C 1　非決定性並列プログラムのデバッギング

斎藤　信男（電総研）

1. はじめに

McCarthyの提唱したいわゆるプログラムの理論（Mathematical Theory of Computation，略してMTC）は，正しいプログラムを如何に容易に作るのかという極めて実用的な目標をその主目的としてかかげている．これらの手法は，プログラムの正しさをきちんと証明することは不可能である場合でも，デバッギングのやり方に何らかの影響を与え，効率の良いデバッギング手段を得るために役に立つかもしれない．

さて，MTCで得られた手法を，並列プログラムに適用することを考えてみよう．この場合，並列プログラムを，順次型（sequential）プログラムの集合と考えれば，MTCの手法がそのまま使えそうであるが，並列プログラムということから生じてくる特有の性質も存在するので，一概に片付けるわけにはゆかない．並列プログラムの特徴的な性質は，決定性（determinacy）ということである．並列プログラムでは，その実行過程のある時刻を考えると，複数個の命令を実行することを許されているが，互に独立な関係にあり，その実行順序は非同期的である．このとき，非同期的に実行される命令の実行順序の偶然性によって，最終的又は中間の実行結果が左右されないようなプログラムを，決定性的（determinate）であるといい，偶然性により結果が異なってくるようなプログラムを，非決定性的（non－determinate）であるという．たとえば，図1に於ては，forkとjoinの間で2つの独立な制御の流れが存在するが，Assigumentをどのような順序で実行しても最終結果は一定である．それに対し，図2では，yに対するAssigumentと，wに対するAssigumentとの実行順序によって，最終結果は異ってくる．

並列プログラムのモデルについては，幾つかの仕事が報告されている[(1)]が，それらの主題は，どのような条件が成り立てば決定性的になるのかということである．又，並列プログラムを記述するグラフ・モデル[(2)]では，それで記述できるようなプログラムは全て決定性的になるようにモデルを作っている．しかし，たとえばOSに於ける待ち行列の処理や相互背反問題（mutual exclusion problem）のように，本質的に非決定性的な並列プログラムも具体的にあらわれてくる．したがって，非決定性並列プログラムを議論しておくことは必要なことである．

並列プログラムの正当性の証明，又はデバッギングを考えたとき，それが決定性的であれば，ある一つの実行順序について検証を行えばよい．一方，それが非決定性的であれば，異なった結果を出すと思われる実行順序の全てについて検証を行なわなければならない．この報告では，ある種の非決定性並列プログラムについて，異なる結果を生ずる可能性のある実行順序を全て求めるアルゴリズムを提案し，それを計算機上に実施した結果を述べる．このような実行順序が得られれば，それに沿って，正当性の証明なりデバッグなりが可能となる．

