プログラミング－言語・基礎・実践－　5－9
(1991. 11. 21)

並列プログラムの非決定的動作テスト方式

橋本 辰範　　堀田 博文

NTT ソフトウェア研究所

あらまし　並列プログラムのもつ動作の非決定性およびその再現性の欠落は，システムの品質向上を阻害し，その開発，運用に大きな問題をもたらす．この問題解決の方策として，プログラムの動作解析により並列プログラムの非決定的動作を検出し，代替動作がある場合にその実行を行なうテスト方式を提案する．本テスト方式は，事前の実行の近傍に存在する非決定的動作を効率良く検出しそれをテストするものであり，実行履歴保存，非決定的動作検出，再実行制御機構の３つの要素技術で実現する．本方式によると，プログラムの非決定性に基づく動作が網羅的に検査できる．

Testing Non-determinant Behaviors of Concurrent Programs

Tatsunori HASHIMOTO　　Hirofumi HOTTA

NTT Software Laboratories

Abstract　A concurrent program, in general, has non-determinacy in its behavior and the detection or replay of such behavior is hard to achieve. Difficulty in quality assurance of concurrent programs stems from this kind of non-determinacy. A method to detect non-determinacy in a concurrent program and execute alternative potential patterns, if any, is proposed in this paper. This method, in technical view, consists of storing execution history, analyzing non-determinacy in the history, and executing arbitrary pattern. Proposed method enables the tester to test all the potential execution patterns for a set of inputs.

1 はじめに

並列プログラムのもつ動作の非決定性, 及び動作の再現性の欠落は, 並列プログラムを用いたシステムのテスト, デバグを困難にし, そのシステムの品質向上を阻害している. 本稿では, この問題解決の方策として, 次の特徴を持つ並列プログラムテスト方式を提案する.

- 並列プログラムの実行結果に基づく動作解析によりそのプログラムが持つ動作の非決定性を検出.
- その結果を用いて, 非決定性に起因して起こり得る別の動作を実行.

2 並列プログラムのテスト

2.1 関連研究

並列プログラムは, 一般に, 同一の入力に対しても毎回同一の動作をするとは限らない. これを並列プログラムの動作の非決定性と呼ぶ. この非決定的動作およびその再現性の欠落は, 並列プログラムを用いたシステムの開発, 運用に大きな問題を引き起こしている.

問題の所在は大きく次の2つに分類できる.

(1) 非決定的動作が起こりうる箇所の正確な識別.

(2) デバグや再演テストのために必要な動作の再現.

(2)の問題に対する解決策として,[Leb87] や [Tai87] は, 並列プログラムの実行の再現を可能とする方式を提案している. これらは, 並列プログラムの非決定的動作がプロセス間通信や共通変数アクセスに関するプログラム記述がもつ非決定性に依存しているとしている. その上で, これら通信や変数アクセスの実行を忠実に再演する独自の機構を提案するものである. 両文献とも実行履歴の保存と, 個々のメールボックスや共通変数に対するアクセス順の制御が要素技術となっている. しかし, これらの方式はバグが発見された場合のデバグ作業を支援する技術にすぎず, システムの品質向上に積極的に貢献するものではない.

一方,(1)は, 非決定的動作の存在そのものがプログラム設計者の意図に反する場合, あるいは (2)と同様の機構と組み合わせた積極的な全動作パターンのテストを行う場合に有用である. しかし, その解決策は未確立である. すなわち, システムの品質を保証するためには十分なテストが必要であるが, 並列プログラムの非決定的動作に関するテスト技術はいまだ確立されていない.

本稿では (1)の方式を中心に, 並列プログラムのテスト方式を提案する.

2.2 テストの現状

一般にプログラムの出力は, 初期状態と外部入力, および内部状態遷移により決定される. 逐次型プログラムの場合、その内部状態遷移は初期状態と外部入力が決まればプログラムの内容により完全に決定される. ところが, 並列プログラムの場合, 内部状態遷移にプログラム自身では制御できないいくつかの非決定性が関わってくる可能性があるため, 初期状態と外部入力だけで出力を一意に決定することはできない.

しかし, 実際の並列プログラムのテストではこの点が十分考慮されておらず, 初期状態と外部入力と期待する出力の3つ組だけでテストが実施されているのが現状である. このため, テストにより保証されたはずの条件(3つ組)に対しても再度走行させると予測していない動作が起きてしまう可能性がある. このテスト条件の不十分さは並列プログラムのデバグをますます困難にしている.

