1．並行処理プログラムの試験

Testing of Concurrent Programs by Zengo FURUKAWA (Educational Center for Information Processing, Kyushu University), Eisuke ITOH (Computer Center, Kyushu University) and Tetsuro KATAYAMA (Graduate School of Information Science, Nara Institute of Science and Technology).

古川　善吾[1]　伊東　栄典[2]　片山　徹郎[3]

1 九州大学情報処理教育センター
2 九州大学大型計算機センター
3 奈良先端科学技術大学院大学情報科学研究科

1. はじめに

並行処理プログラムは，逐次処理プログラムに比較すると処理の高速化や記述の簡潔さが期待できる．すなわち，順次行う処理を同時に行うことが可能であるので，全体としての処理時間を短縮できる可能性がある．また，プログラムの記述において実行の条件を並行に行われる処理ごとに記述するので記述すべき条件が少なくなる．一方，並行に動作するプログラム単位を適切に制御するための機構が必要である．制御が適切でないと逐次処理プログラムよりも処理が遅くなる可能性がある．さらに，個々のプログラム単位は並行に動作するほかのプログラム単位との間での相互作用があり，その相互作用が適切でなければならない．相互作用が適切でないと，プログラムそのものが動作しなくなったり，計算結果が矛盾するという誤りを生じる可能性がある．並行処理プログラムの記述方法および誤りが発生した時の故障の発見（デバッグ）方法については，これまで多くの議論がなされている[4), 13), 17)]．

並行処理プログラムがどの程度信頼できるか，また，系統的に誤りを発見するためにどのように試験を行えばよいかという並行処理プログラムの試験方法については，現在議論が行われている．たとえば，ソフトウェア工学に関する国際会議ICSE[8)]やAPSEC[3)]がある．また，ACMやIEEEのソフトウェア工学に関するジャーナル[1), 2), 9)]などで議論が行われている．本稿では，並行処理プログラムの試験方法について解説する．とくに，筆者らが進めている並行処理プログラムの試験方法に焦点をあててその特徴を述べる．これによって，並行処理プログラムの試験方法に関する研究の現状およびその困難さの原因を理解していただければ幸いである．まず，第**2**章では，並行処理プログラムの特性を明らかにする．すなわち，並行処理プログラムの難しさの原因を明らかにし，並行処理プログラムで発生する可能性のある誤りについて議論する．第**3**章では，並行処理プログラムの動作のモデル化とそれに対応した試験方法について紹介する．第**4**章では，並行処理プログラムのモデル化として事象相互作用グラフを用いた時の試験方法および信頼性，課題について解説する．第**5**章では，並行処理プログラムの試験を支援するために必要なツールの機能をまとめる．第**6**章では，まとめと今後の課題について述べる．

2. 並行処理プログラムの特性

並行処理プログラムは，逐次的に実行される複数のプログラム単位が通信や同期などの相互作用を行いながら仕事を進めるプログラムである．プログラム単位は，プログラミング言語やプログラムの種類によってプロセスやタスクといろいろな呼び方がなされている．ここでは，プログラム単位あるいは，プロセスと呼ぶことにする．

並行処理プログラムには，非決定性がある．非決定性とは，同一の入力データに対して出力が異なる可能性があることである．逐次処理プログラムでは，同じ入力データに対して出力データは基本的に一定であるけれども，並行処理プログラムではプロセスの実行のタイミングや実行環境，順序付けアルゴリズムによって出力データが異なる可能性が高い．

並行処理プログラムの誤りとしては，(a) 計算誤り，(b) 通信誤り，(c) 同期誤り，がある[4)]．計算誤りは，逐次処理プログラムでも発生する誤りであり，Howdenが路解析テスト法の分析で詳細に分類している[7)]．この計算誤りは，逐次的に実行されるプログラム単位内で発生する誤りである．通信誤りは，プログラムの2つのプロセス間でのデータの受渡しにともなう誤りである．逐次処理プログラムでは，手続きや関数の呼出し時に発生するし，並行処理プログラムでは，プロセス間の通信において発生する．同期誤りは，プ

ロセスの実行順序が不適切であることによって発生する誤りであり，並行処理プログラムに特有の誤りである．さらにこの同期誤りは，(a) 安全性の破壊誤り，(b) 生存性の破壊誤り，(c) 公平性の破壊誤り，に分けることができる．