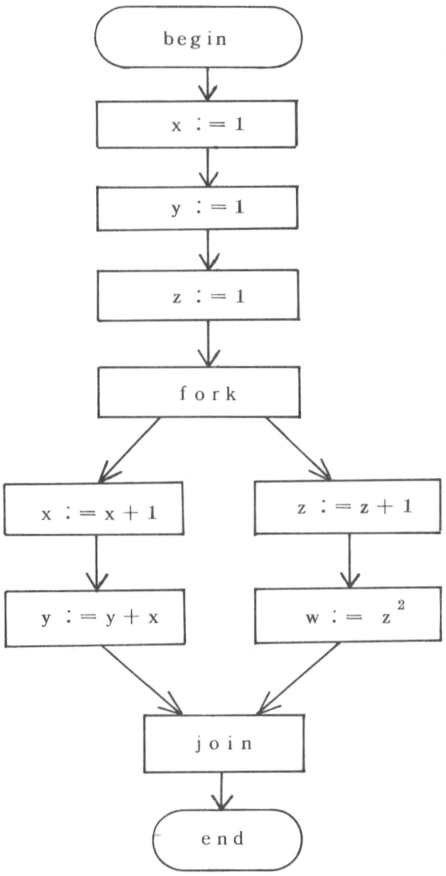

図1.　決定性並列プログラムの例

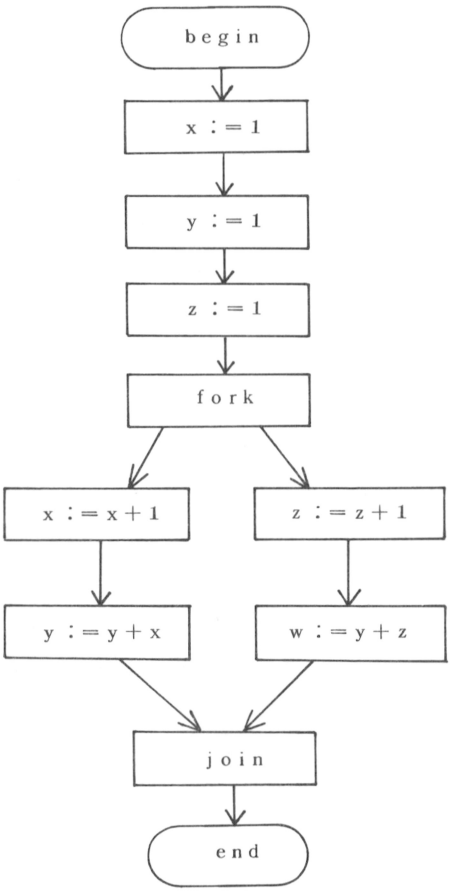

図2.　非決定性並列プログラムの例

2. 非決定性並列プログラムのモデル

ここでは，並列プログラムは独立に実行される順次型プログラムの集合から成り立つと考える．たとえば，図3に示すように，並列プログラムPは，n個の順次型サブ・プログラムP_1，P_2，…，P_nから構成されると仮定する．

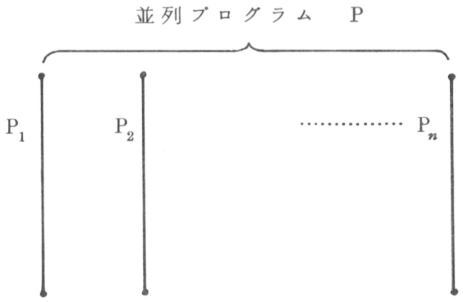

図3.　順次型サブ・プログラムの集合としての並列プログラム

この場合，各順次型サブ・プログラムに於ては，ループおよび分岐点はないものと仮定しておく．

一方，一つの並列プログラムを構成するサブ・プログラム同志は，互に連絡し合って一つの仕事をしている．独立したサブ・プログラム同志の同期をとるためには，いろいろの primitive ，たとえば，wakeup - block とか，P-V operation などが用いられよう．このとき，異ったサブ・プログラムに属する命令に対して，どちらが先に実行されるかによってシステムの状態（たとえば，記憶エリアの値の履歴）が左右されるとき，互に干渉があるという．このような相互干渉をあらわすために，図4に示すように，順次型サブ・プログラムを複数個のプログラム・セクションに分割し，このプログラム・セクションが相互干渉のある命令を含んでいるとき，プログラム・セクション間に相互干渉があると規定する．このような相互干渉をあらわすために，一つの並列プログラムを構成する順次型サブ・プログラムに属する全てのプログラム・セクション間に，相互干渉係数を定義する．すなわち，サブ・プログラム P_i の第 k 番目のセクションを P_{ik} であらわす．又，プログラム・セクション P_{ik} と，プログラム・セクション P_{jl} との相互干渉係数を $e_{i_k j_l}$ とし，相互干渉が存在する場合，$e_{i_k, j_l} = 1$ ，存在しなければ $e_{i_k, j_l} = 0$ とする． 並列プログラムを構成するサブ・プログラム，および各サブ・プログラムを構成するプログラム・セクションを与え，異ったサブ・プログラムに属する全てのプログラム・セクション間の相互干渉係数の値を決めれば，その並列プログラムの動きが規定できることになる．

