解　説

フォールトトレラント分散システム向けアルゴリズム

4. 自己安定アルゴリズムについて†

増澤利光††　片山喜章††

1. はじめに

通常の分散アルゴリズムでは，アルゴリズムの実行開始時に，各プロセッサはあらかじめ決められた初期状態にあり，ネットワーク中にはリンクを伝搬中のメッセージは存在しないと仮定することが多い．これに対し，自己安定アルゴリズム (self-stabilizing algorithm) では，ネットワークの初期状況に何も仮定しない．つまり，アルゴリズム実行開始時の各プロセッサの状態，リンクを伝搬中のメッセージの有無やその内容に関わらず，有限時間内に問題を解くことのできる分散アルゴリズムが自己安定アルゴリズムである．この特徴から，自己安定アルゴリズムは次のような故障耐性をもつ．

1. 各プロセッサのプログラムさえ無事ならば，プロセッサのデータの一部（プログラムカウンタを含む）の破壊，リンクを伝搬中のメッセージの紛失，内容の変質などの「一時誤り (transient error)」が生じても，その後十分に長い間故障が生じなければ問題を解くことができる．

2. アルゴリズムの実行中に形状が変化する動的ネットワーク (dynamic network) でも，十分に長い間形状変化が生じなければ問題を解くことができる．したがって，プロセッサやリンクの故障や復旧が生じてもその後十分に長い間故障や復旧が生じなければ，ネットワークが無故障な部分で問題を解くことができる．

1974年に Dijkstra[9] が自己安定アルゴリズムの概念を導入し，リングネットワークで相互排除を実現するための自己安定アルゴリズムを提案した．その後，自己安定アルゴリズムの研究はそれほど活発に行われなかった．しかし，Lamport[21] が自己安定アルゴリズムの分散環境における故障耐性に対する重要性を指摘して以来，自己安定アルゴリズムの研究は，この数年，特に盛んに行われている．自己安定アルゴリズムのこれまでの研究の多くは相互排除を対象としたものであるが，最近では生成木構成問題やリングネットワークの方向付け (orientation) 問題など，他の問題に対する自己安定アルゴリズムも提案されている．

自己安定アルゴリズムの研究は，いくつかの異なる仮定の下で議論されている．そこで，本稿の **2.** では，これまでに用いられてきた主なネットワークモデルをいくつかの要因で分類・整理しながら紹介する．**3.** では，自己安定アルゴリズムの具体例として，Dijkstra が示したアルゴリズム[9] を紹介する．また，**4.** では今までに知られているいくつかの結果を簡単に紹介する．

2. モデル

2.1 ネットワーク

ネットワーク N は，2項組 $N=(P, L)$ で定義される．ここで，P はプロセッサの集合を表し，L は通信リンクの集合を表す．つまり，L は P の相異なる要素の非順序対の集合であり，$(p, q)\in L$ のとき，p, q 間に全2重リンクが存在する．またこのとき，p, q は互いに他方を隣接プロセッサとよぶ．本稿では，プロセッサ数を n とし，プロセッサを $p_0, p_1, \cdots, p_{n-1}$ と表記する．

プロセッサ p が d 個のプロセッサと隣接するとき，p はこれら d 個の隣接プロセッサを第1隣接プロセッサ，第2隣接プロセッサ，…，第 d 隣接プロセッサとして区別している．以下では，プロセッサ p の第 j 隣接プロセッサを $p[j](1\leq j\leq d)$ と表す．

† On Self-Stabilizing Algorithms by Toshimitsu MASUZAWA and Yoshiaki KATAYAMA (Dept. of Information and Computer Sciences, Faculty of Engineering Science, Osaka University).

†† 大阪大学基礎工学部情報工学科

各プロセッサは状態機械 (state machine) である．各プロセッサは，自分の状態と隣接プロセッサの状態から次の状態を決定する．つまり，各プロセッサ p_i は，2項組 (S_i, T_i) で定義される．ここで S_i は p_i の状態集合，T_i は p_i の状態遷移関数で $S_i \times S'_1 \times S'_2 \times \cdots \times S'_d$ から S_i への関数 $(T_i : S_i \times S'_1 \times S'_2 \times \cdots \times S'_d \rightarrow S_i)$ である．ただし，d は p_i の隣接プロセッサの数，$S'_j (1 \leq j \leq d)$ は $p_i[j]$ の状態集合を表す ($p_i[j]=p_k$ ならば $S'_j = S_k$)．つまり，p_i の状態が $s(\in S_i)$，$p_i[j]$ の状態が $s_j(\in S_j)$ のときに p_i が動作すれば，p_i の状態は $T_i(s, s_1, s_2, \cdots, s_d)$ に変化する．