安全性の破壊誤りは，並行処理プログラムのプロセスの間で利用される資源の操作の相互排除の失敗によって，データの一貫性が失われる誤りである．たとえば，ファイルの内容を更新する場合を考える．2つのプロセスで値を読み込んでそれぞれのプロセスで1加えてから書き込むと値は2増える必要がある．しかしながら，同時に読み込んでしまうと後に書き込んだ値だけが有効になるので1しか増えない．生存性の破壊誤りは，プロセス間での同期の失敗によって，デッドロックと呼ばれるプログラムが有意義な動作をしない誤りである．たとえば，よく知られた問題として「哲学者の食事問題」がある．すなわち，哲学者は2本のフォークを使って食事をしなければならないとする．丸いテーブルに座った複数の哲学者が間に置かれたフォークを1本ずつ取ってしまうと，誰も食事ができなくなる．公平性の破壊誤りは，ライブロックと呼ばれあるプロセスだけが実行されない誤りである．たとえば，「Reader-Writer問題」がある．すなわち，安全性の破壊誤りを起こさないために，Writerは誰も読み出していない時だけ書き込むことができるが，ReaderはWriterが書き込みをしていなければいつでも読み出すことができるとする．このとき，Readerが次々に読み出しを行うとWriterはいつまでも書き込むことができない．

このような特性をもつ並行処理プログラムの試験が困難な理由は以下のとおりである．

(1) プログラムの状態数の増大

逐次的に実行されるプロセスが互いに影響を及ぼし合うので並行処理プログラム全体の状態をプロセスの状態の組合せによって定義することができる．しかしながら，並行処理プログラムの状態数は，プロセスの数やプロセスの状態数に対応して組合せ論的に増大する．このことは，試験において実現すべき状態の数が増大することを意味するので，試験すべきデータ集合の生成がより困難になる．

(2) プログラムの正しさあるいは誤りの定義が困難

項番 (1) に述べたようにプログラムの状態数が膨大であるために，正しい状態と誤り状態とを明示的に定義することが困難であることが多い．このことは，試験結果が正しいものであるか否かの判定が困難であることを意味するし，特定の誤りを発見する試験データの作成が困難になる．

3. 並行処理プログラムのモデル化と試験

プログラムにはソースプログラムとして記述された静的な側面と，計算機で実行した時の動的な側面とがある．並行処理プログラムでは，この両者の間の差が大きいので，動的な側面を形式的に表現することが重要であり，これをモデル化と呼ぶ．これらのモデルを用いたモデル化と試験方法として以下のものがある．

並行状態グラフ　Taylorらは並行処理プログラムの解析方法として並行状態グラフを提案している[15]．並行処理プログラムを構成する個々のプロセスの状態の組合せを並行状態と定義する．並行状態は，プロセスの状態を変化させる操作によって遷移する．並行状態グラフは，並行状態を節点，遷移を枝とするグラフである．Taylorらは，並行状態グラフを用いて並行処理プログラムをモデル化し，試験が十分に行われたかどうかの評価のために並行状態グラフの節点や枝，あるいはそれらの列を測定対象とする試験基準を提案している．しかしながら，並行状態グラフには，(a) 並行処理プログラムの中のプロセスの数が固定されていなければならない，(b) 並行状態の数が組合せ論的に増大する，という欠点がある．

事象相互作用グラフ　片山らは，プロセスの制御の流れに基づく事象グラフ，プロセス間の通信や同期を表す相互作用とで並行処理プログラムをモデル化した[11]．片山らは，この事象相互作用グラフの上で並行処理プログラムの試験項目として協調路（Copath）を定義し，自動的に協調路を作成するTcGenを試作している．一方，伊東らは，事象相互作用グラフの上での長さkの並行事象の列（「順序列（Ordered sequences)」と呼ぶ）を測定対象とする試験の十分性評価技法について分析している[10]．

