プログラミング 11−11
(1997. 1. 23)

安定後の1故障を考慮したリングでの自己安定相互排除プロトコル

上田英一郎† 片山喜章†† 増澤利光† 藤原秀雄†

奈良先端科学技術大学院大学

†情報科学研究科 ††情報科学センター

〒630-01 奈良県生駒市高山町 8916-5
e-mail:eiitir-u@is.aist-nara.ac.jp

あらまし

自己安定プロトコルとは，任意のネットワーク状況から実行を開始しても解を求めて安定するプロトコルである．この性質から，自己安定プロトコルは任意の一時故障に耐性がある．自己安定相互排除プロトコルが，1故障状況（正当な状況から，1プロセスの一時故障により到達する状況）から実行を開始した場合に，特権を持つ非故障プロセスが高々1つであることを保証するとき，このプロトコルを強安定相互排除プロトコルという．

本稿では，1方向リング上の巡回ラウンド数 $\lfloor \frac{n}{2} \rfloor + 1$ の強安定相互排除プロトコルを提案する．また巡回ラウンド数 $\lfloor \frac{n}{2} \rfloor$ の強安定相互排除プロトコルは存在しないことを示し，本稿で提案するプロトコルが巡回ラウンド数に関して最適であることを示す．

キーワード 分散アルゴリズム，自己安定，強安定，相互排除，1方向リング

A superstabilizing protocol for mutual exclusion on rings

Eiichirou UEDA† Yoshiaki KATAYAMA†† Toshimitsu MASUZAWA† Hideo FUJIWARA†

†Graduate School of Information Science, Nara Institute of Science and Technology

††Information Technology Center, Nara Institute of Science and Tchonology

8916-5, Takayamacho, Ikoma, Nara 630-01
e-mail:eiitir-u@is.aist-nara.ac.jp

Abstract

A *self-stabilizing* protocol is a protocol that achieves its intended behavior regardless of the initial configuration. Thus, a self-stabilizing protocol is resilient to any number and any type of transient faults. A self-stabilizing mutual exclusion protocol is called a *superstabilizing* mutual exclusion protocol if it can recover from any 1-faulty configuration while satisfying that at most one non-faulty process has privilege, where 1-faulty configuration is a configuration reachable from a legitimate configuration by a transient fault of a single process.

This paper presents a superstabilizing mutual exclusion protocol with $(\lfloor \frac{n}{2} \rfloor + 1)$-latency on unidirectional rings. This paper also shows that there is no superstabilizing mutual exclusion protocol with $\lfloor \frac{n}{2} \rfloor$-latency. Thus, the proposed protocol is latency-optimal.

key words distributed algorithm, self-stabilization, superstabilization, mutual exclusion,unidirectional ring

1 まえがき

ネットワークで相互接続されたプロセスから構成される分散システムにおいて，分散システムのプロセスが協調して問題を解くためのプロトコル（分散アルゴリズム）の研究が盛んに行われている．通常のプロトコルでは，プロトコル実行開始時の分散システムの大域状況が，あらかじめ決められた初期状況（各プロセスの状態があらかじめ決められた初期状態）であると仮定する．

これに対し，自己安定プロトコル（self-stabilizing protocol）は，プロトコル実行開始時の分散システムの大域状況について何も仮定しない．つまり，自己安定プロトコルは，任意の初期状況から実行を開始しても，問題の解を求めて安定するプロトコルである．この性質から，自己安定プロトコルでは，プロセスの一時的な故障により分散システムがどのような大域状況に陥っても，故障したプロセスが復旧すれば，分散システムの大域状況が自動的に正常な状況に戻る．つまり，長期に渡ってネットワークの状況を安定に保ち，一時故障に柔軟に対応することの求められる分散システムを動作させる場合に適している．自己安定プロトコルは1974年にDijkstra [1] によって初めて導入された概念であるが，一時故障に対する優れた故障耐性により，故障耐性のあるプロトコルとして注目され，特に近年，多くの研究が行われている．