分散アルゴリズムとは，ネットワークの各プロセッサの状態集合と状態遷移関数を定めるものと考えられる．

ここで示したモデルでは，各プロセッサは隣接プロセッサの状態を知ることができると仮定している．分散アルゴリズムがよく議論されているネットワークモデルは，メッセージ通信（しかも，メッセージがリンクを伝わる伝送遅延は有限だが，上限がない）を仮定することが多い．自己安定アルゴリズムで用いられているネットワークモデルも，通信方式の違いにより，次のように分類できる．

1. 状態通信モデル：すべての隣接プロセッサの状態を知ることができる．本稿では，特に断わらない限り状態通信モデルを用いる．

2. レジスタ通信モデル：各リンクの各方向に対して一つのレジスタが存在し，そこに隣接プロセッサへのメッセージを書き込むことにより通信を行う．つまり，リンク $(p_i, p_j) \in L$ は，二つのレジスタ R_{ij}, R_{ji} から構成されており，p_i から p_j への通信はレジスタ R_{ij} を介して行い，p_j から p_i へはレジスタ R_{ji} を介して行う．

3. メッセージ通信モデル：各プロセッサはメッセージの送受信によって通信を行う．

初期の自己安定アルゴリズムでは状態通信モデルを主に用いていたが，このモデルでは隣接プロセッサごとに異なる情報を通信することができない．このため，リングネットワークのような対称性のあるネットワークでは，たとえ特別なプロセッサが一つだけ存在しても（次の分類における準均一なネットワーク），相互排除を行う自己安定アルゴリズムが存在しない[18]．ただし，特別なプロセッサが一つだけ存在し，すべてのプロセッサが同一方向（時計回りか反時計回り）に方向付けられた方向感覚付きのリングネットワーク* では相互排除問題を解ける[9]．一方，レジスタ通信モデルでは，隣接プロセッサごとに異なる情報を通信できる．しかし，レジスタにプロセッサの状態を書き込めるとしても，状態遷移とレジスタへの書込みが一つの原子動作 (atomic step) で行えないようなモデルでは，レジスタに書き込まれている状態が現在の状態であるとは限らない（一つ前の状態が書き込まれている可能性がある）．このことが，レジスタ通信モデルで問題を解くことを困難にしている．メッセージ通信モデルでは，アルゴリズム実行開始時の状態が，（メッセージを送信していないにもかかわらず）メッセージ送信直後の状態になっている可能性があり，メッセージを実際に送信したかどうかを確認できない．このため，メッセージに対する返事を待っているとデッドロックに陥る危険がある．また，アルゴリズム実行開始時にいくつかのメッセージがリンクに存在しているか分からないので，これらの（誤った）メッセージの影響が無限に続く可能性がある．これらのために，レジスタ通信モデルに比べ，メッセージ通信モデルで自己安定アルゴリズムを開発することは，より困難であると考えられている[20]．

また，ネットワーク中のすべてのプロセッサを同一の状態機械とするかどうかによって，次のように分類できる．

1. 均一 (uniform)：すべてのプロセッサは同一の状態機械である（同一のプログラムを実行する）．分散アルゴリズムの議論では，同一のプログラムを実行するとしながらも，実際には各プロセッサが相異なる識別子をもち，識別子をパラメタとするプログラムを実行することを許すことが多い．しかし，ここで均一なネットワークとはそのような識別子も存在しないネットワークである．このようなネットワークは匿名ネットワーク (anonymous network) と呼ばれることもある．

2. 準均一：定数個（通常1個）の特別なプロセッサがあり，これら以外のプロセッサは同一の状態機械である．定数個の特別なプロセッサはそ

* リングネットワークにおいて，各プロセッサ p に対し $p[1]=q$ ならば $q[2]=p$ が成り立つとき，方向感覚付きのリングネットワークと呼ぶ．

れぞれ異なる状態機械であってもよい.

3. 識別子付き（識別子パラメタ化）：各プロセッサは相異なる識別子をもち，識別子をパラメタとする状態機械である（識別子をパラメタとするプログラムを実行する）.

形状に対称性のある均一なネットワークでは相互排除問題を解く（決定性）自己安定アルゴリズムは存在しない．したがって，均一なネットワークでは確率的アルゴリズムがいくつか提案されている[16),19)]．一方，準均一なネットワーク[13)]，識別子付きネットワーク[2)]では，任意の形状のネットワークで相互排除問題を解くことができる.

文献13)などでは，1個の特別なプロセッサが存在するネットワークを均一なネットワークと呼んでいるが，本稿では，すべてのプロセッサが同一の状態機械であるネットワークと陽に区別するために，準均一という用語を導入している.

ネットワーク状況（network configuration）は $c=(s_0, s_1, \cdots, s_{n-1})$ で表される．ここで，$s_i(0\leq i\leq n-1)$ はネットワーク状況 c でのプロセッサ p_i の状態を表す $(s_i\in S_i)$.