3 並列プログラムのモデルと非決定的動作

本稿では, 並列プログラムとして, 相互に情報をやりとりし協調動作を行なう複数の並列プロセスから構成されるプログラムを考える.

一般的に並列プログラムの非決定的動作は以下の点で発生する.

(1) 同期通信

プロセスのスケジューリング, 外部入力のタイミング, 通信遅延などを原因とする各プロセス進行のタイミングの非決定性により生じるメッセージ送信と受信の組合せの相違による.

(2) 共有オブジェクトへのアクセス

(1)と同様のプロセス進行のタイミングの非決定性により生じる共通データ等へのアクセスタイミングのずれによる.

(3) 割込み

プログラムの進行とは独立に発生するタイマ割込みや外部割込みによる.

(4) OS 呼出し

時刻など, 静的に決定できない値を利用することによる.

この中で (4) については, OS 呼出しをシミュレートするルーチンを与えることにより決定的な動作が確認できる ([Tai87]). また, (3) の割込みについては本稿では扱わないこととする. よって, 以下の考察は割込みの存在しない並列プログラムに限定する.

実際の並列プログラムでは, プロセス相互の情報交換として (1) と (2) の方法が共存するものが多い. しかし, これらの方法は相互に他方をモデル化できることが知られている. また, 通信方式にも多彩な方式があり, ハードウェアの構成によって数種の通信方式がプログラム内に混在することもある. つまり, 通信方式はネットワーク上でのプロセス配置に依存するが, ネットワーク透過モデルを仮定することでこれを統一に扱うことができる. そこで本稿では, ネットワーク透過で, プロセス間の情報交換に通信だけを用いるモデルを採用する. すなわち, 上記 (1)のみの非決定的動作を仮定することになる.

さらに通信方式についてモデル化を進める. 一般的に, 通信方式は, 送信と受信の対応, 送信方式, 受信方式という観点で分類できる. 送信と受信の対応については, 一つの送信に対し通信可能な受信が必ず一つしか存在しない一対一, 一つの送信に対し通信可能な受信が複数存在しえる一対多, 逆に一つの受信に対し通信可能な送信が複数存在しえる多対一, さらに, 一対多でかつ多対一である多対多の 4 種類が考えられる. また、送信方式, 受信方式についてはそれぞれ閉塞, 非閉塞がある. ここで, 閉塞とは通信が確立するまでそれ以降の実行を待ち合わせる方式である.

これらの組合せの中から現存する代表的な通信方式に適合するモデルを挙げると, 以下の 3 つに絞られる (表 1) .

- 受信閉塞, 送信閉塞, 一対一
- 受信閉塞, 送信閉塞, 多対一
- 受信閉塞, 送信非閉塞, 多対多

表 1: 代表的通信モデル

通信モデル			例
送信対受信	送信方式	受信方式	
一対一	閉塞	閉塞	SAL*
多対一			Ada
多対多	非閉塞		CHILL

* [Ich86] より参照.

これらのうち, 上の 2 つは 3 番目のモデルを用いて表現することができる. よって本稿では, 送信非閉塞, 多対多の通信モデルに基づいた並列プログラムのモデルを定義し, それを基に議論を進める.

本稿で扱う並列プログラムのモデルを以下にまとめる.

動作 複数のプロセスによるメッセージ通信を用いた協調動作. 割込みはなし.

通信 多対多, 送信側非閉塞, 受信側閉塞. プロセス間通信の記述方法は以下の通り.

- 送信

  形式 send Sig [to Proc]

  意味 シグナル Sig をプロセス Proc に送出し走行し続ける. ただし,to Proc がない場合は任意のプロセスが Sig を受信できる.

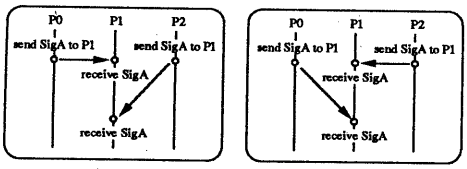
- 受信

  形式 receive Sig [Sig [Sig ···]]

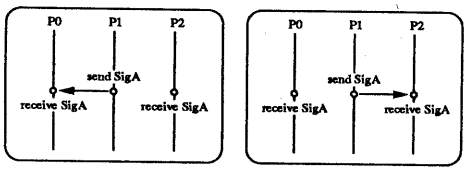
  意味 例えば receive Sig1 Sig2 Sig3 という記述はシグナル Sig1,Sig2,Sig3 のいずれかを受け取る. 指定したシグナルがいずれも未到着の場合, いずれかのシグナルが到着するのを待合わせる. 複数の受信可能シグナルが到着済みであるとき, どのシグナルが受けとられるかは不定とする.