事象列　並行処理プログラムを構成するプロセスの実行は，逐次的である．そのために，プロセス内での事象の発生は，全順序関係がある．並行処理プログラム全体では，プロセス間での同期で制限された半順序である．この半順序をインタリーブモデルの上での並行処理プログラムの実行を考えることによって全順序化したものが事象列である．K. C. Taiは，この事象列を用いて並行処理プログラムの誤りを定義すると同時に，プログラムの実行例から抽出した事象列に基づいてプログラムを実行するための，再実行方式を提案した[14]．

以下，事象相互作用モデルに基づいた並行処理プログラムの試験十分性評価と試験項目作成法について詳細に述べる．

4. 事象相互作用グラフに基づく試験

4.1 事象相互作用グラフ

並行処理プログラムを逐次的に動作するプログラム単位が並行に動作するプログラムであるとして，その動作を事象相互作用グラフで表す．事象相互作用グラフは，事象グラフの集合と相互作用の集合でプログラムを表す．まず，プログラム単位内の制御の流れを制御フローグラフとして表す．その制御フローグラフでプログラム単位間での相互作用を行う文を並行事象文（プロセスの生成，同期や通信を行う文，共有資源を操作する文）とする．並行事象文および並行事象文に関係のある文（並行事象文を含む判定文）以外の文を削除して事象グラフを構築する．相互作用は，2つの事象グラフの中の並行事象文と相互作用のラベルで表す．たとえば，プロセスを生成する文とプロセスの開始文とは，プロセスの名前をラベルとする相互作用である．また，Adaのランデブーであるエントリ呼び出し文とエントリ受け付け文とは，エントリ名をラベルとする相互作用である．

4.2 順序列試験基準

事象相互作用グラフの並行事象文の集合（並行事象文集合CE）において並行事象文の長さkの順序列集合OSC_kを次のように定義する．

$$OSC_k = \{<e_1, e_2, \ldots, e_k> \mid k \geq 1, e \in CE, 1 \leq i \leq k\},$$

この長さkの順序列集合に対して被覆率（実行された順序列の個数とOSC_kの要素数との割合）を100％にすることを要求する試験基準を順序列試験基準OSC_kと呼ぶ．任意の長さの順序列試験基準をOSCと表すことにする．

kが1の時，順序列集合OSC_1は，並行事象文の集合と等しくなり，順序列試験基準OSC_1はすべての並行事象文を少なくとも1回試験において実行することを要求する．この試験基準は，逐次処理プログラムにおけるC_1試験基準（文被覆試験基準）に包含される．すなわち，C_1試験基準が満たされているならば，OSC_1試験基準も満たされている．

kが2の時，順序列集合OSC_2は，並行事象文の対の集合である．これを順序対集合と呼び，試験基準を順序対試験基準と呼ぶことにする．この試験基準は，逐次処理プログラムの試験基準C_2（分岐被覆試験基準）には包含されていないので，kが2以上の順序列試験基準OSC_kが並行処理プログラムに特有の試験基準である．Adaのランデブーに着目した試験基準「ランデブー通路テスト基準」[5]や「共有変数のデータフローテスト基準」[6]は，並行事象文をそれぞれ，エントリ呼び出し文とエントリ受け付け文，および，共有変数の操作文，に限定したOSC_2である．

4.2.1 順序列試験基準の信頼性

試験基準が満たされた時に，プログラムに存在するすべての誤りが発見できる時，試験基準は信頼できるという[7]．しかしながら，任意のプログラムについて信頼できる試験基準は，全数試験だけである．すなわち，プログラムのすべての入力データ（並行処理プログラムではさらに，すべての実行順序）で試験を行った時，すべての誤りを発見できるし，誤りが発見されなければ，プログラムが正しい．しかしながら，実用的な時間で全数試験を行うことはほとんど不可能である．そこで，プログラムに含まれる可能性のある誤りを限定した上で，その誤りの発見可能性に基づいて試験基準の信頼性を議論する．すなわち，試験基準が満たされた時に，プログラムに含まれる特定の種類のすべての誤りを発見することを保証できるならば，試験基準は，その誤りの種類に対して信頼できるという．また，入力データがある値をとった時のみ誤りを発見できるならば部分的に信頼できるという．