Aを任意のアルゴリズム，（Aによって決まる）任意のネットワーク状況を c，プロセッサ集合を P, P の任意の部分集合を Q とする．Q に属するすべてのプロセッサが（Aによって決まる）状態遷移関数に従って同時に状態を変えると，ネットワーク状況が c' になるとする．このことを $c\rightarrow(Q, A)c'$ と表す．つまり，$c=(s_0, s_1, \cdots, s_{n-1})$, $c'=(s'_0, s'_1, \cdots, s'_{n-1})$ とすると，$c\rightarrow(Q, A)c'$ のとき，任意の $p_i\notin Q$ に対して $s'_i=s_i$ が成り立ち，任意の $p_i\in Q$ に対して，$s'_i=T_i(s_i, s''_1, s''_2, \cdots, s''_d)$ が成り立つ．ここで，d は p_i の隣接プロセッサの数を表し，$p_i[j]=p_k$ なら $s''_j=s_k$ である.

プロセッサの部分集合の無限系列をスケジュールと呼ぶ．A を分散アルゴリズム，c_0 を（A によって決まる）ネットワーク状況，$T=Q_0, Q_1, Q_2, \cdots$ を任意のスケジュールとする．このとき，ネットワーク状況の無限系列 $E=c_0, c_1, c_2, \cdots$ が，各 $i(0\leq i)$ について $c_i\rightarrow(Q_i, A)c_{i+1}$ を満たすならば，E を「初期状況 c_0，スケジュール T に対するアルゴリズム A の実行」と呼ぶ．スケジュール T に，ネットワークのすべてのプロセッサが無限回現れるとき，T は公平（fair）であるという．また，実行 E のスケジュール T が公平なとき，実行 E は公平であるという.

許されるスケジュールの違いにより，ネットワークモデルは次のように分類できる.

1. Cデーモン（Central daemon）：各 $i(0\leq i)$ に対して $|Q_i|=1$ が成り立つスケジュール $T=Q_0, Q_1, Q_2, \cdots$ のみを考えるモデル．つまり，同時に一つのプロセッサしか動作しないという制限を加えたネットワークモデルである.

2. Dデーモン（Distributed daemon）：任意のスケジュールを考えるモデル．つまり，同時に複数のプロセッサが動作することを許すネットワークモデルである.

レジスタ通信モデルでは，さらにプロセッサの原子動作の大きさをも考慮して，次のように3種類のデーモンが考えられる.

1. Cデーモン：「同時に一つのプロセッサしか動作しない」かつ「1原子動作ですべての隣接レジスタから情報を読み込み，自分の状態を変化させ，必要があればすべての隣接レジスタに情報を書き込める」.

2. Dデーモン：「同時に複数のプロセッサが動作できる」かつ「1原子動作ですべての隣接レジスタから情報を読み込み，自分の状態を変化させ，必要があればすべての隣接レジスタに情報を書き込める」.

3. R/Wデーモン（Read/Write daemon）：「同時に一つのプロセッサしか動作しない」かつ「1原子動作では，一つの隣接レジスタからの読込みあるいは書出しと，それに引き続く状態遷移ができる」.

定義より，CデーモンはDデーモンの特殊な場合とみなせるので，DデーモンはCデーモンよりも弱いモデルである．また，レジスタ通信モデルの場合，R/Wデーモンの下で問題を解く自己安定アルゴリズムが，Cデーモン，Dデーモンの下でも問題を解くことが示せる[13)]．したがって，R/WデーモンはCデーモン，Dデーモンよりも弱いモデルである．また，Cデーモンの下では解けるが，Dデーモンの下では解けない問題が存在する[26)]．このことにより，DデーモンはCデーモンよりも真に弱いモデルである.

2.2 自己安定

自己安定アルゴリズムとは，任意のネットワーク状況からアルゴリズムの実行を始めても問題を

解くことができるアルゴリズムである．そこで，まず問題の解を得るということから定義する．生成木構成問題やリーダ選択問題では，ネットワークの生成木や一つのリーダを求めるとネットワークは定常状態（静止状態）になる．このような問題の場合は，ネットワークの解状況が満たす条件を記述すればよい．たとえば生成木構成問題なら，解状況ではネットワーク上の求めた生成木での隣接プロセッサを各プロセッサが認識している状況である．しかし相互排除問題では，特権 (privilege) をもつプロセッサが高々一つであり，かつすべてのプロセッサが無限回特権をももつような実行（ネットワーク状況の無限系列）を正しい実行と定義する必要がある．そこで本稿では，正しい実行が満たすべき条件を記述することによって，問題を定義する．

LS を，定義された正しい実行 (legal sequence) の集合とする．任意のネットワーク状況 c_0 で始まるアルゴリズム A の任意の公平な実行 $E=c_0, c_1, c_2, \cdots$ に対して，ある j が存在し，$E_j=c_j, c_{j+1}, \cdots$ が LS に属するとき，A は LS に関して自己安定であるという．またこのとき，正しい実行 E_j 中の各ネットワーク状況 $c_j, c_{j+1}, \cdots$ を，正当な状況 (legitimate state) と呼ぶ．