このモデルでは, 通信処理に関して次の 2 種類の非決定性が存在する (図 1).

- 1 つの receive に対し複数の send(複数のプロセスも可) が候補として存在.
- 1 つの send に対し複数の receive(複数のプロセスも可) が候補として存在.



(a) 一つのreceiveに対し複数のsendが存在



(b) 一つのsendに対し複数のreceiveが存在

図 1: 非決定的動作の例

4 非決定的動作テスト

4.1 テスト方針

並列プログラムのもつ動作の非決定性が原因で起こる不定の動作についてその正当性を実行テストにより保証するには, 起こりうる全動作を意図的に実行してその結果を確認しなければならない. そのためには, まず, 対象とする並列プログラムが非決定的動作を起こす可能性があるかどうかを明らかにする必要がある. さらに, その並列プログラムにどのような非決定的動作があるかを検出し, その条件を非決定的動作に関するテストケースととらえ, またそのうちのどのケースをテスト実行させるかを決定しなければならない.

ここで, 非決定的動作を検出し, 非決定的動作に関するテストケースを抽出する方法には, ソースプログラムを静的に解析する方法と, プログラムを実行させてその振舞いを解析する方法がある.

並列プログラムを構成する各プロセスの動作は, それぞれを記述しているプログラムの制御構造に依存しており, 初期状態および実行時に決定される入力データ (外部入力及び他プロセスからのメッセージ) に基づいて決定される. つまり, 並列プログラムの動作を静的に解析するためには, 各プロセスの制御構造上の全パスを考慮する必要がある. しかし, パスを有限個に限定できるとは限らないうえに, 他プロセスとのメッセージ送受パターンの可能性の解析まで必要であることから, 並列プログラムの静的解析によるテストケース抽出は現実的ではない. これは, 現実の逐次型プログラムの構造テストにおけるテストケース作成の尺度として C2(全パス網羅) がほとんど用いられていないことからもうかがえる. たとえプロセスの制御構造が単純ですべてのパスが抽出できたとしても, それぞれのパスの組合せにおいて, 再びプロセス間の関係を考慮した動作可能性を判定する必要があり, 問題をさらに複雑化している. さらに, 静的解析では「かもしれない」レベルのものの扱いが常に問題となり, 最終判断には人手の介在を必要とするという問題を内包している.

これらの困難さは, 並列プログラムの静的解析に基づくテストケース作成の可能性を否定するものではないが, 本稿では, より現実的でかつ正確な解を与えるものとして, プログラム実行結果に基づく動作解析をもとに並列プログラムの非決定的動作に関するテストケースを抽出する方式を提案する.

本稿で提案する方式は, ある入力に対してプログラムを実行したときの履歴を基に, 同一の入力で非決定性を原因として起こりうる全実行パターンを過不足なく生成できるものである. 一度の実行に対しては, その実行で起こった各受信がどの送信文からのメッセージを受け取り得たかの情報を本方式では生成する. さらに, 実行されなかったが受け取り得た送信メッセージを実際に受け取らせるように制御しながらプログラムを実行し, 同様の非決定性解析, 制御しながらの実行を繰り返す.

4.2 テスト手順

実際のプログラム動作の結果を解析することにより非決定的動作の有無とテストケース, すなわち代替動作を起こす条件を検出し, その条件を制御情報とするテスト機構を用いて検出された代替動作をさらにテスト実行する方式を採る. 1 種の入力に対して非決定性動作関連テストを実施する手順を (1) ～ (3) に示す (図 2).

(1) テスト実行と履歴保存

テスト対象である並列プログラムにある入力データを与え実行する. このとき, プロセス毎に通信履歴 (実行された送信命令と受信命令を実行順に並べたもの) をシグナル名と共に記憶する.

