論　文

分散システムにおける全域状態監視アルゴリズム

正員　森保　健治[†]　　正員　曽根岡昭直[†]　　正員　真鍋　義文[†]

Global States Monitoring Algorithm for Distributed System

Kenji MORIYASU [†], Terunao SONEOKA [†] *and* Yoshifumi MANABE [†], *Members*

あらまし　複数プロセスからなる分散システムにおける同期誤りには，デッドロックのように一度陥ったら以後継続的に成立する誤りだけでなく，それぞれのプロセスは正常に動作しているにもかかわらず，システム全体としては要求された動作から外れるような誤りも存在する．このような誤りは，各プロセスの状態をローカルに監視していたのでは検出できないことが多く，システム全体の状態（全域状態）を監視することが有効である．システムの動作を監視するための全域状態として安定全域状態が知られている．安定全域状態とは各プロセス間のチャネルが空である全域状態で，メッセージの送受信の区切りとなる状態である．本論文では，システム動作中における安定全域状態の継続的な検出法について論ずる．検出に必要な情報はシステムの通常のメッセージに付加（piggyback）して送るという前提条件下で，各プロセスで記憶すべき情報量の下界を示し，最小の情報量で安定全域状態を検出する分散アルゴリズムを与える．更に，提案するアルゴリズムの計算量がオーダ的に最適であることを示す．

1. まえがき

近年，通信システムの高度化が急速に進み，多様な通信サービスの実現が可能となってきた．それと共に，システム上の通信ソフトウェアはより規模が大きく複雑になる傾向にある．サービスの多様化はサービスの短命化につながり，対応するソフトウェア開発期間の短縮が要求されるようになる．また，ソフトウェアの大規模化，複雑化に従い，高品質なソフトウェアの開発は困難になり，実行過程で初めて発見されるような誤りが存在しやすくなる．従って，通信システム（分散システム）は低品質な通信ソフトウェアに対して耐力のあるシステムであることが望まれる．そのためには，ソフトウェアに誤りが生じても正常に処理を継続できる技術（フォールトトレランス技術）を含有したネットワーク基盤の確立が重要である[6]．

フォールトトレランス技術は，誤り検出技術と誤り回復技術の二つの要素技術からなる．従来の研究は，ローカルに誤りを検出できると仮定した上での誤り回復技術の研究が多い[7],[11]．誤り検出技術については，デッドロックのように一度陥ったら以後継続的に成立するような誤りの検出のみが研究されてきた[3]．しかし，動作は継続しているにもかかわらず，システム全体としては要求された動作から外れるような誤りも存在する．このような誤りは，分散システム特有の誤りであり，通信の同期の不整合に起因する（以下では同期誤りと呼ぶ）．同期誤りが生じた場合，各プロセスはローカルには正常に動作しているため，プロセスの状態を個別に監視しているのでは検出することはできないことが多い．従って，システム全体の状態（以下では全域状態と呼ぶ）の監視が要求される[4]．特にチャネルにメッセージがない全域状態（以下では安定全域状態と呼ぶ）は，同期誤り検出に有効だと言われている[1],[2]．

本論文では，分散システムの動作中に安定全域状態を継続的に監視することを論ずる．2.では，分散システムのモデルを提示し，安定全域状態を定式化し，安定全域状態検出問題を示す．3.では，安定全域状態を検出する上でシステム実行系列内のプロセス状態，イベントの順序関係を明らかにし，全域状態が安定全域状態となるための必要十分条件を求め，安定全域状態検出問題を解く手順を示す．4.では，システム実行系列からの安定全域状態検出に必要な情報の通常のメッセージに付加（piggyback）して送る，という前提条件のも

† NTT ソフトウェア研究所，武蔵野市
NTT Software Laboratories, Musashino-shi, 180 Japan

とで各プロセスで記憶すべき情報の下界を示す．更に最小の記憶情報量で安定全域状態検出を行う分散アルゴリズムとその計算量がオーダ的に最適であることを示す．

2. 分散システムにおける安定全域状態

2.1 分散システムのモデル

分散システムは，N 個のプロセスの集合と任意のプロセス対を相互接続するチャネル(FIFO，メッセージ紛失，メッセージ内容の変更なし)の集合からなる．各プロセス $P_i(1\leqq i\leqq N)$ は，システム内の他のプロセスと非同期的にメッセージ通信を行い，送信イベント，受信イベントおよび自発的イベントによって状態遷移する(但し，システムの外の環境との送信イベント，受信イベントは自発的イベントに含める)．プロセス状態 s がイベント e によってプロセス状態 s' へ状態遷移した場合を s, e, s' と記述すると，プロセスの動作は初期状態 s_0 で始まるプロセス状態とイベントの交互系列 $s_0, e_0, s_1, e_1, s_2, \cdots$ である．但し，各プロセス状態およびイベントは，すべて異なるラベル付けがされているものとする．

プロセスの部分動作を以下の実行系列 H で定義する．

[定義 2.1] プロセス P における任意のプロセス状態あるいはイベント x_i から x_j までを表すプロセス実行系列を以下で記述する．

$H_P(x_i, x_j)=x_i, x_{i+1}, \cdots, x_j$ □

なお，本論文では特に混乱のない限り，プロセス P の実行系列を単に H_P と記述する．また，H_P を構成するプロセス状態およびイベントからなる集合(以下では H_P 構成集合と呼ぶ)を HS_P とする．更に，動作中のプロセス P の現状態を CS_P と書く．

システムの実行系列 GH を定義する．

[定義 2.2] N 個のプロセス $(P_1, P_2, \cdots, P_N)$ からなる分散システムにおけるシステム実行系列は，N 個のプロセス実行系列の組で表す．

$GH=(H_{P_1}, H_{P_2}, \cdots, H_{P_N})$ □

同様に，GH を構成するプロセス状態およびイベントからなる集合(以下では GH 構成集合と呼ぶ)を $HS(=HS_{P_1}\cup HS_{P_2}\cup\cdots\cup HS_{P_N})$ とする．