3. 相互排除問題のための自己安定アルゴリズム

Dijkstra は，方向感覚付きのリングネットワークで相互排除問題に対する準均一な決定性自己安定アルゴリズムを三つ提案している[9]．それぞれ，プロセッサ（状態機械）の状態数K（Kはn以上の任意の定数），状態数 4，状態数 3 のアルゴリズムである．これらのアルゴリズムでは状態通信モデルを用いており，Cデーモンの下で動作する．まず，状態数Kのアルゴリズムを紹介する．

なお，以下のアルゴリズムでは，各プロセッサの動作を，条件部と条件部が成立したときの動作で記述する．そして，条件部が真であり，動作可能（状態遷移可能）なプロセッサを特権をもつプロセッサとみなす．

3.1 状態数 K のアルゴリズム

[アルゴリズム 1]

各プロセッサ $p_i(0\leq i\leq n-1)$ は，以下のアルゴリズムを実行する．ただし，アルゴリズム中の

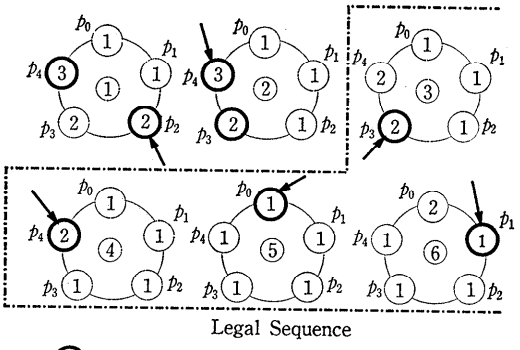


図-1 An example of executing K-state algorithm

s_i はプロセッサ p_i の状態を表す．

p_0 :

$s_{n-1}=s_0 \Rightarrow s_0:=(s_0+1) \bmod K$

$p_i(1\leq i\leq n-1)$:

$s_i\neq s_{i-1} \Rightarrow s_i:=s_{i-1}$

[アルゴリズム 1 の実行例]

アルゴリズム 1 の実行例を**図-1** に示す．図-1 において，3 番目のネットワーク状況以降は特権をもつプロセッサが一つだけなので，正当な状況である．

アルゴリズム 1 が自己安定相互排除アルゴリズムであることを示すには，以下のことを示せばよい．

1. 特権をもつプロセッサはいつかは一つになる（収束性）．

2. 特権をもつプロセッサが一つになると，それ以降は，特権をもつプロセッサは常に一つのまま（閉包性）．

3. すべてのプロセッサは何回でも特権をもつ（公平性）．

アルゴリズム 1 において，閉包性が成り立つことは容易に示せる．また，特権をもつプロセッサが常に一つ以上存在するのは明らかである．なぜならすべてのプロセッサの状態が同じなら，p_0 が特権をもち，異なる状態のプロセッサが存在すれば，必ず p_0 以外に特権をもつプロセッサが存在するからである．また，p_i が動作すると p_i の条件部は偽になるので，p_{i-1} ($i=0$ の場合は p_{n-1}) が動作しない限り，p_i は次の動作をできない．このことと，特権をもつプロセッサが常に一つ以上存在することから，すべてのプロセッサが何回でも特権をもつこと（公平性）がいえる．初期状況

での p_0 の状態を a とする．p_0 が n 回動作した直後を考えると，$p_{n-i}(1\leq i\leq n-1)$ は $n-i$ 回以上動作したことになり，ネットワーク状況は $(a+n, a+x_1, \cdots, a+x_{n-1}) \bmod K$ (ただし，$n>x_1\geq x_2\geq\cdots\geq x_{n-1}\geq 0$) となる．次に p_0 が特権をもつのはネットワーク状況が $(a+n, a+n, \cdots, a+n)$ になったときであり，このとき，特権をもつプロセッサは p_0 だけである．つまり，特権をもつプロセッサはいつかは一つになる．つまり，収束性も成り立つ．

アルゴリズム1では，p_i は自分と p_{i-1} の状態だけから動作を決定しているので，一方向リングでも実行可能である．

3.2 状態数3のアルゴリズム

[アルゴリズム2]

各プロセッサ $p_i(0\leq i\leq n-1)$ の状態 s_i は 0, 1, 2 のいずれかで表される．各プロセッサの状態遷移を以下に示す．ただし，アルゴリズムに現れる加算はすべて3を法とする加算である．

p_0 :

$s_0+1=s_1\Rightarrow$

$s_0{}'=s_0+2$

p_{n-1} :

$s_{n-2}=S_0$ and $s_{n-1}\neq s_0+1\Rightarrow$

$s_{n-1}{}'=s_0+1$

p_i $(1\leq i\leq n-2)$:

$s_{i-1}=s_i+1$ or $s_i+1=s_{i+1}\Rightarrow$

$s_i{}'=s_i+1$

[アルゴリズム2の実行例]