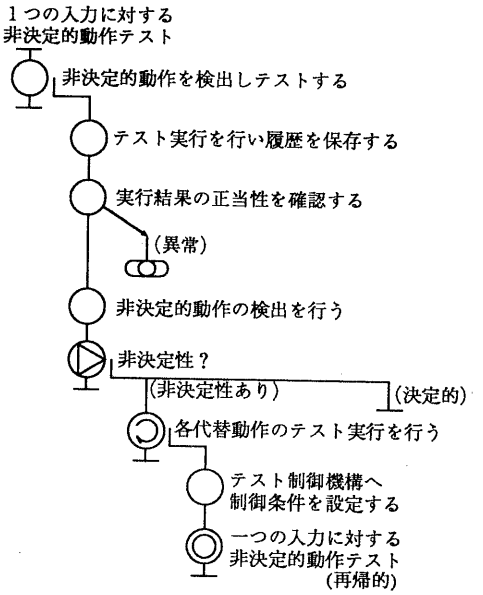


図 2: テスト手順

(2) 非決定的動作検出

各プロセスの通信履歴から組合せ可能なシグナル送受信の組を求める. 一対一に決定できない送受信の組が 1 組でも存在する場合, その実行は非決定的動作を含んでいたと判定する.

(3) テスト制御機構への制御条件設定と再実行

再演テスト法 [Leb87, Tai87, Has91] で確立されているシグナル送受信の組合せを制限する機構を用い, 通信履歴とは異なる送受信の組合せでプログラムを実行する. このときの送受信の組合せがその動作の制御条件となる. 再実行における通信履歴が以前のものと同じである保証はないため, 再実行においても通信履歴を記憶しておき同様の解析を実施する. このように再帰的に再実行を繰り返すことで種々の非決定的動作の正当性を確認する.

4.3 履歴保存の方式

履歴保存は, 基本的にはこれまでに使用されているいずれかの方法をもちいる. 本稿では履歴保存の実現方式について特に規定しないが, ただ一つの制約を明らかにしておく.

それは, 履歴保存がプログラム動作の論理に影響を与えてはならないということである. この制約は, テストの正当性を保証するための条件である. たとえば, プロセス間通信を実現する実行時ルーチンに履歴保存機構を取り込んでテストし, その機構を取り除かずにそのまま運用することで論理を保証するという方法がある.

本稿では, 履歴を保存しながら実行する場合も, 実用での実行と同じタイミングで通信が行われると仮定する. この仮定により, 実用でも実行される可能性の高い部分がテストされることになる.

次に通信に関する動作を解析するために保存すべき情報について示す。もちろん,

- 非決定性の所在の判断
- 他選択枝用制御条件の作成

を可能とするに十分な並列プログラムの通信に関する情報を履歴として保存する必要がある. この情報とは, 具体的には,

- 送信要求の内容
- 受信要求の内容
- 各プロセスにおけるそれらの実行順序
- 受信実行で受信した信号を特定する情報

である. ここで, 本稿の通信モデルでは, 受信実行時に到着済みの信号が複数ある場合, 到着順には関係なくそれらすべての信号が受信可能である. そのため, 同一プロセスから送出された同じシグナル名をもつ複数の信号を特定するためにプロセス毎の送信信号の番号付けが必要となる.

つまり, 本稿の並列プログラムモデルでは, 実行された send または receive 命令の実行順に沿った並びと, receive で受信した信号はどのプロセスが何番目に送出したものであるかという情報が履歴を形成する. 履歴の例は 5.1 に示す.

4.4 非決定的動作検出方式

実行により保存された履歴情報をもとに, その実行における非決定的動作の有無と代替動作の条件を検出する方式を以下の step.1 ～ step.3 に示す.

step.1 通信履歴から依存関係に基づくサイクルなし有向グラフGを生成する．Gは以下に示すものである．

- ノード集合 $N = S \cup R$, ただし,
  $s \in S$：各プロセスにおける送信の実行,
  $r \in R$：各プロセスにおける受信の実行（受け取り）．
- エッジ集合 $E = E_o \cup E_m$, ただし,
  E_o：1プロセス内での送信実行 $s(\in S)$, 受信実行 $r \in R$ を実行順につなぐもの,
  E_m：実際の信号送受（$s(\in S)$ で送出した信号を $r(\in R)$ で受信した場合 s から r にエッジがある）．

step.2 G上の全ノードに対し到達可能解析を実施する．

step.3 step.2の結果を用い，受信 $j(j \in R)$ が受け取りえた送信集合 PS_j を求める．

- $PS_j = (S - (CS_j \cup RS_j)) \cap MS_j$, ただし,
  CS_j：エッジ $(i \rightarrow j) \in E_o, i \in N$ なる i に到達可能な送信集合,
  RS_j：j から到達可能な送信集合,
  MS_j：受信 j とプロセス名，シグナル名が適合する送信集合．