順序列試験基準の並行処理プログラムの誤りに対する信頼性は，以下のとおりである[6), 10)]．

(1) 通信誤り

順序対試験基準は，完全通信誤りについては信頼できる．また，部分通信誤りに対しては部分的に信頼できる．

(2) 安全性の破壊誤り

同期誤りの中の安全性の破壊誤りに対しては，操作において相互排除を必要とする共有資源の資源数（資源を同時に操作できるプロセスの数）をrとしたとき，長さr+2の順序列試験基準OSC_{r+2}が信頼できる．

(3) 生存性の破壊誤り[10]

同期誤りの中の生存性の破壊誤り（デッドロック）に対しては，実行時のプロセスの数がpであるとき，長さp+1の順序列試験基準OSC_{p+1}が部分的に信頼できる．

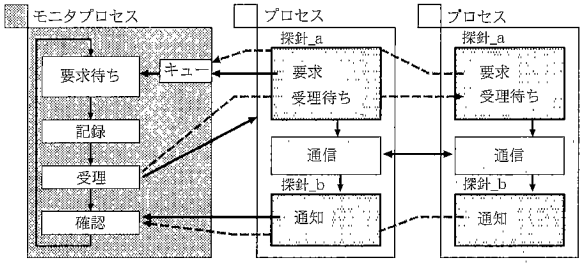


図-1 被覆率測定のための実行制御プロセス

4.2.2 順序列試験基準の被覆率計測

順序列試験基準に基づく被覆率の計測には，ツールが必要である．並行処理プログラムの動作を計測するには，計測ツールの影響を最小にする必要がある．そのほかにも，動的に生成されたプロセスの識別や計算機システム間での時間の一意性の保証が必要になる．順序対試験基準に基づく被覆率計測ツール[10]では，プログラムのソースコードに計測結果を累積することにより動的に生成されたプロセスの識別の必要性をなくした．また，時間の一意性については，1つの実行制御プロセスによって保証している．

順序対被覆率計測ツールの中心となる実行制御プロセス（モニタプロセス）を**図-1**に示す．被計測プロセスの通信文の前と後ろに探針_aと探針_bを挿入する．実行時には，実行要求を受け取って実行を記録した後に，実行受理を出す．さらに，通信の終了の通知を受け取って実行を確認する．これによって，実行時の並行性を逐次化してしまう．しかしながら，試験された実行系列は元の並行処理プログラムで実行可能な実行系列であるので，誤りが発見された場合には，元の並行処理プログラムでも誤りが発生する．

この被覆率計測ツールを計算機ネットワーク上のテキストベースの会話ツールphoneプログラムに適用した結果を**表-1**に示す[10]．phoneプログラムは，3つのプロセスから構成されており，並行事象文集合は44個の文を含んでいるので，順序対集合は，1,936（$=44^2$）個の要素をもっている．この中で，実行可能な順序対は，1,849（95.5%）である．ただし，実際に，被覆率計測プログラムで計測できた順序対は，666（34.4%）である．

試験において実行されなかった順序対についても表-1に示している．実行されない並行事象文に起因するものが87である．それ以外については，現在の被覆率計測プログラムの機能が不足しているために計測ができない順序対と実行環境に依存する順序対がある．これらについては，解決が可能であり，並行処理プログラムの試験で順序列試験基準を適用できる．

この順序列試験基準を適用するにあたって最も問題となる課題は，順序列の数の増大である．先に示したphoneプログラムはそれほど大規模なプログラムではない．それでも，このプログラムには44の並行事象文があり，実行すべき順序対は1,936（$=44^2$）である．安全性破壊誤りの有無を確認するためには，共有資源数が1であれば，85,184（$=44^3$）の順序組を実行する必要がある．さらに，生存性破壊誤りの有無を確認するためには，3つのプロセスがあり，3.7×10^6（$=44^4$）の順序対を実行する必要がある．これだけの試験を実行することは現実には困難である．この順序列数の増大への対策には2つの考え方ができる．

表-1 phoneプログラムの試験における順序対の被覆率

順序対の総数	1,936
実行された順序対数	666
機能不足のために実行されない順序対数	620
試験環境を必要とする順序対数	563
実行可能な順序対数（=666+620+563）	1,849
実行不能な順序対数	87

(1) 逐次処理においてもC_1試験基準は限定された有効性しかもたないが試験の進捗を管理する現実的な指標として用いられている．同様に，並行処理プログラムでも最も容易な指標として順序対試験基準OSC_2を用いる．