アルゴリズム2の実行例を**図-2**に示す．図-2において，5番目のネットワーク状況以降は特権をもつプロセッサが一つだけなので，正当な状況

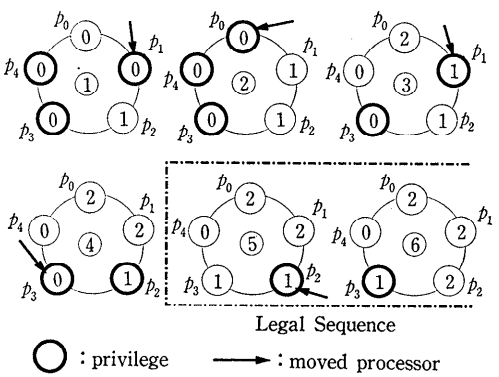


図-2 An example of executing 3-state algorithm

である．

アルゴリズム2が正しいことを理解することは容易でない．実際，Dijkstra はこのアルゴリズムを提案してから12年後に，その正当性を文献11)で示している．その証明に使われた手法は非常にエレガントであり，ここで簡単に紹介する．そのほかにいくつかの文献でも，同様の手法で自己安定アルゴリズムの正当性が示されている．

[アルゴリズム2の正当性]

ネットワーク状況 $c=(s_0, s_1, \cdots, s_{n-1})$ に対し，関数 f を次のように定義する．

$$f(c)=|L(c)|+2\times|R(c)|$$

ここで $L(c), R(c)$ は，次のように定義される．

$$L(c)=\{s_i \mid s_i+1=s_{i+1}(\bmod 3),\ i=0, 1, \cdots, n-2\}$$

$$R(c)=\{s_j \mid s_{j-1}=s_j+1(\bmod 3),\ j=1, 2, \cdots, n-1\}$$

つまり，$s_i\in L(c)(0\leq i\leq n-2)$ なら，ネットワーク状況 c において，p_i が特権をもつ．また，$s_k\in R(c)(1\leq k\leq n-2)$ なら，ネットワーク状況 c において，p_k が特権をもつ $(s_{n-1}\in R(c)$ のときに，p_{n-1} が特権をもつかどうかは，p_0 の状態による)．

アルゴリズム2より，1動作（1状態遷移）における関数 f の変化量を $\Delta f, |L(c)|, |R(c)|$ の変化量をそれぞれ $\Delta L(c), \Delta R(c)$ とすると，次のような変化が考えられる．

〈p_0 の動作による変化〉

$s_0=s_1-1\rightarrow s_0=s_1+1$:

$$\Delta L(c)=-1, \Delta R(c)=+1, \Delta f=+1 \quad (1)$$

〈$p_i(1\leq i\leq n-2)$ の動作による変化〉

$s_{i-1}-1=s_i=s_{i+1}\rightarrow s_{i-1}=s_i=s_{i+1}+1$:

$$\Delta L(c)=\Delta R(c)=\Delta f=0 \quad (2)$$

$s_{i-1}=s_i=s_{i+1}-1\rightarrow s_{i-1}+1=s_i=s_{i+1}$:

$$\Delta L(c)=\Delta R(c)=\Delta f=0 \quad (3)$$

$s_{i-1}-1=s_i=s_{i+1}-1\rightarrow s_{i-1}=s_i=s_{i+1}$:

$$\Delta L(c)=-1, \Delta R(c)=-1, \Delta f=-3 \quad (4)$$

$s_{i-1}-1=s_i=s_{i+1}+1\rightarrow s_{i-1}=s_i=s_{i+1}-1$:

$$\Delta L(c)=+1, \Delta R(c)=-2, \Delta f=-3 \quad (5)$$

$s_{i-1}+1=s_i=s_{i+1}-1\rightarrow s_{i-1}-1=s_i=s_{i+1}$:

$$\Delta L(c)=-2, \Delta R(c)=+1, \Delta f=0 \quad (6)$$

〈p_{n-1} の動作による変化〉

$s_{n-2}-1=s_{n-1}\rightarrow s_{n-2}+1=s_{n-1}$:

$$\Delta L(c)=+1, \Delta R(c)=-1, \Delta f=-1 \quad (7)$$

$s_{n-2}=s_{n-1}\rightarrow s_{n-2}+1=s_{n-1}$:

$$\Delta L(c)=+1, \Delta R(c)=0, \Delta f=+1 \quad (8)$$

$f(c)=0$ なら，$s_0=s_1=\cdots=s_{n-1}$ が成り立つ．このとき，p_{n-1} だけが特権をもつ．$f(c)=1$ なら，ある $i(0\leq i\leq n-2)$ が存在して，任意の $j,k(j\leq i<k)$ に対して，$s_j=s_k-1$ が成り立つ．このとき，p_i だけが特権をもつ．したがって，$f(c)\leq 1$ ならば，ネットワーク状況 c において特権をもつプロセッサは一つだけである．