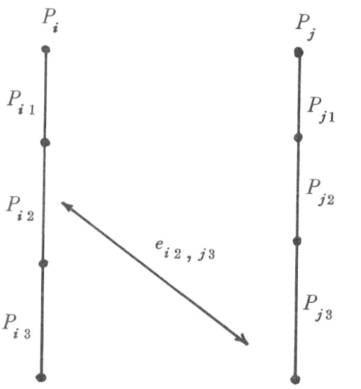

図4. プログラム・セクションとその相互干渉係数

3. 実行過程とその同値類

2.で述べた並列プログラムのモデルに於て，サブ・プログラム i の k 番目のプログラム・セクションを，対 (i, k) であらわすとき，並列プログラム全体は，次の順列の組であらわされる．

$$\begin{matrix} (1,1)\ (1,2)\ \cdots\cdots\cdots\cdots\ (1,n_1) \\ (2,1)\ (2,2)\ \cdots\cdots\cdots\cdots\ (2,n_2) \\ \vdots \qquad\qquad\qquad\qquad\quad \vdots \\ (m,1)\ (m,2)\ \cdots\cdots\cdots\cdots\ (m,n_m) \end{matrix} \qquad (1)$$

ここに，mはサブ・プログラムの数，n_k ($1 \leqq k \leqq m$)は，サブ・プログラム k を構成するプログラム・セクションの数をあらわす．いま，$N = (\sum_{k=1}^{m} n_k)$ とおくと，並列プログラムの一つの実行過程は，N個の対全体の一つの順列を用いてあらわすことができる．この場合，同一のサブ・プログラムに属するプログラム・セクションの間では，決められた順次関係を保たねばならない．したがって，N個の対の順列に於て，次の条件を満すものを，実現可能な実行過程という．

$$\forall_i\ (1 \leqq i \leqq m)\ \forall_j\ (1 \leqq j \leqq n_i - 1)$$
$$\{(i, j) \text{の方が} (i, j+1) \text{より前に存在する}\} \qquad (2)$$

(1)であらわされる並列プログラムに於て，(2)を満すような実現可能な実行過程の数は，n_1 個の同一の物，n_2 個の同一の物，…，n_m 個の同一の物を並べる順列の数と一致するから，

$$C = \frac{N}{n_1!\ n_2! \cdots\cdots n_m!} \qquad \text{但し} \sum_{k=1}^{m} n_k \qquad (3)$$

となる．

次に，2.で述べたプログラム・セクション間の相互干渉係数を考えると，実現可能な実行過程に対し，最終結果が必ず同一になるかどうかを判定することができる．すなわち，プログラム・セクション(i,k)と，プログラム・セクション(j,l)の相互干渉係数$e_{i_k j_l}$が1ならば，実行過程$\sigma_s = \cdots(i,k)\cdots(j,l)\cdots$と，実行過程$\sigma_t = \cdots(j,l)\cdots(i,k)\cdots$とは，異った最終結果を持つ可能性があり，$e_{i_k j_l}$が0ならば，$\sigma_s$と$\sigma_t$とは必ず同一の最終結果を持つ．最終結果が必ず同一になることを，実行過程の同値関係とすれば，実現可能な実行過程を同値類に分類することができる．

そこで，異ったサブ・プログラムに属する全てのプログラム・セクション間に相互干渉係数を与え，実現可能な実行過程を相互干渉係数によって同値類に分けたとき，各同値類の代表元を求めるアルゴリズムを考えてみよう．