(2) 並行処理プログラムとしての試験は困難であるとして，実行時に生存性の破壊や安全性破壊誤りの発生を監視し，発生した時点で対策を立てる．これは，実行時検査であり，プログラムの試験方法ではないのでこれ以上は議論しない．

4.3 試験項目作成

並行処理プログラムは，非決定性があるために試験を実行するためには入力データだけではなく，実行のタイミングを規定しなければならない．実行のタイミングの規定方法としては，順序列を用いる方法が考えられる．片山らは，並行処理プログラムを事象相互作用グラフとしてモデル化し，そのグラフの上でプログラムの試験データを規定する条件として試験項目を協調路（Cooperating Path: Copath）として以下のように定義している[11]．

A，Bを事象グラフとし，α，βをそれぞれA，B上の路とする時（$\alpha\in$Path（A），$\beta\in$Path（B）），相互作用InterActionsの任意の要素<a, b, X>に対して，αとβ内のaとbの個数が同じである路の対を協調路と呼ぶ．

$$\mathrm{Copath}(A,B) = \{<\alpha,\beta> \mid \alpha\in\mathrm{Path}(A)\wedge\beta\in\mathrm{Path}(B)\wedge \mathrm{Suc}(<\alpha,\beta>,\mathrm{InterActions})\},$$

$$\mathrm{Suc}(<\alpha,\beta>,\mathrm{InterActions}) = \forall<a,b,X> [<a,b,X>\in\mathrm{InterActions}\rightarrow [\mathrm{Num}(\alpha,a)=\mathrm{Num}(\beta,b)]],$$

ただし，Num（α，a）は列αの中に含まれるaの要素数である．

プロセスが2つ以上存在する場合，すなわち事象グラフが2つ以上の場合の協調路は，任意の2つの事象グラフの間で協調路となる路の組である．すなわち，並行処理プログラムが，m個のプロセスからなる場合，

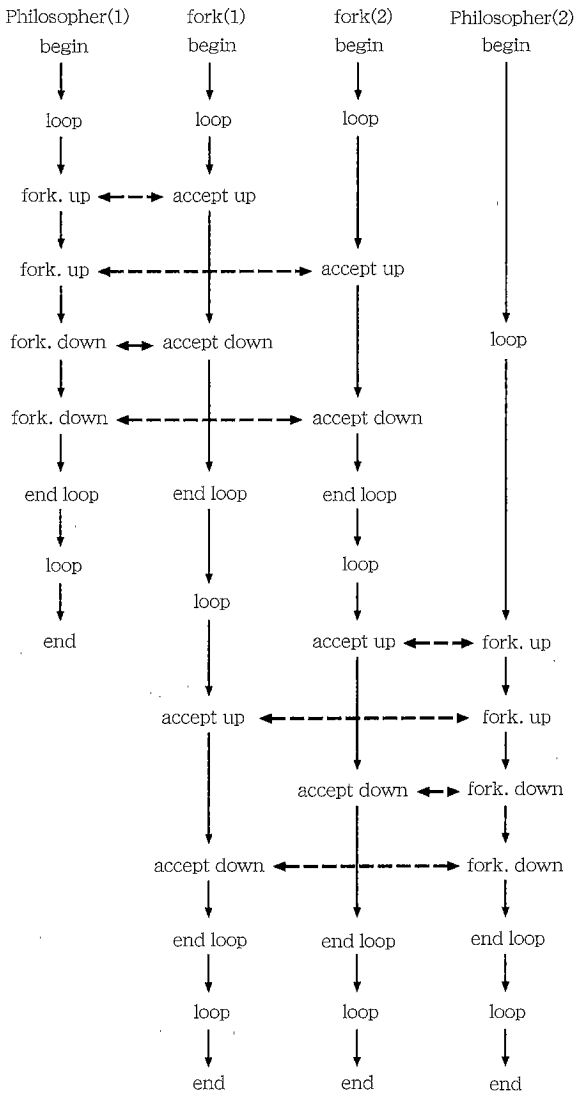


図-2　哲学者の食事の協調路

協調路は，m個の路の組となる．

$$\begin{aligned}\mathrm{Copaths}\ (\mathrm{EGs}) = \{ &< \alpha_1, \alpha_2, \ldots, \alpha_{|\mathrm{EGs}|} > \\ &| \ \forall i,j\ [[1 \le i,j \le |EGs|, i \ne j] \wedge \\ &< \alpha_i, \alpha_j > \in \mathrm{Copath}\ (\mathrm{EG}_i, \mathrm{EG}_j) \wedge \\ &\mathrm{EG}_i, \mathrm{EG}_j \in \mathrm{EGs}]\} \ .\end{aligned}$$