特権をもつプロセッサが一つになると，(2)，(3)の動作によって，特権をもつプロセッサは p_1 から p_{n-2} の間を移動し，(1)，(7)によって p_0, p_{n-1} で折り返す．すなわち，特権をもつプロセッサが一度一つになると，その後，特権をもつプロセッサは一つのまま（閉包性）であり，すべてのプロセッサが無限回特権をもつ（公平性）．

したがって，$f(c)>1$ なら有限回の動作で $f(c)\leq 1$ となることを示せば，収束性を示したことになり，アルゴリズム2の正当性が証明できる．以下では，$f(c)>1$ なら，有限回の動作で $f(c)$ が必ず減少することを簡単に説明する．

まず，$f(c)>1$ なら，ネットワーク状況 c において特権をもつプロセッサが存在することを示す．先に述べたように，$|L(c)|\geq 1$ または $|R(c)|\geq 2$ のときは，特権をもつプロセッサが必ず存在する．また，$|L(c)|=0$ かつ $|R(c)|=1$ のとき $(s_j\in R(c)(1\leq j\leq n-1)$ とする)，ネットワーク状況 c において p_j が特権をもつことが分かる．したがって，$f(c)>1$ なら特権をもつプロセッサが存在する．

$f(c)$ が増加するのは(1)，(8)が実行されたときである．(8)の実行の後再び(8)が実行されるまでに，(1)が実行されなければならない．また，無限の実行を考えると，(1)が無限回実行されることを示せる．(1)の実行後再び(1)が実行されるまでに，特権が一つになっているか，あるいは，(4)または(5)が実行されることを示せる．したがって，特権が一つにならない限り，(1)，(8)が高々1回ずつ実行される間に，(4)または(5)が実行され，この間に f の値は少なくとも1減少することがいえる．したがって，$f(c)>1$ なら有限回の動作で $f(c)\leq 1$ となり，このとき，特権をもつプロセッサは一つだけである．したがって，収束性が成り立つ．

方向感覚付きのリングネットワークにおいて，相互排除問題を解く状態数2の自己安定アルゴリズムが存在しないことが知られている[24]．したがって，アルゴリズム2は，リングネットワークで相互排除問題を解く状態数最小の自己安定アルゴリズムである．また，Dijkstra は，方向感覚付きのリングネットワークで相互排除問題を解く状態数4のアルゴリズムも示している[9]．そのアルゴリズムは，方向感覚付きのリスト状ネットワーク（一次元アレイ）でも相互排除問題を解くことができる．（アルゴリズム2では，p_0 以外のすべてのプロセッサは両隣のプロセッサの状態に依存して次の状態を決めているので，リスト状ネットワークで動作しない．）方向感覚付きのリスト状ネットワークにおいて，相互排除問題を解く状態数3のアルゴリズムが存在しないことが知られている[24]．したがってこのアルゴリズムは，リスト状ネットワークで相互排除問題を解く状態数最小の自己安定アルゴリズムである．

4. 既知の主な結果

1. 準均一な方向感覚付きリングネットワーク上

- 文献 9)：E. W. Dijkstra

初めて自己安定アルゴリズムの概念を導入した論文．Cデーモンで相互排除問題を解く，それぞれプロセッサの状態数が $K, 4, 3$ である三つの自己安定アルゴリズムを示した．

- 文献 6)：G. M. Brown ら

Dデーモンの下で，方向感覚付きリストで相互排除問題を解くアルゴリズムを示している．また，Cデーモンの下で動作するアルゴリズムがDデーモンの下でも動作するための十分条件として，相互非干渉 (non-interfering) という条件を示した．

2. 均一な方向感覚付きリングネットワーク上

- 文献 10)：E. W. Dijkstra

均一なリングネットワークのサイズ（プロセッサ数）n が合成数ならば，その上での決定性自己安定アルゴリズムが存在しないことを証明している．

- 文献 7)：J. E. Burns ら

サイズが素数の均一なリングネットワーク上では，Dデーモンで決定性自己安定アルゴリズムが存在することを示した．

● 文献 16) : T. Herman

任意のサイズの同期式リングネットワーク上で相互排除問題を解く，コイントスを利用したランダムな自己安定アルゴリズムを示した．リングネットワークのネットワーク状況の対称性を打破するために確率が利用されている．