ここで，GH 内のプロセス状態およびイベントに対して反射律，推移律が成立する前順序関係“$\leqq$”を定義する．

[定義 2.3] プロセス状態 s, s'，イベント e, e'，プロセス状態あるいはイベント x, y, z に対して，

- $x\leqq x$ (反射律)
- s, e, s' なら，$s\leqq e$ かつ $e\leqq s'$
- e をメッセージ m の送信イベント，e' を同一メッセージ m の受信イベントとすると，

  $e\leqq e'$ かつ $e'\leqq e$
- $x\leqq y$ かつ $y\leqq z$ なら，$x\leqq z$ (推移律) □

これは，メッセージ転送に遅延がないと仮定した場合の，プロセス状態およびイベント間の順序関係を示していると考えられる．但し，$\neg(x\leqq y)$ は $x\not\leqq y$ と記述する．また，一つのプロセスのプロセス状態およびイベントからなる任意の集合 A に対して，“$\leqq$”において極小のプロセス状態，極大のプロセス状態を以下のように記述する．

$\min(A)=s \quad s.t.\ ^{\forall}s'\in A, s\leqq s'$

$\max(A)=s \quad s.t.\ ^{\forall}s'\in A, s'\leqq s$

“$\leqq$”について，以下の同値関係を定義する．

[定義 2.4] プロセス状態あるいはイベント x, y に対して以下が成り立つとき，x と y は“$\leqq$”に関して同値であり，$x\equiv y$ と表す．

$x\leqq y$ かつ $y\leqq x$ □

例えば，送信/受信の対応するイベント e，e' は，$e\equiv e'$ である．また，二つのプロセス P_i，P_j がそれぞれ同時にメッセージ m_{ij}，m_{ji} を送りあった場合，P_i の m_{ij} 送信後で m_{ji} 受信前の状態 s_i，P_j の m_{ji} 送信後で m_{ij} 受信前の状態 s_j は，$s_i\equiv s_j$ である．つまり，メッセージ転送に遅延がないと仮定した場合の，ある同一の瞬間に行われるとみなせるイベント，およびそれらのイベント間で瞬時に生じたプロセス状態は，“$\leqq$”に関して同値関係にある．

更に，以下のプロセス状態間の関係を定義する．

[定義 2.5] プロセス状態 $s(\in HS_{P_i})$, $s'(\in HS_{P_j})(1\leqq i, j\leqq N)$ が安定であるとは，以下を満足することであり，$s\Leftrightarrow s'$ と表す．

$s\not\leqq s'$ かつ $s'\not\leqq s$ □

プロセス P_i の状態 s_{P_i} とプロセス P_j の状態 s_{P_j} が安定であれば，s_{P_i} と s_{P_j} は同時に生起し得る．また，その時点で P_i，P_j 間のチャネルは空である．但し，$\neg(s\Leftrightarrow s')$ は $s\not\Leftrightarrow s'$ と記述する．

2.2 安定全域状態

分散システムにおける同期誤りには，デッドロックのように一度陥ったら以後継続的に成立するような誤りだけでなく，それぞれのプロセスはローカルには正常に動作しているにもかかわらず，システム全体としては要求された動作から外れるような誤りも存在する．

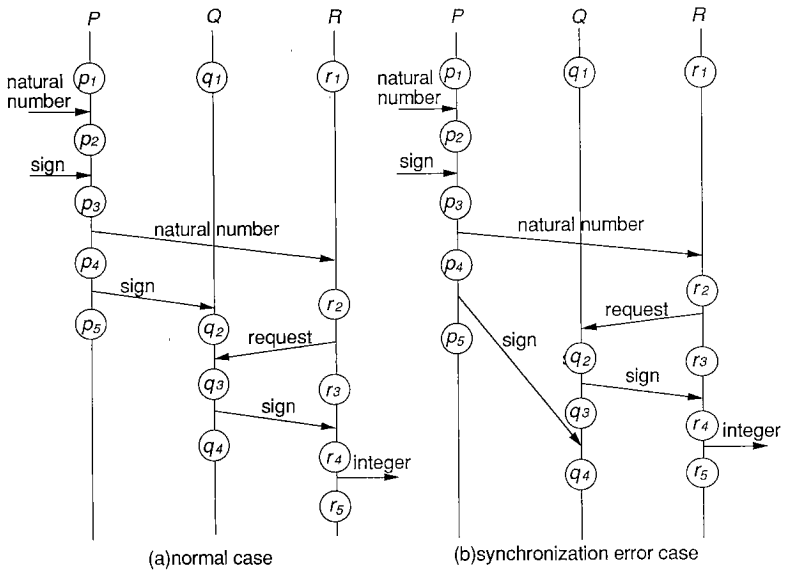


図1　同期誤りの例
Fig. 1　Synchronization error.

以下にそのような同期誤りの例を示す．

[例2.1]　三つのプロセス P, Q, R があり，P には自然数と符号 (positive, negative) が入力される．R は，P で入力された自然数と符号を整数の絶対値とその符号とみなし，整数として出力する．但し自然数は P から R へ直接送信されるが，符号はいったん Q へ送信され記憶される．Q は，R からの参照に応じてそこで記憶している符号を R へ送信する (図1(a))．

今，P から Q へ転送された符号の到着が遅れた場合を考える．R が出力する整数の符号を Q へ参照する際に，まだ Q でその符号が受信されていないならば，Q は誤った符号を R へ送信する可能性がある (図1(b))．しかも，R が誤った出力をしたとしても，P, Q, R はローカルにこの誤りを検出することはできない．□

デッドロック等を含めて，こうした現象は各プロセス動作が要求された組合せと異なることにより生ずる．そしてこのような同期誤りを検出するためにはシステム動作の代表的な全域状態を監視することが有効である[10]．