ここで、グラフGは，ノード p からノード q へのパスが存在する任意のノード q に対して，ノード集合

$P_q = \{p|$ ノード p からノード q へのパスが存在する $\}$

とするとき，すべてのノード $p \in P_q$ が実行されないとノード q は実行できないという依存関係を意味する．到達可能解析はすでに確立している方法を用いる．

また，CS_j は受信jを実行するためにすでに通信が確立されていなければならない送信のノード集合である．RS_j は，受信jの実行により履歴通りの通信が確立しないかぎりその実行が保証されない送信のノード集合である．よって CSj, RSj とも受信jの受信対象とはなりえないノード集合である．

5 解析例

3つのプロセスP,Q,Rで構成される並列プログラムに対する非決定的動作テストの実施例を示す．

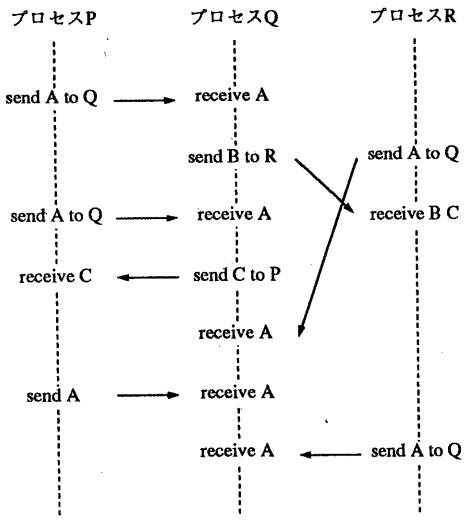


図 3: 通信シーケンス図

5.1 通信履歴の例

ある実行において次の通信履歴が記録されたとする．

- プロセスPの通信履歴
  {S(A,Q),S(A,Q),R((C),Q,2),S(A,any)}
- プロセスQの通信履歴
  {R((A),P,1),S(B,R),R((A),P,2),S(C,P),
  R((A),R,1),R((A),P,3),R((A),R,2)}
- プロセスRの通信履歴
  {S(A,Q),R((B,C),Q,1),S(A,Q)}

ここで,S(Sig,Proc) は送信 send Sig to Proc の実行を，R((Sig1,Sig2,...),Proc,Num) は受信 receive Sig1 Sig2 ... の実行とそのとき実際に受け取った信号の送出元プロセス Proc とそのプロセス内での送信通番 Num を示す．ここで示した履歴は図3に示す通信シーケンスを表したものである．

5.2 非決定的動作検出の例

この実行について非決定的動作が存在したかどうかを検出するために，その実行の各受信において組合せ可能であった送受信の組を求める．

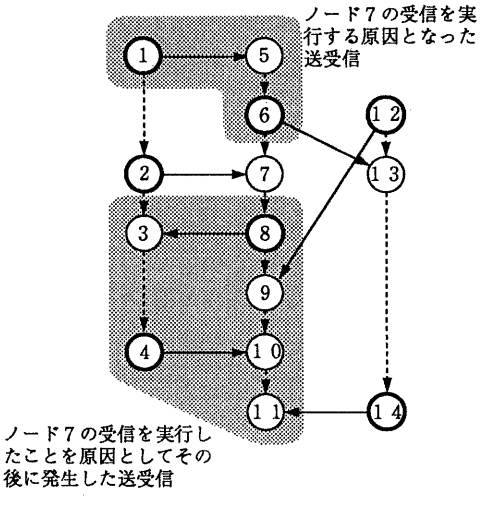


図 4: 依存グラフ上での解析

step.1 プロセス P,Q,R の通信履歴をもとに依存グラフを作成する (図 4). この依存グラフは 3 に示した通信シーケンス図に対応しているため, 説明は省略する. 依存グラフを作成する際, 各ノードにノード番号を付与する. ここでは, プロセス P の通信履歴の最初の要素から順に, そして P の要素すべてに番号付けしたら次に続けて Q の通信履歴要素へ, さらに R の通信履歴要素へと番号を付与した.