● 文献 19) : A. Israeli ら

Dデーモンの下で，均一な任意のネットワークおよびサイズが変化する動的リングネットワーク上で，相互排除問題を解く自己安定アルゴリズムを示した．

3. 方向感覚のないリングネットワーク上

● 文献 13) : S. Dolev ら

方向感覚のないリングネットワーク上では，状態通信モデルで相互排除問題を解く自己安定アルゴリズムがないことを証明している．

4. その他の問題

(a) 生成木構成問題

● 文献 13) : S. Dolev ら

R/W デーモンを初めて紹介した論文．以下の三つのアルゴリズムを提案．

i．任意のグラフ上で生成木を構成する．

ii．木構造ネットワーク上で相互排除問題を解く．

iii．任意のグラフ上で相互排除問題を解く．

特にiiiは，i，iiのアルゴリズムの公平な合成 (fair composition) を用いて解いている．

● 文献 28) : 小谷ら

R/W デーモンの下で，識別子ありの任意のネットワーク上で重み最小生成木を構成する自己安定アルゴリズムを示した．

● 文献 17) : S. Huang ら

Dデーモンの下で，任意の準均一なネットワーク上で幅優先木を構成する自己安定アルゴリズムを示した．

(b) リングの方向付け

● 文献 18) : A. Israeli ら

Dデーモンの下で，任意サイズの均一なリングネットワーク上でリングの方向付け問題を解く，ランダムな自己安定アルゴリズムを示した．また，以下の不可能性を示した．

i．状態通信モデルの偶数サイズの準均一なリングネットワーク上で，Cデーモンの下でリングの方向付け問題を解く決定性自己安定アルゴリズムは存在しない．

ii．状態通信モデルの奇数サイズの準均一なリングネットワーク上で，Dデーモンの下でリングの方向付け問題を解く決定性自己安定アルゴリズムは存在しない．

iii．レジスタ通信モデルの偶数サイズの均一なリングネットワーク上で，Dデーモンの下でリングの方向付け問題を解く決定性自己安定アルゴリズムは存在しない．

● 文献 26) : 片山ら

Cデーモンの下で，レジスタ通信モデルの任意サイズのリングネットワーク上でリングの方向付け問題を解く決定性自己安定アルゴリズム，および，R/W デーモンの下で，奇数サイズのリングネットワークのリングの方向付け問題を解く決定性自己安定アルゴリズムの二つを提案している．また，CデーモンとDデーモンの真の能力差を初めて示している．

(c) 時計合わせ[22]，リーダ選択[14]など．

5. おわりに

本稿では，自己安定アルゴリズムのモデルと基本的な考え方，過去知られている主な結果を紹介した．自己安定アルゴリズムは，故障耐性のある分散アルゴリズムの一分野として研究されてきたが，その歴史は比較的新しく，通常の分散アルゴリズムでは解かれているが，自己安定アルゴリズムで解かれていない問題も多い．

また，アルゴリズムの評価基準も，通常の分散アルゴリズムと同様のもののほかに，自己安定アルゴリズムに特有なものも考えられるだろう．たとえば，安定した状態（解状況）に至った後に故障が起こった場合，ネットワーク上で故障箇所からどの程度の範囲に影響が及ぼされるのかということも，評価として有効ではないかと思われる．

謝辞 本稿執筆の機会を与えていただいた編集委員会の委員各位に深謝いたします．日頃ご指導いただく大阪大学都倉信樹教授に感謝いたします．また，貴重なご意見をいただいた真鍋氏ならびに閲読者の方々に感謝いたします．

参考文献

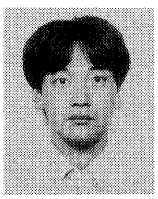
1) Afek, Y. and Kutten, S.: Memory-Efficient Self Stabilizing Protocols for General Networks, in *Proc. 4th International Workshop on Distributed Algorithms* (*LNCS 486*), pp. 15-28(1990).