一般に分散システムにおける全域状態は，各プロセスのプロセス状態とプロセス間の各チャネルの状態の組で定義される[4]．しかし，**2.1** に述べた分散システムモデルでは，チャネルの状態はその全域状態までの各プロセスの実行系列により一意に決定することができるので，全域状態 GS を N 個のプロセス状態の組 $(s_{P_1}, s_{P_2}, \cdots, s_{P_N})$ で定義する．

分散システムにおいて監視すべき全域状態として，すべてのチャネルが空である全域状態が同期誤りの検出に有効であると言われている[1],[2]．このような全域状態は安定全域状態と呼ばれる．ある全域状態 GS におけるすべてのチャネルが空であることは，GS 以前に送信されたすべてのメッセージは GS 以前に受信され，かつ GS 以前に受信したすべてのメッセージは GS 以前に送信されたものであるということである．安定全域状態を以下に定義する．

[定義2.6]　全域状態 $GS=(s_{P_1}, s_{P_2}, \cdots, s_{P_N})$ が以下の条件を満たすとき，GS を安定全域状態と呼ぶ．

$^{\forall} i, j (1 \leqq i, j \leqq N)$ および $^{\forall} m_{ij}$ (プロセス P_i から P_j へのメッセージ) に対し，

$$-e(m_{ij}) \leqq s_{P_i} \quad \text{iff} \quad +e(m_{ij}) \leqq s_{P_j} \qquad (1)$$

但し，

- $-e(m_{ij})$: m_{ij} 送信イベント
- $+e(m_{ij})$: m_{ij} 受信イベント　□

以下の定理が成立する．

[定理2.1]　$GS=(s_{P_1}, s_{P_2}, \cdots, s_{P_N})$ が安定全域状態であることと以下の命題は等価である．

$^{\forall} i, j (1 \leqq i, j \leqq N)$ に対して，$s_{P_i} \Leftrightarrow s_{P_j}$　(2)

(証明)

・必要条件　以下の m_{ij} が存在する場合を考える：

$-e(m_{ij}) \leqq s_{P_i}$ かつ $s_{P_j} \leqq +e(m_{ij})$

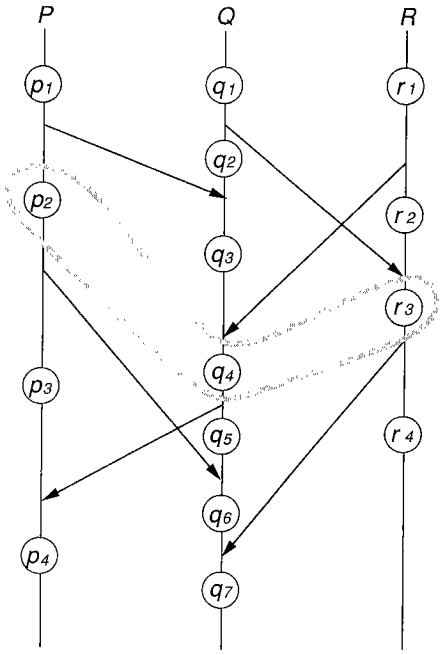


図2 安定全域状態の例
Fig. 2 Stable global state.

$s_{P_j} \leqq s_{P_i}$ より $s_{P_i} \not\leqq s_{P_j}$ となる．これは，条件(2)を満たさない．

$-+e(m_{ij}) \leqq s_{P_j}$ かつ $s_{P_i} \leqq -e(m_{ij})$

$s_{P_i} \leqq s_{P_j}$ より $s_{P_i} \not\leqq s_{P_j}$ となる．これは，条件(2)を満たさない．

・十分条件　$\forall m_{ij}$ に対して，条件(1)が成立すれば，

$s_{P_i} \Leftrightarrow s_{P_j}$

は明らかである．任意の二つのプロセスに対して条件(1)が成立するなら，GS は安定全域状態である．

但し，安定全域状態は図2に見られるように必ずしも実時間において実際に生じる状態でないことに注意する．プロセス状態 p_2, q_4, r_3 は同時に生起していないが，互いに安定な関係にあるため，それらの組 (p_2, q_4, r_3) は安定全域状態であると言う．

図1では，システムが正常な動作をしていれば (p_5, q_3, r_3) が安定全域状態として検出される．しかし，上の安定全域状態を検出できず，代わりに (p_4, q_2, r_3) が安定全域状態として検出されれば，同期誤りであると判定することができる．

本論文では，システムの動作中に，システム実行系列から安定全域状態を検出することを論ずる．検出専用のプロセスを想定せずに，以下の安定全域状態検出問題を考える．

[問題2.1] 各プロセスは，現状態 CS の時点までに得たシステム実行系列から安定全域状態の系列を順次検出する．その際，それぞれのプロセスによって検出された安定全域状態に含まれるプロセス状態の集合は，すべての安定全域状態に含まれるプロセス状態の集合に一致するものとする．

一般に同一のプロセス状態が複数の相異なる安定全域状態に含まれるために，すべての安定全域状態を検出することは実際的でない．問題2.1は，安定全域状態に含まれているプロセス状態は，必ずいずれかの安定全域状態に含まれて検出されることを要求している．

文献(12)では，誤り回復に必要なチェックポイントを記憶するために，起動をかけ，専用の制御メッセージを送り合い，安定全域状態を検出するアルゴリズムを提案している．このアルゴリズムは常時動作させるものではない．また文献(13)では，本論文で提案するアルゴリズムと同様に，安定全域状態を検出するのに必要な情報を通常のメッセージに載せて送り合うアルゴリズムを提案しているが，実行系列をそのまま送っており，安定全域状態検出のためには冗長な情報を含んでいる．本論文では，誤り検出のために，最小の情報で安定全域状態を継続的に監視するアルゴリズムを提案する．