定義1

任意の二つの対の組(i,k)，(j,l)に於て，

（1）　$i > j$

又は（2）　$i = j$かつ$k > l$

ならば，(i,k)の方が(j,l)より大きいといい，$(i,k) \succ (j,l)$と書く．

定義2

N個の対の順列σ_iとσ_jとに於て，定義1の順序関係を用いて，lexicographic order をつけることができる．すなわち，

$\sigma_i = a_1 a_2 \cdots\cdots a_N$

$\sigma_j = b_1 b_2 \cdots\cdots b_N$　　（但し，a_i，b_jは一つの対をあらわす）

としたとき，あるk（$1 \leqq k \leqq N$）が存在して，

$a_1 = b_1$，……，$a_{k-1} = b_{k-1}$，　$a_k \succ b_k$

が成り立つならば，σ_iはσ_jよりも大きいといい，

$\sigma_i \succ \sigma_j$　　と書く．

定義3

N個の対の順列σ_iに於て，任意の二つの要素(i,k)と(j,l)とに対し，$(i,k) \succ (j,l)$で，かつ，(i,k)が(j,l)よりも前に存在するとき，(i,k)と(j,l)は逆順の関係にあるという．

実現可能な実行過程に於ては，$(i,k) \succ (j,l)$のとき，定義1で述べた（2）の条件は成り立たないから，定義3を次のように書き換えてもよい．

定義3′

N個の対の順列σ_iが実現可能な実行過程をあらわすとき，任意の二つの要素(i,k)と(j,l)に対して，$i > j$で，かつ，(i,k)が(j,l)よりも前に存在するとき，(i,k)と(j,l)は逆順の関係にあるという．

定義4

N個の対の順列σ_iに於て，任意の隣り同志の二つの要素(i,k)，(j,l)に対して，$e_{i_k j_l} = 0$ならば，(i,k)と(j,l)の位置を互換した順列σ_jは，実行結果が必ずσ_iと同じになる．このような互換による変換を，同値変換と

いう．

アルゴリズム1

与えられたN個の対の順列σ_iが実現可能実行過程のとき，どのような同値変換を施しても，σ_iのlexicographic orderの大きさを減少することが不可能ならば，又，そのときだけ，σ_iを一つの類の代表元とする．

アルゴリズム1を用いると，相互干渉係数によって分けられた同値類に対して，lexicographic orderで考えて最小の要素をその同値類の代表元として求めることになる．アルゴリズム1を更に具体化すると，次のようなアルゴリズム2が得られる．

アルゴリズム2

1. Nコの対を大きさの順番に並べた順列

$$\sigma_0=(1,1)(1,2)\cdots(1,n_1)(2,1)(2,2)\cdots(2,n_2)$$
$$\cdots\cdots\cdots\cdots$$
$$\cdots\cdots\cdots\cdots(m,1)(m,2)\cdots(m,n_m)$$

は代表元である．

2. σ_iに於て，逆順の関係にある二つの要素の組$(i,k),(j,l)$を全て見つけ出す．（但し，$i>j$とする．）

① 全ての逆順の組$(i,k),(j,l)$に対して，$e_{i_k j_l}\neq 0$の場合 $\Rightarrow$ σ_iは代表元である．

② 幾つかの逆順の組$(i,k),(j,l)$に対して，$e_{i_k j_l}=0$の場合

(i) $e_{i_k j_l}=0$となる全ての逆順の組$(i,k),(j,l)$に対し，(j,l)だけを動かす同値変換をどのように施しても，(j,l)を(i,k)よりも前に持って来ることが不可能な場合 $\Rightarrow$ σ_iは代表元である．

(ii) $e_{i_k j_l}=0$となる一つの逆順の組$(i,k),(j,l)$に対し，(j,l)だけを動かす同値変換を何回か施した結果，(j,l)を(i,k)よりも前に持って来ることが可能な場合 $\Rightarrow$ σ_iは代表元ではない．