2) Afek, Y., Kutten, S. and Yung, M.: Memory-Efficient Self-Stabilizing Protocols for General Networks, in *Proc. 4th International Workshop on Distributed Algorithms* (*LNCS 486*), pp. 15-28 (1990).
3) Anagnostou, E. and El-Yaniv, R.: More on the Power of Random Walks: Uniform Self-Stabilizing Randomized Algorithms (Preliminary Report), in *Proc. of 5th International Workshop on Distributed Algorithms* (*LNCS 579*), pp. 31-51 (1991).
4) Anagnostou, E., El-Yaniv, R. and Hadzilacos, V.: Memory Adaptive Self-Stabilizing Protocols (Extended Abstract), in *Proc. 6th International Workshop on Distributed Algorithms* (*LNCS 647*), pp. 203-220 (1992).
5) Awerbuch, B., Patt-Shamir, B. and Varghesee, G.: Self-Stabilization By Local Checking and Correction, in *32nd Symposium on Foundations of Computer Science*, pp. 268-277 (1991).
6) Brown, G.M., Gouda, M.G. and Wu, C.: Token Systems That Self-Stabilize, *in IEEE Trans. Comp.* 38, 6, pp. 845-852 (1989).
7) Burns, J.E. and Pachl, J.: Uniform Self-Stabilizing, Rings, in *ACM Trans. Prog. Lang. Syst.* Vol. 1, No. 2, pp. 330-344 (1989).
8) Chen, N.: A Self-Stabilizing Algorithm for Constructing Spanning Trees, in *Inf. Process. Lett. 39*, pp. 147-151 (1991).
9) Dijkstra, E.W.: Self-Stabilizing Systems in Spite of Distributed Control, in *Comm. of the ACM*, Vol. 17, No. 11, pp. 643-644 (1974).
10) Dijkstra, E.W.: Self-Stabilization in Spite of Distributed Control, in *Reprinted in Selected Writing on Computing: A Personal Perspective*, Springer-Verlag, Berlin, pp. 41-46 (1982).
11) Dijkstra, E.W.: A Related Proof of Self-Stabilization, in *Distributed Computing 1*, pp. 5-6 (1986).
12) Dolev, S. and Israeli, A.: Resource Bounds for Self-Stabilizing Message Driven Protocols, in *Proc. of 10th annual ACM Symposium on Principles of Distributed Computing*, pp. 19-21 (1991).
13) Dolev, S., Israeli, A. and Moran, S.: Self Stabilization of Dynamic Systems Assuming Only Read/Write Atomicity, in *Proc. 9th Symposium on Principles of Distributed Computing*, pp. 103-117 (1990).
14) Dolev, S., Israeli, A. and Moran, S.: Uniform Dynamic Self-Stabilizing Leader Election (Extended Abstract), in *Proc. of 5th International Workshop on Distributed Algorithms* (*LNCS 579*), pp. 31-51 (1991).
15) Ghosh, S.: Binary Self-Stabilization in Distributed Systems, in *Inf. Process. Lett. 40*, pp. 153-159 (1991).
16) Herman, T.: PROBABILISTIC SELF-STABILIZATION, in *Inf. Process. Lett. 35*, pp. 63-67 (1990).
17) Huang, S. and Chen, N.: A Self-Stabilizing Algorithm for Constructing Breadth-First Trees, in *Inf. Process. Lett. 41*, pp. 109-117 (1992).
18) Israeli, A. and Jalfon, M.: Self-Stabilizing Ring Orientation, in *Proc. 4th International Workshop on Distributed Algorithms* (*LNCS 486*), pp. 1-13 (1990).
19) Israeli, A. and Jalfon, M.: Token Management Schemes and Random Walks Yield Self Stabilizing Mutual Exclusion, in *Proc. of 9th Symposium on Principle of Distributed Computing*, pp. 119-131 (1990).
20) Katz, S. and Perry, K.J.: Self-Stabilizing Extensions for Message-Passing Systems, in *Proc. of 9th Symposium on Principle of Distributed Computing*, pp. 91-103 (1990).
21) Lamport, L.: Solved Problems, Unsolved Problems and Non-Problems in Concurrency, in *Proc. of 3rd Symposium on Principles of Distributed Computing*, pp. 1-11 (1984).
22) Lu, M., Zhang, D. and Murata, T.: Analysis of Self-Stabilizing Clock Synchronization by Means of Stochastic Petri Nets, in *IEEE Trans. Comput.*, Vol. 39, No. 5, pp. 597-604 (1990).
23) Schneider, M.: Self-Stabilization, in *ACM Comput. Surv.*, Vol. 25, No. 1, pp. 45-67 (1993).
24) Tchuente, M.: *Parallel Computation on Regular Arrays*, Section 2.3, pp. 33-48, Manchester University Press (1991).
25) 梅本成俊，角川裕次，山下雅史：自己安定相互排除アルゴリズムの実験的評価とその改良，信学技報 COMP 93-16 (1993).
26) 片山喜章，増澤利光，都倉信樹：リングの方向付け問題を解く自己安定アルゴリズム，情報処理学会，アルゴリズム研究会 (92-AL-25-8) (1992).
27) 角川裕次，山下雅史：Self-Stabilizing 2-Mutual Exclusion on Bidirectional Rings, 冬の LA シンポジウム (1993).
28) 小谷晃一，片山喜章，増澤利光，都倉信樹：ネットワークの重み最小生成木を構成する自己安定分散アルゴリズムについて，電子情報通信学会技術研究報告，COMP 92-6 (1992).
29) 西川直樹，増澤利光，都倉信樹：相互排除問題を解く均一な自己安定分散アルゴリズム，電子情報通信学会論文誌 D-I Vol. J75-D-I, No. 4, pp. 201-209 (1992).

(平成5年6月8日受付)

増澤 利光（正会員）

昭和34年生．昭和57年大阪大学基礎工学部情報工学科卒業．昭和62年同大学院博士課程修了．工学博士．同年同大学情報処理教育センター助手．現在，同大基礎工学部助手（情報工学）．平成5年度コーネル大学計算機科学科客員助教授．分散アルゴリズム，並列アルゴリズムに関する研究に従事．ACM，IEEE，電子情報通信学会各会員．



片山 喜章

昭和41年生．平成2年大阪大学基礎工学部情報工学科卒業．平成4年同大学院博士課程前期修了．工学修士．現在，同大学院博士課程後期在学中．分散アルゴリズムに関する研究に従事．電子情報通信学会会員．