哲学者の食事問題の協調路の例を図-2に示す[11]．図では，哲学者が2人の時である．

この協調路の作成において，終了判定には，先に述べた順序対試験基準を用いることができる．片山らはAda並行処理プログラムに対してこの協調路を作成するシステムTcGenを試作してその有効性を確認している[11]．ここで作成した協調路を試験項目として試験を実施するには，協調路を実現するためのプログラムの実際の入力データ（試験データ）の作成，および，相互作用の実現が必要である．そのためには，以下の課題を解決する必要がある．

実行可能性の確保　プログラムのソースコードに基づいて協調路を作成するのでその実行可能性の保証が必要になる．プログラムの制御フローグラフから任意の実行路を作成した場合，実行可能である保証はない．これを自動的に解決することは困難である．これを解決するためには，プロセス内の逐次処理を実行するための入力データの作成において連立不等式の解決あるいは人手による解決が必要になる．

協調路の実現　被試験プログラムを協調路のとおりに実行するためには，プログラムを強制的に実行する必要がある．この強制実行の方法については，Tai[14]，菰田ら[12]が検討している．また，並行処理プログラムでの入力データおよび実行タイミングの実現のためのシミュレータの開発も必要である．シミュレータについては，本特集「3．制御用プログラムの試験」の解説[16]で紹介されている．

5．試験実施環境

試験の課題の1つである効率化を実現する手法に，プログラムを実行して試験を支援するために試験環境がある．これまで試験だけのための実行環境というより，デバッグ支援と合わせた環境として実現されていることが多い．並行処理プログラムの試験支援環境としてもデバッグ機能[13]との連携での実現が考えられる．並行処理プログラムの試験支援環境として必要な機能および現状は以下のとおりである．

(1) プログラムの実行

被試験プログラムを実行し，実行経過を記録するために，以下のような機能が必要である．

(a) 実行条件の設定

被試験プログラムを実行するために変数の値や実行のタイミングを設定する．並行処理プログラムでは，変数の値だけで実行状態が決まらない．並行処理プログラムの非決定性を排除して試験する条件を明示するために実行のタイミングまで決定する強制実行が必要になる．

(b) 実行状況の計測

実行状況を計測するために，被試験プログラムの実行に影響を与える．とくに並行処理プログラムでは，実行時間への影響がプログラムの動作そのものを変更する可能性がある．これに対処するために，たとえば，実行順序における半順序関係の保存などが必要になる．また，実行状況の計測および記録は，並行処理プログラムの1つのプロセスとして実現するので，このプロセスによって障害を導入しないことが必要である．

(2) 実行結果の判定

並行処理プログラムでは，非決定性のために実行結果が一意に定まらない可能性がある．そのために，実行結果の判定を期待出力との単純な比較によっては実現できない．結果の判定そのものは人手に頼る必要があるため，その判断を支援する機能が必要である．一方，生存性の破壊誤り（デッドロック）は，試験環境で発見し，被試験プログラムの実行そのものを中断する必要がある．

(3) 試験の評価

プログラムの試験の進捗状況を調べ，プログラムの正当性あるいは試験の十分性を評価する．たとえば，第**4**章で述べた順序列試験基準に基づいた計測によって得られる被覆率で試験十分性を評価する．

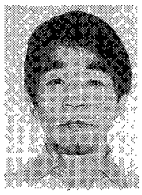
6. おわりに

並行処理は，第**1**章で述べたように，ある意味で古くて新しい問題である．その試験についてもいろいろな議論がなされている．それらの議論の中のいくつかについて紹介してきた．とくに，事象相互作用グラフに基づいたテスト技法を詳細に紹介し，並行処理プログラムの試験における問題点を明らかに，その解決策の一例を示した．