（アルゴリズム2の説明）

σ_0は，lexicographic orderで考えると，全順列中，最小のものであるから，一つの代表元になる．

又，2－①で述べたように，全ての逆順の組の相互干渉係数が1ならば，それらの組の位置を交換することは不可能であるから，同値変換を用いて，σ_iのlexicographic orderの大きさをそれ以上減少させることは不可能である．したがって，σ_iは同値類のうち最小の要素であり，代表元となる．

2－②－(i)では，相互干渉係数が0であるような逆順の組$(i,k),(j,l)$があっても，(j,l)を動かす同値変換では，(j,l)を(i,k)の前に持って来ることはできないから，σ_iのlexicographic orderの大きさを減少できないことになり，上と同じ理由で，σ_iは代表元となる．2－②－(ii)では，相互干渉係数が0である逆順の組$(i,k),(j,l)$の一つに対して，(j,l)を動かす同値変換を用いて，(j,l)を(i,k)の前方にまで持って来ることができるから，σ_iのlexicographic orderの大きさを減少できることになり，したがって，σ_iは同値類の最小要素ではないから，代表元とはならない．

4. アルゴリズムの実施

次に，3.で述べた同値類の代表元を求めるアルゴリズムを，実際に計算機上に実施した結果を述べる．

プログラムの構成は，与えられた並列プログラムに対して実現可能な実行過程を求めること，および，実現可能な実行過程が同値類の代表元になるかどうか判断することの2つに大きく分けられる．

4.1　実現可能な実行過程を求めるプログラム

並列プログラムを構成する m 個の順次型サブ・プログラムに対するプログラム・セクションは，次のようなドット形式で記述する．

$(1.\ 1)(1.\ 2)$ ……… $(1.\ n_1)$

$(2.\ 1)(2.\ 2)$ ……… $(2.\ n_2)$

⋮　　⋮

$(m.\ 1)(m.\ 2)$ ……… $(m.\ n_m)$

最初に，2つの異ったサブ・プログラムに属するプログラム・セクションの要素による順列を求めることを考えよう．たとえば，$n_1 = 2$，$n_2 = 3$ とし，サブ・プログラム1の要素を◯で，サブ・プログラム2の要素を□であらわす．順列の初期値は，次のものである．

σ_0 =①②[1] [2] [3]

実現可能な実行過程という条件から，同じサブ・グループ内での順序は保存されていなければならない．そこで，先ず，[1] を1つ前へ進め，その状態で残りの [2] を [1] を越えないという条件のもとで前へ進める．[2] を前へ1つ進める毎に，残りの [3] を [2] を越えないという条件で前へ進める．[2] 又は [3] がそれ以上進めなくなったら，それらを動かし始めた状態にまで [2] および [3] を戻し，今度は [1] を前へ1つ進めることを試みる．これらの過程を，図5に示す．

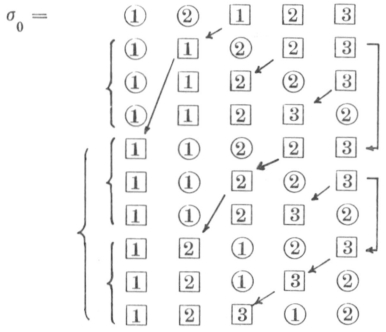

図5.　2つのサブ・プログラムの要素による順列

上述の操作は，次のような再帰的手続き（ recursive procedure ）mixtwo で記述することができる．

```
recursive proc: mixtwo(plist,top,last) ;
  begin
    bottom := last ;
  labl: find boundary in plist from top to bottom ;
    if notfound then return ;
    plist := commute(plist,boundary) ;
    print  plist ;
    mixtwo(plist, boundary + 1, last) ;
    bottom := boundary ;
    go to labl
end ;
```