3. システム実行系列内のプロセス状態，イベント

本章では安定全域状態を検出する上で，システム実行系列内のプロセス状態，イベントの性質について述べる．

GH 構成集合 HS を，同値関係 "≡" によって同値類 μGH に分割する．このとき相異なる同値類 μGH, $\mu GH'$ に対して，以下が成立することは明らかである．

$\exists x(\in \mu GH), y(\in \mu GH')$ に対して，$x \leqq y$ なら，

$\forall x'(\in \mu GH), y'(\in \mu GH')$ に対して，$x' \leqq y'$

従って同値類 μGH に対し，以下で定義する "≦" は対称律 ($\mu GH \leqq \mu GH'$ かつ $\mu GH' \leqq \mu GH$ なら，$\mu GH = \mu GH'$) を満たし，μGH 間の順序関係となる．

[定義3.1] μGH, $\mu GH'$ と $\exists x(\in \mu GH)$, $y(\in \mu GH')$ に対して，

$x \leqq y$ なら，$\mu GH \leqq \mu GH'$ □

同様に，安定の関係を次の μGH 間の関係に拡張する．

[定義3.2] μGH, $\mu GH'$ が安定であるとは，以下を満

足することであり，$\mu GH \Leftrightarrow \mu GH'$ と書く．

$\mu GH \not\leqq \mu GH'$ かつ $\mu GH' \not\leqq \mu GH$ □

ここで，プロセス状態が安定全域状態に含まれない条件は以下のとおりである．

[定理 3.1] プロセス状態 s が以下の条件を満たすことと，安定全域状態に含まれないことは等価である．

$\exists s'(\neq s)$ に対して，$s, s' \in \mu GH$

(証明)

・必要条件　あるプロセス P_i のプロセス状態 s が安定全域状態に含まれないとは，以下のプロセス P_j が存在することである．

$\forall s_j(\in HS_{P_j})$ に対して，$s \not\leqq s_j$　(3)

更に，H_{P_j} には以下のプロセス状態 s_{j1}，s_{j2} が存在する．

$s_{j1} = \min(\{s_j \mid s_j \in HS_{P_j}, s \leqq s_j\})$　(4)

$s_{j2} = \max(\{s_j \mid s_j \in HS_{P_j}, s_j \leqq s\})$　(5)

すべての s_j に対して条件(3)を満たさなければならない．s_{j1} と s_{j2} の順序関係が以下の場合を考える．但し，$s_{j2} \leqq s_{j1}$ で s_{j2} と s_{j1} の間にプロセス状態 s_{j3} が存在する場合は，$s \Leftrightarrow s_{j3}$ となり条件(3)に反する．

(1) $s_{j1} \leqq s_{j2}$ の場合：

$s \leqq s_{j1} \leqq s_{j2} \leqq s$

となり，$s \equiv s_{j1}(s_{j2})$ となるプロセス状態が存在する．つまり，s を含む μGH に $s_{j1}(s_{j2})$ が含まれる．

(2) s_{j2}, e_j, s_{j1} の場合 (e_j：送信または受信イベント)：s_{j1} および s_{j2} が式(4)，(5)であるためには，e_j に対応する受信または送信イベント e_k をもつプロセス P_k の H_{P_k} に，以下の送信または受信イベント e_{k1}，e_{k2} が存在する．

$e_k \leqq e_j$ より，$s \leqq e_{i1} \leqq e_{k1} \leqq e_k$

$e_j \leqq e_k$ より，$e_k \leqq e_{k2} \leqq e_{i2} \leqq s$

但し，$e_{i1}, e_{i2} \in H_{P_i}$

従って，

$s \leqq e_{k1} \leqq e_k \leqq e_{k2} \leqq s$

となり，$e_{k1} \leqq s_k \leqq e_{k2}$ なる s_k に対して，$s \equiv s_k$ が成立する．つまり，s には同一の μGH に含まれる s_k が存在する．

・十分条件　同値類 μGH に含まれるプロセス状態 s が唯一のプロセス状態でないとは，$s(\in HS_{P_i})$ に対して，$s \equiv s_{P_j}$ なる $s_{P_j}(\in HS_{P_j}, i \neq j)$ が存在することである．なぜなら，$s \leqq s'$ なるプロセス状態 $s'(\in HS_{P_i})$ が同一の μGH に含まれるとすれば，同時に $s' \leqq s$ でなければならない．従って，他のプロセス P_j との通信によって，$s \leqq s' \leqq s_{P_j} \leqq s$ なる s_{P_j} が存在するはずである ($s' \leqq s$ のときも同様である)．

s を含む安定全域状態があるとすれば，$s \Leftrightarrow s_j(s_j \in H_{P_j})$ なるプロセス状態 s_j が少なくとも一つなければならない．しかし，H_{P_j} に s_j が存在するとすれば，$s \leqq s_{P_j}$ より $s_j \leqq s_{P_j}$ である．一方 $s_{P_j} \leqq s$ より $s_j \leqq s$ が成立し，s_j と s が安定であるということに反するため，このような s_j は存在しない．従って，s は安定全域状態に含まれない．

従って，安定全域状態については，以下の定理が成立する．

[定理 3.2] $GS = (s_{P_1}, s_{P_2}, \cdots, s_{P_N})$ について以下の条件が満たされるとき，GS は安定全域状態である．但し，$\forall i(1 \leqq i \leqq N)$ に対し，$s_{P_i} \in \mu GH_i$ とする．

$\forall i, j(1 \leqq i, j \leqq N)$ に対し，

$\mu GH_i \Leftrightarrow \mu GH_j$　(6)