step.2 送受信の組合せにおける制約となる全ノード間の依存関係を明らかにするために, 全ノードに対する到達可能解析を実施する. 依存グラフをもとにノード i からノード j への到達可能性を示す可到達行列 A_{ij} を求めると

$$A_{ij} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

となる. ここで,0 は非到達,1 は到達を示す. たとえば $A_{2,8} = 1$ はノード 2 からノード 8 へ到達可能であることを示す.

step.3 到達可能解析結果を用いてそれぞれの受信が受け取りえた送信集合を求める. まず, 送信ノード集合 S は,

$$S = \{1, 2, 4, 6, 8, 12, 14\}$$

である. ここで, 例としてノード 7 の受信 $R((A), P, 2))$ について解析すると, CS_7 , RS_7 はそれぞれ

$$\begin{aligned}
CS_7 &= \{k | A_{ki} = 1, (i \to 7) \in E_o, i \in N, k \in S\} \\
&= \{k | A_{k6} = 1, k \in S\} \\
&= \{1, 6\} \\
RS_7 &= \{k | A_{7k} = 1, k \in S\} \\
&= \{4, 8\}
\end{aligned}$$

となる.CS_7 はノード 7 の受信が実際に起こるために必要であった送受信関係の送信側を示している. また,RS_7 は, ノード 7 の受信が起こったために起こり得た送受信関係の送信側を示している. さらに履歴情報より, シグナル A を送信したノードの集合は,

$$MS_7 = \{1, 2, 4, 12, 14\}$$

が明かとなる. よってノード7の受信が受け取り得た送信の集合は,

$$\begin{aligned} PS_7 &= (S - (CS_7 \cup RS_7)) \cap MS_7 \\ &= (\{1,2,4,6,8,12,14\} - (\{1,6\} \cup \{4,8\})) \\ &\quad \cap \{1,2,4,12,14\} \\ &= \{2,12,14\} \end{aligned}$$

となる.

この解析結果は, このプログラム実行において, ノード7の receive がノード2,12,14 の send のいずれかを受信することができたという非決定性をもっていたということを示している. 各受信ノードがそれぞれただ一つの送信ノードとしか組み合わせられないときに, その実行が決定的に動作したことが保証できる (それは, 実際に起こった送受関係を示す).

5.3 再実行

複数の送信ノードが組合せ可能であったとき, 再演テスト法で提案されている通信順序制御機構を用いて, 実際に行われた送受の組合せ以外の通信を強要する再実行を実施する. 上述の例では, ノード7の receive が,12,14(2はすでに実行済み) の send を受け取ったときの動作を通信順序制御機構を用いてテストとして実現する. ここで, その動作が仕様に適合しているかどうかをチェックする. 動作が正しいことが確認できれば, さらにこのときの通信履歴を保存しておき, そこから別の非決定的動作が検出されるかどうかを調べる.

6 おわりに

本稿では, 通信履歴に基づく並列プログラムの非決定的動作テスト方式を提案した. 本方式は, 起こりえる非決定的動作の全可能性の検出を目的とするものではなく, 通信による状態遷移に着目し, 動作の近傍に存在する非決定的動作を効率良く検出するものである. この方式には以下2つの利点がある.

- プログラムの非決定性に基づく誤動作を検出できる.
- テスト実施により保証された動作の条件を明確化できる.

今後は, 本方式を実際のシステム開発に適用しながら, 検出された代替動作の選択基準について検討する.

参考文献

[Has91] 橋本, 生沼, 堀田, "分散マルチプロセスのテストにおけるイベント系列制御方式", 情報処理学会第42回全国大会, 1991.

[Ich86] H.Ichikawa, M.Itoh and M.Shibasaki, "Protocol Oriented Service Specification and its Transformation into SDL", Trans. IECE Japan, vol. E69, No. 4, Apr. 1986.

[Leb87] T.J.Leblanc and J.M.Mellor-Crummey, "Debugging Parallel Programs with Instant Replay", IEEE Trans. on Computers, vol. C-36, No. 4, Apr. 1987.

[Tai87] K.Tai and S.Ahuja, "Reproducible Testing of Communication Software", Proc. Compsac87, pp.331-337.