最初のcall：

mixtwo（（（1. 1）（1. 2）…（1. n_1）（2. 1）（2. 2）…（2. n_2））, 0, n_1+n_2）；

上の手続きで，plictは，ドット形式であらわされるプログラム・セクション要素の順列を格納するリスト，top，lastは，plist内のポインタで，topからlastまでの間の要素に対して2つのグループの混ぜ合せを施すことを示す．boundaryは，plist内のポインタで，順列内の隣り同志の組に於て，前の要素が第1のグループ，後の要素が第2のグループとなっているような場所を示す．そのような隣同志を置換し合うことにより，新しい順列が作られてゆく．関数commuteは，plist内のboundaryで示す要素と，その後隣りとの置換を施すことを行う．その置換を施す毎に新しい順列が作られるので，それをprint命令で書き出す．次に，mixtwoを再帰的に呼び出すが，このときtopポインタはboundaryより1つ後を指定するので，第2のグループの残りの要素について混ぜ合せを行うことを示している．tの再帰的呼び出しから戻ってくると，再びラベルlab 1へジャンプし，新しいboundaryを捜すが，そのときは，bottomは，先に見つかったboundaryの値にしておくので，先に見つかったboundaryよりも前にある新しいboundaryだけを見つけることになる．

3つ以上の異ったサブ・プログラムに属するプログラム・セクションの要素を用いた順列を作るときには，2つのグループの混ぜ合せの手法を拡張して用いる．すなわち，3つのグループが存在するとき，図6に示すように，第1と第2のグループの混ぜ合せの1つの場合に対し，それらを1つのグループと見做し，第3のグループとの混ぜ合せを，2つのグループの混ぜ合せと考えて，先に述べた方法を適用する．

第1のグループと第2のグループの混ぜ合せの1つの場合　　第3のグループ

図6　3つのグループの混ぜ合せ

この操作は，mixtwoを拡張した次のような再帰的手続きmixで記述することができる．

```
recursive proc: mix(plist,next,length,ng,top,last) ;
  begin
    bottom := last ;
    labl:  find boundary in plist from top to bottom ;
    if notfound then return ;
    plist := commute(plist,boundary) ;
    if next = null then
                    begin
                      print  plist ;
                      go to lab2
                    end ;
```

```
    newlist := APPEND(plist,CAR(next)) ;
    p := last + CAR(length) ;
    mix(newlist,CDR(next),CDR(length),ng + 1,0,p) ;
  lab2: mix(plist,next,length,ng,boundary + 1,last) ;
    go to lab1
  end ;
```

上の手続きで，plistは途中で作られるk個のグループの要素の順列を格納するリスト，nextは，残りの$(m-k)$個のグループの要素を示すリスト，ngはplist内に作られる順列を構成するグループ数(すなわちk)，lengthはnextで示される各グループの要素の数を示す．top，lastは，mixtwoに於ける意味と同じである．したがって，最初のcallは，次のようになる．

最初のcall：

$$\text{mix}((),((1.1)(1.2)\cdots(1.n_1)\cdots(m.1)(m.2)\cdots(m.n_m))),$$
$$(n_1, n_2, \cdots, n_m), 0, 0, 0);$$

最初のcallでは，plistにはnullを与え，nextに対して，N個の要素を大きさの順に並べたリストを与える．boundaryは，最初のk個のグループを同一のグループと見做した場合に，mixtwoに於けるように，前の要素が第1のグループ，後の要素が第2のグループになっているような隣り同志の組の存在する場所を示すポインタである．nextリストがnullになったときは，N個の要素を全て使った順列が出来たことになるので，その場合のみ，print命令で出力をする．これは，最初の$(m-1)$個のグループの中間結果の順列と，m番目のグループとの混ぜ合せ行うときに対応する．この場合，ラベルlab2にジャンプして，boundaryの後からlastまでの要素に対して混ぜ合せ操作を続ければ良い．nextリストがnullでないときは，中間結果の順列が入っているplistの後に，nextリストの最初のグループをつけ足し，lastのポインタもそのグループの長さだけ増加してからmixを再帰的に呼び出す．このとき，next，lengthは，それぞれCDR(next)，CDR(length)を指定する．又，ngは，1つ増加させておく．