$|\mu GH_i| = 1$　(7) □

条件(6)については，定理2．1より明らかである．条件(7)については，定理3.1より明らかである．

定理3.2より，問題2.1の安定全域状態検出問題は，以下の手順で解けることがわかる．

1．μGH に複数のプロセス状態が含まれる場合，その μGH に含まれるすべてのプロセス状態を削除することにより，一つのプロセス状態のみを含む μGH を残す．

2．1で残ったプロセス状態を含む μGH の集合から，互いに安定である μGH の組を検出する．

4. 安定全域状態の検出

4.1 検出に必要な制御情報

本論文では安定全域状態を以下の前提で検出する．

[前提 4.1] 安定全域状態の検出に必要な情報は，通常のメッセージに付加して転送する． □

この前提は検出専用のメッセージを使用しないことにより，システムの本来の実行系列を保存するための条件である．前提4.1のもとでは，各プロセスは安定全域状態の検出に必要な情報(以下では制御情報と呼ぶ)を記憶し，通常のメッセージに載せて交換し収集する．制御情報はメッセージ通信により CS_P が進行するに連れて変化していく．以下では必要十分な制御情報を明らかにする．

3.で述べたことから，制御情報には一つのプロセス状態のみを含む μGH と，各 μGH 間の順序関係を判定するための情報が必要である．従って，定理3.1を満たすプロセス状態を制御情報に含める必要がない．また，現時点CSまでに得られた μGH に受信イベントが存在する場合，対応する送信イベントも既に同一 μGH 内にあることは明らかであるため，その場合の送信イベン

トも制御情報に含める必要がない．

以下では，μGH から定理 3.1 を満たすプロセス状態，および互いに対応する送信/受信イベントがそろった場合の送信イベントを除いた集合を $r\mu GH$ とする．$r\mu GH$ は一つのプロセス状態を含むか，一つ以上のイベントを含むかのいずれかになる．μGH 間に成立する順序関係は $r\mu GH$ 間でも満たされることは明らかである．

プロセス P の実行系列 H_P の H_P 構成集合 HS_P が，

$HS_P \subseteq \mu GH \cup \mu GH' \cup \cdots \cup \mu GH''$

$\mu GH \leqq \mu GH' \leqq \cdots \leqq \mu GH''$

のとき，$r\mu GH$ の系列で表されたプロセス縮退実行系列 rH_P で安定全域状態を検出できる．

$rH_P = r\mu GH, r\mu GH', \cdots, r\mu GH''$

なお H_P と同様に，特に混乱のない限りプロセス P の縮退実行系列を単に rH_P と記述する．更に，rH_P 構成集合を rHS_P とする．

次に，既に安定全域状態であると判定された全域状態より前の制御情報は，その全域状態以降の安定全域状態の検出には必要がない．プロセス P の CS_P が s_P である時点までに判定された安定全域状態のうち，最後の安定全域状態を last_SGS(s_P)=($s_{P1}, s_{P2}, \cdots, s_{PN}$) とすると，last_SGS($s_P$) は定理 3．2 の式(6)を満たすため，$e \leqq s_{Pi} (1 \leqq i \leqq N)$ なるイベント e は，last_SGS(s_P) 以降の $r\mu GH$ の前後関係に影響がないためである．

以上より，安定全域状態検出に必要十分な制御情報は以下のとおりである．

[定理 4.1] プロセス P の CS_P が s_P である時点において，安定全域状態検出のために必要十分な制御情報は，last_SGS(s_P)=($s_{P1}, s_{P2}, \cdots, s_{PN}$) とすると，以下のものである．

($rH_{P1}, rH_{P2}, \cdots, rH_{PN}$)

但し，$^{\forall} i (1 \leqq i \leqq N)$ に対して，

$\min(rHS_{Pi}) = s_{Pi}$

(証明) ここでは必要性を示す．$r\mu GH$ に単独で含まれるプロセス状態情報の必要性は明らかである．制御情報のうち受信イベントが失われれば，それ以降にある安定全域状態に含まれるプロセス状態との順序関係が得られず，安定全域状態が検出できない場合がある．受信イベントとの対応が得られていない場合の送信イベントが失われれば，安定な関係にないプロセス状態の組を安定であると判定し，誤った安定全域状態を検出してしまう場合が生じる．

十分性については次節で示す．次節では，プロセス P の CS_P が s_P である時点における制御情報を View(s_P) と記述し，最小の情報を用いて正しく安定全域状態を検出するアルゴリズムを与える． □

4.2 制御情報の構造

各プロセスは，定理 4.1 で示した制御情報を記憶し，送信メッセージに制御情報を付加する．メッセージ受信時には，受信側プロセス自身のもつ制御情報と受信メッセージに付加されていた制御情報から，安定全域状態が存在するかどうかを調べる．

本節では制御情報 View を有向グラフで表現することによって[14]，安定全域状態検出アルゴリズムを説明する．

図 3(a)は実システムの実行系列 GH，図 3(b),(c)は，それぞれ q_2，r_3 における制御情報 View(q_2)，View(r_3) である．

View において，ノードもラベルのついている有効枝もそれぞれ縮退実行系列における $r\mu GH$ を表す．ノードはイベント (送信イベント (○)，受信イベント (●)，自発的イベント(●)) の集合，または開始点 (◆)，終端点 (△) を表す．開始点は一つ，終端点は各プロセス対応に一つずつあり，それぞれ縮退実行系列 rH のデリミタを表す．

有向枝 (→,⇢) につけられたラベルは，定理 3．2 の条件(7)を満たすプロセス状態を表す．有向枝→には必ずラベルが付き，直前のイベントを含む $r\mu GH$ または開始点を始点，直後のイベントを含む $r\mu GH$ または自プロセスの終端点を終点とする．これは，現時点 CS までにその始点のノードと終点のノード間のプロセス縮退実行系列が得られていることを表す．有向枝→にはラベルの有無で 2 種類あり，ラベル付き有効枝→は，開始点または直前のイベントを含む $r\mu GH$ を始点，受信をまだ確認していない送信イベントを含む $r\mu GH$ または他プロセスの終端点を終点とする．ラベルなし有向枝⇢は，単に終端点との前後関係を示す．View の初期値は，開始点から自プロセスの終端点へのラベルつき有向枝→と他プロセスの終端点へのラベル付き有向枝⇢からなる．