参考文献

1) Transaction on Software Engineering and Methodology, ACM.
2) Software Engineering Notes, ACM.
3) Proc. of 1996 Asis-Pasific Software Engineering Conference, IEEE-CS (1996).
4) Ben-Ari, M.: Principles of Concurrent Programming, Prentice Hall International, Inc. (1982).
5) 古川善吾, 牛島和夫: ランデブー通路を用いたAda並行処理プログラムのテスト充分性評価, 電子情報通信学会論文誌, Vol.J75-DI, No.5, pp.288-296 (1992).
6) 古川善吾, 有村耕治, 牛島和夫: 並行処理プログラムにおける共有変数のデータフローテスト基準, 情報処理学会論文誌, Vol.33, No.11, pp.1394-1401 (Nov. 1992).
7) Howden, W. E.: Reliability of the Path Analysis Testing Strategy, IEEE Trans. Softw. Eng., Vol.2, No.3, pp.208-215 (1976).
8) Proc. of 19th International Conference on Software Engineering, ACM (1997).
9) Transacetion on Software Engineeging, IEEE.
10) Itoh, E., Furukawa, Z. and Ushijima, K.: A Prototype of a Concurrent Behavior Monitoring Tool for Testing of Concurrent Programs, Proc. APSEC '96, pp.345-354 (1996).
11) 片山徹郎, 弧田敏行, 古川善吾, 牛島和夫: 並行処理プログラムにおけるテストケースの定義と生成ツールの試作, 情報処理学会論文誌, Vol.34, No.11, pp.2223-2232 (Nov. 1993).
12) 弧田敏行, 片山徹郎, 古川善吾, 牛島和夫: Ada並行処理プログラムのテストケース作成とその強制実行に関する一考察, 第20回JAPAN SIGAda予稿集, pp.9-15 (1993).
13) McDowell, C. E., Helmbold, D. E.: Debugging Concurrent Programs, ACM Computing Surveys, Vol.21, No.4 (1989).
14) Tai, K. C.: On Testing Concurrent Programs, Proc. Compsac '85, pp.310-317 (1985).
15) Taylor, R. N., Levine, D. L. and Kelly, C. D.: Structural Testing of Concurrent Programs, IEEE Trans. Softw. Eng., Vol.18, No.3, pp.206-215 (1992).
16) 内平直志, 川田秀司: 制御用プログラムの試験, 情報処理, Vol.39, No.1, pp.19-25 (Jan. 1998).
17) Venkatesan, S. and Dahan, B.: Testing and Debugging Distributerd Programs Using Global Predicates, IEEE Trans. Softw. Eng., Vol.21, No.2, pp.163-177 (1995).

(平成9年10月2日受付)



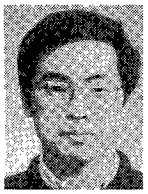
古川　善吾（正会員）

1952年生．1977年九州大学大学院工学研究科情報工学専攻修了．同年日立製作所システム開発研究所入社．1986年九州大学工学部勤務．1992年同大学情報処理教育センター助教授，現在に至る．工学博士．ソフトウェア工学，とくにソフトウェアテスト法，日本語文書出力方式に興味をもつ．電子情報通信学会，ソフトウェア科学会，ACM，IEEE-CS各会員．

e-mail:zengo@ec.kyushu-u.ac.jp

伊東　栄典（正会員）

1969年生．1992年九州大学工学部情報工学科卒業．1994年同大学院情報工学専攻修士課程修了．1997年同大学院博士後期課程修了．同年九州大学大型計算機センター助手，現在に至る．工学博士．ソフトウェア工学，とくにソフトウェアテスト法に興味をもつ．e-mail:itou@cc.kyushu-u.ac.jp



片山　徹郎（正会員）

1969年生．1991年九州大学工学部情報工学科卒業．1993年同大学院情報工学専攻修士課程修了．1996年同大学院博士後期課程修了．同年奈良先端科学技術大学院大学情報科学研究科助手，現在に至る．工学博士．ソフトウェア工学，とくにソフトウェアテスト法，システムソフトウェアに興味をもつ．電子情報通信学会，ソフトウェア科学会各会員．

e-mail:kat@is.aist-nara.ac.jp