4.2 同値類の代表元かどうかを判定するプログラム

前節のmixのプログラム内で，N個の対からなる順列plistが得られ，printをする命令があるが，そのplistに対し，アルゴリズム2を適用して，その順列が同値類の代表元となるかどうかを判定する．そのような判定を行う操作は，次のような手続きnewclassにより記述される．

```
boolean proc: newclass(plist,n) ;
  begin
    for i = 1 step 1 until (n - 1) do
      for j = (i + 1) step 1 until  n      do
        if plist(i)>plist(j)
          then
            if  e(plist(i),plist(j)) = 0
              then
                begin
                  if j =(i + 1)then return(F) ;
                  if movable(plist,i,j) = T then return(F)
                end ;
```

```
    return(T)
  end ;

boolean proc: movable(plist,i,j) ;
  begin
    for k =(j - 1)step -1 until(i + 1)do
      if e(plist(j),plist(k)) ≠ 0 then return(F) ;
    return(T)
  end ;
```

newclassの結果がTrueならば，代表元であり，Falseならば代表ではないことを示す．plistは，判定の対象となる順列を格納してあるリスト，nはその要素の数を示す．又，forループを2つ用いて，逆順となっている要素同志の全ての組を見つけ出している．このforループを抜け出したということは，アルゴリズム2の2－①又は2－②－(i)に相当するから，Trueを結果の値としてreturnする．2－②－(ii)に相当するのは，movableというboolean procedureの値がTrueの場合であるが，これは，同値変換によりlexicographic orderで考えた大きさが減少できることを示している．したがって，この場合，newclassの結果としてFalseを持ってreturnする．なお，このとき，隣り同志の要素が逆順で，かつ，その要素間の相互干渉係数が0ならば，あきらかに，lexicographic orderの大きさを同値変換により減少できるから，Falseをnewclassの結果として，returnする．このケースは，判定が容易であり，かつ，頻繁にあらわれるから，なるべく多くの場所でこの判定を行うことが望ましい．

以上のプログラムは，LISPの操作を含んだALGOL形言語で記述したが，実際には，LISP 1.5 (ELISP)[3]を用いて実施してある．

5. おわりに

非決定性並列プログラムの1つのモデルに対し，最終結果が異なる可能性のある実行過程を求めるアルゴリズムを与え，それをLISPを用いて実施した．こうして得られた実行過程の全てに対し，正当性の証明，あるいは，デバッギングを行えば，全体の並列プログラムの正しさが保証される．このモデルは，非常に単純なものであるが，同じ手法は，図7に示すような2方向分岐点だけを持つ並列プログラムに対してもあてはめることが可能である．

ループを含んでいる場合には，簡単に拡張することは出来ない．これは，将来の大きな問題であろう．

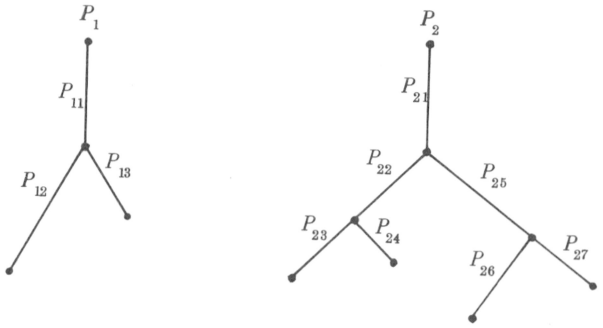

図7　2方向分岐点をもった並列プログラム

参 考 文 献

(1) たとえば

Karp , R.M. and Miller , R.E. Parallel Program Schemata , JCSS , Vol. 3, May 1969.

(2) たとえば

Rodriguez , J.E. A Graph Model for Parallel Computation , MAC-TR-64 , MIT, Sep. 1969.

(3) E-LISP インタプリタ プログラム解説書, 電子技術総合研究所, 日本ソフト , March 1972.

本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html