View は次の性質をもつ．

(1) View には，定理 3.2 の条件(7)より，定理 3.1 を満たすプロセス状態に対応する有向枝は存在しない．

(2) View において，一つのプロセス状態が 2 本の有向枝にラベル付けされることはない．

(3) イベントの集合を表すノードの入枝数と出枝数は等しい．

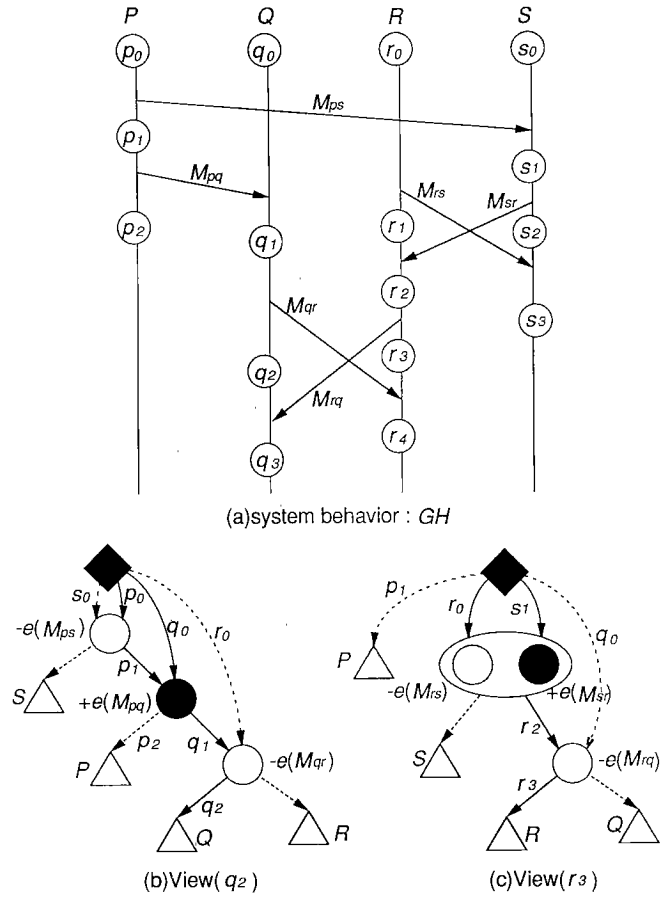


図3　システム動作と View の例
Fig. 3　System behavior and View.

（4） 開始点から出枝の集合は，最後に検出した安定全域状態(last_SGS)を示す．例えば，View(q_2)，View(r_3)においては，last_SGS(q_2)=(p_0, q_0, r_0, s_0)と last_SGS(r_3)=(p_1, q_0, r_0, s_1)となる．

（5） 開始点と終端点集合を分ける最小枝カットセットの枝の数は，プロセス数(N)に等しい．

View の性質2より，あるプロセスの最後の安定全域状態以降の縮退実行系列 rH は，開始点◆から各プロセスの終端点△への1本の道として表される(以下では rH(View) と記述する)．例えば，図3(b)において，rH_P(View(q_2)) は次のように表現されている．

$$rH_P(\mathrm{View}(q_2)) = \blacklozenge \xrightarrow[-e(M_{ps})+e(M_{pq})]{p_0} \bigcirc \xrightarrow{p_1} \bullet \xrightarrow{p_2} \Delta P$$

この場合は一つの有向枝，点がそれぞれ一つの $r\mu GH$ を表しているが，図3(c)の rH_R(View(r_3)) のように二つのイベントで一つの $r\mu GH$ を構成しているものもある．

$$rH_R(\mathrm{View}(r_3)) = \blacklozenge \xrightarrow[-e(M_{rs})+e(M_{sr})-e(M_{rq})]{r_0} (\bigcirc\bullet) \xrightarrow{r_2} \bigcirc \xrightarrow{r_3} \Delta R$$

しかもこの $r\mu GH$ 内には，Viewの性質1より，r_1，s_2 に相当する有向枝は含まれていない．

View における安定全域状態とは，以下の定理で表される全域状態である．

[定理4.2]　$GS=(s_{P_1}, s_{P_2}, \cdots, s_{P_N})$ が安定全域状態であることと，View において GS の要素からなる枝カットセットが，開始点と終端点集合を分ける最小枝カットセットであることは等価である．但し，枝カットセットの要素はプロセス状態がラベル付けされている有向枝に限る．

(証明)

・必要条件　　対偶を示す．View の性質1より，定理3.2の条件式(7)を満たさないプロセス状態は，

View 上の有向枝として表されないから自明．GS に含まれるプロセス状態が条件(6)を満たさない(すなわち順序関係がある)とすれば，その GS を表す枝カットセットを複数回横切る有向枝の系列 (rH) があることになり，始点と終端点集合を分ける最小枝カットセットにならない．

・十分条件　View の性質(1)より，安定全域状態に含まれる任意のプロセス状態 s_{Pi} は，View においてラベル付き有向枝として表され，更に条件(6)より，rH_{Pi} の開始点から s_{Pi} までと他の任意の rH_{Pj} の s_{Pj} から終端点までが交差することはない．なぜなら，交差しているノードを $r\mu GH'$ とすると $r\mu GH_i(\ni s_{Pi})$, $r\mu GH_j(\ni s_{Pj})$ との間で $r\mu GH_j \leqq r\mu GH' \leqq r\mu GH_i$ となり，$s_{Pi} \notin s_{Pj}$ となるためである．従って，安定全域状態を表す枝集合は，開始点と終端点集合を分ける最小枝カットセットとなる．

