を用いて空間履歴を検索し，空間パターンを評価する．その結果，真となった場合に，すなわちこれらの空間パターンに一致する部分空間が遷移記録上に存在する場合に，DBM は，その時間帯を次の形で出力し，停止する．

(一致始まり時刻，一致終わり時刻)，……

各空間パターンを個別に適用し，状態分布解析を繰返した結果，p1に対して (32, 146) なる結果が求められた．すなわち，時刻 32 の OP に誤りがあり，それが最終時刻146まで続く．p1以外はすべて偽となった．

この解析で，プログラムには PR：566 が資源 (str "chopstick", int 5) を待ち続ける状態に落ち込む誤りがあることが発見された．箸5は生成されなかったと推定される．実際に [**not** PT ("chopstick", int 5)] を空間パターンとして空間履歴を検索すれば，(1, 146) なる結果を得る．そこで箸5を生成するはずの主プログラムを見れば，eval (str "done", phil (i+1, loops)) から哲学者プロセス PR：566 は生成されるが，out ("chopstick", i) からは箸5は生成されない．これらの OP を含むループの視察からは phil (i+1, loops) ではなく，phil (i, loops) とすべきである．

このようにして，誤りの PR を追い詰めることができる．この単純な例題では，この段階で誤りを認識できよう．しかし，一般にはそう簡単ではなく，PR の実行の再現が必要になる．しかし，誤りを含む PR の特定が，プロセスの再現への必然的な道程である．

5. あとがき

Linda-モデルの上に展開したシステムによる，並列プログラムの誤り処理の方法を示した．この方法の重点は，プログラム全体の動きの包括的な情報から，誤りに関係するプロセスを特定することである．この過程で，タプル主体解析の段階の軽い介入を除いては，経験則解析に至るまでは，ほとんど使用者の介入を必要としない．ただし，経験則に関しては，なお使用経験に基づいた改良，追加が必要であろう．タプル空間パターンの作成は経験を必要とする．この作成のためには経験則による自動的な作成の手法を展開することが考えられる．

我々のシステムは SONY NWS841 の上に Prolog および C を用いて実現された．Prolog は推論部分に使用した．大きさは，原始プログラムの形で，C言語処理系への追加，約 20 KB を含めて，約 153 KB である．

本文の例題に対するシステムの実行時間は，経験則解析と状態分布解析にそれぞれ10数秒を必要とする．これは言語の問題もあるが，処理法本来の性質にもよる．ほかの部分は秒以下で問題はないと推定される．

もう一つの問題は，プログラムの実行段階でかなりの量のデータの集積を行うことである．しかし，今までの方法が極めて弱体であるから，本方法の提案は現段階では有意義なものと考える．データを適当に抽出して，量を軽減することが，次の課題となろう．

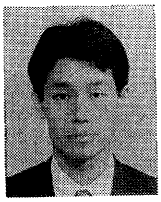
謝辞 本システムの作成過程において，武田正之講師（東京理科大学工学部情報科学科）の適切なる技術的助言を受けた．ここに謹んで感謝する．

参考文献

1) Ben-Ari, M.: *Principles of Concurrent Programmings*, Prentice-Hall (1982).
2) Carriero, N. and Gelernter, D.: How to Write Parallel Programs. A Guide to the Perplexed, *Comput. Surv.*, Vol. 21, No. 3, pp. 323-358 (1989).
3) McDowell, C. E. and Helmbold, D. P.: Debugging Concurrent Programs, *Comput. Surv.*, Vol. 21, No. 4, pp. 593-622 (1989).
4) Gelernter, D. and Bernstein, A.: Distributed Communication via Global Buffer, *Proc. ACM Symposium on Principles of Distributed Computing*, pp. 10-18 (1982).
5) Matsuoka, S. and Kawai, S.: Using Tuple Space Communication in Distributed Object

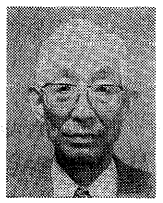
Oriented Language, *Proc. Object Oriented Programming, System, Languages and Application* (OOPSLA'88), pp. 276-284 (1988).
6) Leler, W.: Linda meets UNIX, *Computer*, Vol. 23, No. 2, pp. 43-54 (1990).
7) Roman, Gruia-C. and Cunningham, H. C.: Mixed Programming Metaphors in a Shared Dataspace Model of Concurrency, *IEEE Trans. Softw. Eng.*, Vol. 16, No. 12, pp. 1361-1373 (1990).
8) 吉田紀彦，崎楢修二：協調処理モデル Cellula の分散処理系，情報処理学会論文誌，Vol. 32, No. 7, pp. 906-913 (1991).
9) Roman, Gruia-C. and Cunningham, H. C.: A Shared Dataspace Model of Concurrency, *9th International Conference on Distributed Computing Systems*, pp. 270-279 (1989).
10) Ahuja, S. and Carriero, N. and Gelernter, D.: Linda and Friends, *Computer*, Vol. 19, No. 8, pp. 26-34 (1989).
11) Carriero, N. and Gelernter, D.: Linda in Context, *Comm. ACM*, Vol. 32, No. 4, pp. 444-458 (1989).
12) Dijkstra, E. W.: Hierarchical Ordering of Sequential Processes, *Acta Informatica*, Vol. 1, pp. 115-138 (1971).
13) Gelernter, D.: *Multiple Tuple Spaces in Linda*, *PARLE '89*, Springer-Verlag, pp. 20-27 (1989).

(平成4年7月15日受付)
(平成5年6月11日採録)



河本 達哉（正会員）

1965年生．1989年東京理科大学理工学部情報科学科卒業．1991年同大学大学院理工学研究科前期博士課程修了．現在，東京工業大学理工学研究科博士後期課程（情報工学専攻）在学中．言語処理系，並列分散処理方式，分散人工知能，機械学習の研究に従事．人工知能学会会員．



井上 謙蔵（正会員）

1920年生．九州大学理学部卒業．工学博士．東京理科大学理工学部勤務．主として，プログラム言語処理系に関する研究に従事．著書「言語理論入門」（共立出版，監修），「ソフトウェア講座」（昭晃堂，編集）等，電子情報通信学会，ソフトウェア科学会，ACM 各会員．