4.3 安定全域状態検出アルゴリズム

プロセス P_i において実行されるアルゴリズムを図4に示す．

メッセージ送信時は，View を付加するだけなので，以下は受信時の処理について説明する．

4.3.1 View の合成と安定全域状態の検出

記憶している View(s_{Pi}) と受信した View(s_{Pj}) を以下の手順で合成する．

(1) View(s_{Pi}) と View(s_{Pj}) の開始点を統合する

定理4.1より，安定全域状態の検出に必要のない last_SGS(s_{Pi}) および last_SGS(s_{Pj}) より前の制御情報を削除する．

(2) View(s_{Pj}) 内の各 rH のうち View(s_{Pi}) にない部分を View(s_{Pi}) に追加する

View(s_{Pi}) と View(s_{Pj}) におけるプロセス P_k の max(rHS_{Pk}) 同士を比較し，View(s_{Pj}) の rH の方が順序関係 "≦" において新しい場合，その rH を View(s_{Pi}) に追加する．その際対応する送信イベント○，受信イベント●があると，縮退して一つの受信イベント●とする．

(3) 安定全域状態とならないプロセス状態を削除する

View において，定理3.1を満たすプロセス状態は有向閉路内に現れる．なぜなら，有向閉路上の $r\mu GH$ に含まれるプロセス状態の集合は，一つの $r\mu GH$ に属するためである．従って，有向閉路上の有向枝は，各 μGH を一つの $r\mu GH$ に縮退する際に削除する．有向グラフ内での有向閉路の発見は，文献(5)と同様の方法で可能である．

```
メッセージ m_ij 送信時
procedure send_message(input: View(s_Pi), m_ij; output: View(s_Pi))
begin
  View(s_Pi) に送信イベント，遷移後のプロセス状態を追加する;
  View(s_Pi) を付加して m_ij を送る;
end

メッセージ m_ji 受信時
procedure receive_message(input: View(s_Pi), m_ji; output: View(s_Pi))
begin
  View(s_Pi) に受信イベント，遷移後のプロセス状態を追加する;
  View(s_Pi) と m_ji に付加されている View(s_Pj) を合成する;
  合成された View(s_Pi) から安定全域状態を検出する;
end

自発的遷移時
procedure self_transition(input: View(s_Pi); output: View(s_Pi))
begin
  View(s_Pi) に自発的イベント，遷移後のプロセス状態を追加する;
  View(s_Pi) から安定全域状態を検出する;
end
```

図4　安定全域状態検出アルゴリズム
Fig. 4　Stable global state detection algorithm.

(4) View から安定全域状態の系列を検出する

開始点から順に定理4.2を満たす最小枝カットセットを探す．View の性質(4)より，開始点の出枝数は最小枝カットセットであり，性質(5)よりその本数は N である．ノードを一つ進め，そのノードの入枝と出枝を入れ替えた枝集合も，View の性質(3)より，最小枝カットセットとなる．その最小枝カットセットがすべてラベル付きの有向枝であれば安定全域状態であると判定し，その最小枝カットセットまでの $r\mu GH$ を開始点に縮退させる．この手順を繰り返すことによって，View から安定全域状態の系列を検出する．

View の性質(1)および定理4.2より，求めるべき安定全域状態はすべて検出し，誤った安定全域状態を検出することはないことは明らかである．

4.3.2 アルゴリズムの動作例

図5に，図3で示したプロセス Q が M_{rq}(View(r_3) が付加されている) を受信した際，View(q_2) と View(r_3) を合成して新たに View(q_3) を生成する過程を示す．

(1) last_SGS(View(q_2)) と last_SGS(View(r_3)) を比較し，開始点◆からの有効枝を (p_1, q_0, r_0, s_1) とする(図3(b),(c)参照)．

(2) max(rH) 同士の順序関係を比較し，P, Q に関しては rH_P(View(q_2)), rH_Q(View(q_2)) をそのまま残し，R, S に関しては View(r_3) の rH_R(View(r_3)), rH_S(View(r_3)) を View(q_2) に付加する(図5(a)参照)．

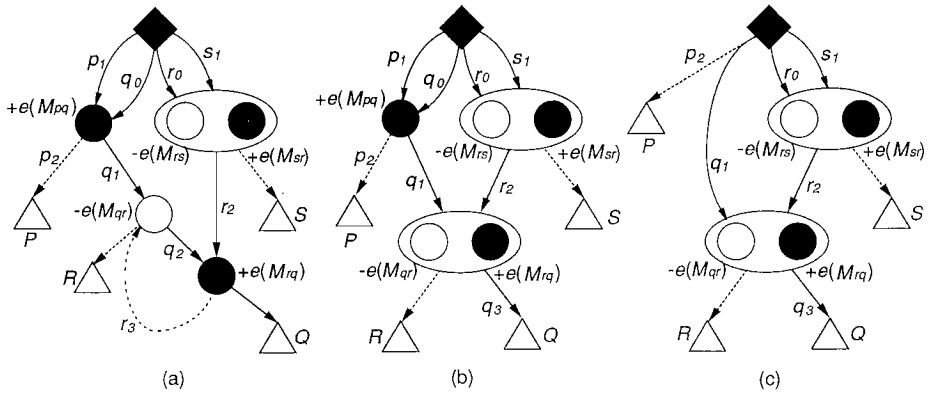
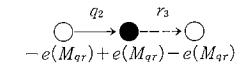


図5　Viewの合成例
Fig. 5　A composition of View (q_2) and View (r_3).

（3）　図5（a）の有向閉路：



を縮退して，安定全域状態に含まれないプロセス状態 q_2，r_3 を削除する（図5（b）参照）．

（4）　図5（b）の$+e(M_{pq})$●直後の最小枝カットセット（p_2, q_1, r_0, s_1）は安定全域状態であると判定する．安定全域状態と判定された最小枝カットセットまでを開始点に縮退し，最終的に得られたViewがView(q_3)である（図5（c）参照）．

4.4　アルゴリズムの評価

安定全域状態検出アルゴリズムの計算量の評価を行う．以下 $r\mu GH$ の数を n とする．

・Viewの合成の計算量：rH の追加は $r\mu GH$ を順に付加するため $O(n)$．有向閉路の縮退の計算量は，文献（5）より $O(n)$．全体として $O(n)$．

・安定全域状態の検出の計算量：開始点から安定全域状態までのすべての有効枝を順次探索し，かつ同じ有効枝は通らないため，O（有効枝の数）．

有向枝の数$=O(n)$より，計算量は $O(n)$ である．ここで，$O(n)$ はViewの大きさのオーダである．検出専用のメッセージを使用しない前提における必要最小の制御情報はViewであるから，このアルゴリズムはオーダ的に最適である．

5.　むすび

分散システムの同期誤り検出技術の一技法として安定全域状態の監視法について論じた．安定全域状態は，メッセージの送受信の区切りとして，同期誤りの検出に有効であるとされている．本論文では，動作中に得られた分散システムのシステム動作に対して安定全域状態の系列を，「検出に必要な制御情報は，通常のメッセージに付加して転送する」という前提のもとで，いずれかのプロセスで検出する問題について考えた．

上の前提のもとで，記憶すべき制御情報量の下界と最小の情報量で安定全域状態を検出するアルゴリズムを示した．また，提案した安定全域状態検出アルゴリズムは，上の前提のもとで，最小の計算オーダで安定全域状態の系列を検出できることを明らかにした．このアルゴリズムを転送情報量の面で最適化すること，更に安定全域状態監視と結び付けた通信ソフトウェア設計法を検討することが今後の課題である．

また，フォールトトレランス技術のもう一方の要素技術である誤り回復技術では，同期誤りから正常な状態へ戻ることを目的としているが，安定全域状態は同期誤りに陥る前の正常な状態（リカバリポイント）としても有効と考えられる[7],[12],[13]．

謝辞　本テーマに取り組むきっかけを与えて頂き，御討論頂いたNTTソフトウェア研究所市川主幹研究員，同じく御討論頂いた今瀬主幹研究員，伊藤主幹研究員，加藤主任研究員に深謝します．

文　献

（1）　Zafiropulo P., West C. H., Rudin H., Cowan D. D. and Brand D. : "Towards analyzing and systhesizing protocols", IEEE Trans. Commun., **COM-28**, 4, pp. 651-661 (April 1980).

（2）　Brand D. and Zafiropulo P. : "On communicating finitestate machines", J. Assoc. Comput. Mach., **30**, 2, pp. 323-342 (April 1983).

（3）　Chandy K. M., Misra J. and Haas L. M. : "Distributed deadlock detection", ACM Trans. Comput. Syst., **1**, 2,

pp. 144-156 (May 1983).
(4) Chandy K. M. and Lamport L. : "Distributed snapshots : Determining global states of distributed systems", ACM Trans on Comp. Syst., **3**, 1, pp. 63-75 (Feb. 1985).
(5) Aho A. V., Hopcroft J. E. and Ullman J. D. : "The Design and Analysis of Computer Algorithms", 野崎ほか訳, サイエンス社, pp. 174-177 (Oct. 1977).
(6) 市川晴久, 伊藤正樹, 加藤 順 : "並行処理の監視性と通信プロトコル実現への適用", 信学論 (B), **J70-B**, 5, pp. 565-575 (1987-05).
(7) Koo R. and Toueg S. : "Checkpointing and rollbackrecovery for distributed systems", IEEE Trans. Software Eng., **SE-13**, 1, pp. 23-31 (Jan. 1987).
(8) Lamport L. : "Time, Clocks, and the Ordering of Events in a Distributed System", Commun. ACM, **21**, 7, pp. 558-565 (July 1978).
(9) Lamport L. : "Specifying concurrent program modules", ACM Trans. Program. Lang. & Syst., **5**, 2, pp. 190-222 (April 1983).
(10) 曽根岡昭直, 今瀬 真 : "並行システムにおけるグローバル状態監視法", 昭63信学春季全大, SD-5-2 (1988-03).
(11) Venkathsh K., Radhakrishman T. and Li H. F. : "Optimal checkpointing and local recording for dominofree rollback recovery", Inf. Process. Lett., 25, pp. 295-303 (July 1987).
(12) 角田良明, 若原 恭 : "通信ソフトウェアのフォールトトレラント化", 信学技報, **IN87-70** (1987-11).
(13) 角田良明, 若原 恭 : "効率良いプロトコルエラー回復のためのメッセージ送受信系列の分散収集法", 信学論(D-I), **J73-D-I**, 2, pp. 134-140 (1990-02).
(14) 森保健治, 曽根岡昭直, 真鍋義文 : "分散システムにおける同期誤り検出法", 信学技報, **IN88-152** (1989-03).

(平成2年5月16日受付, 10月29日再受付)

森保 健治

昭60早大・理工・電子通信卒. 昭62同大大学院修士課程了. 同年日本電信電話株式会社入社. 以来, 通信ソフトウェアの研究に従事. 現在, NTTソフトウェア研究所ソフトウェア開発技術研究部研究主任.

曽根岡昭直

昭55東大・工・電子卒. 昭57同大大学院修士課程了. 同年日本電信電話公社(現NTT)入社. 以来, 通信網設計, グラフ理論, 分散システムの研究に従事. 現在, NTTソフトウェア研究所ソフトウェア開発技術研究部主任研究員. 工博. 昭63年度学術奨励賞受賞. IEEE会員. ACM会員.

真鍋 義文

昭58阪大・基礎工・情報卒. 昭60同大大学院修士課程了. 同年日本電信電話株式会社入社. 以来, グラフ理論, 通信網信頼性, 分散システムの研究に従事. 現在, NTTソフトウェア研究所ソフトウェア基礎技術研究部研究主任. 情報処理学会会員. ACM会員.