分散システムで自己安定プロトコルを長期的に動作させる場合，分散システム中の多数のプロセスが同時に故障することは稀であり，安定状態からかけはなれた状態からの動作ではなく，安定状況から少し状態変化（小変動）した状況からのプロトコルの動作が重要である．しかし，従来の自己安定プロトコルは，小変動に対しても，分散システムが正当な状況にいずれ復旧することは保証しているが，その復旧過程については何も要求していない．そのため，ごく少数のプロセスが一時故障した場合でも，その影響が分散システム全体に及び，分散システムの動作がしばらく混乱する場合もある．例えば，Dijkstra [1] は1方向リングでの相互排除を実現する自己安定プロトコルを提案しているが，このプロトコルでは，1プロセスの故障によって特権を持つプロセスが2個になった場合，これら2つの特権が移動し，分散システム中に特権を持つプロセスが2個存在する状況がしばらく続くことがある．

文献 [2, 3, 4] は，小変動に対してすぐれた特性を持つ自己安定プロトコルを提案している．特に，文献 [2, 3] は，小変動が生じた場合に，再び正常な状況に復旧するまでの復旧過程がある制約を満たす強安定プロトコル（superstabilizing protocol）を提案している．例えば，Herman [3] は，1方向リングでの相互排除問題に対し，1故障状況（正常な状況から，1プロセスが故障により状態を変えることによって到達する状況）からの実行において，故障プロセスが一時的に特権を持つ場合を除き，特権を持つプロセスが高々1つであることを保証している．特権を持たないプロセスの状態が，故障によって，特権を持つ状態に変化してしまうことは避けられないので，Hermanの制約は相互排除に関して最もきつい制約の1つと考えることができる．

Hermanの強安定相互排除プロトコル [3] は，Dijkstraの自己安定相互排除プロトコル [1] を拡張したものである．Dijkstraの自己安定相互排除プロトコルの基本方針は，1方向リング上でトークンを巡回させ，トークンを持つプロセスが特権を持つとみなすことである．このため，1プロセスの故障により偽の特権（トークン）が発生すると，この偽の特権も移動するために，2プロセスが特権を持つ状況がしばらく続いてしまう．Hermanの強安定相互排除プロトコルでは偽の特権の移動を防ぐために，主トークン，副トークンという2種類のトークンを利用し，主トークンと副トークンを同時に持つプロセスが特権を持つこととしている．しかし，主トークンは副トークンがリングを1周するごとに1つ進むだけであり，正常な状況において，すべてのプロセスが特権を持つまでには，副トークンがリングをn周（nはプロセス数）しなければならない．このため，Hermanの強安定相互排除プロトコルの巡回ラウンド数（正常な状況での相互排除プロトコルの性能を評価するための評価尺度）はnである．また，Herman [3] は，巡回ラウンド数1の強安定相互排除プロトコルは存在しないことを証明している．

本稿では，巡回ラウンド数$\lfloor \frac{n}{2} \rfloor + 1$の強安定相互排除プロトコルを提案する．この強安定プロトコルは，Hermanの強安定プロトコルに比べ，巡回ラウンド数を改善しているだけでなく，各プロセスの状態空間の大きさも$O(n^{n+1})$から$O(n^2)$に改善している．さらに，本稿では，巡回ラウンド数$\lfloor \frac{n}{2} \rfloor$の強安定相互排除プロトコルは存在しないことを証明し，本稿で提案する巡回ラウンド数$\lfloor \frac{n}{2} \rfloor + 1$の強安定相互排除プロトコルが，巡回ラウンド数に関して最適であることを示す．

2 諸定義

2.1 分散システム

分散システムNは，プロセスの集合Pと1方向通信リンクの集合Lの2項組(P, L)で定義される．本稿では，n個のプロセスが1方向通信リンクでリング状に接続された分散システムのみを対象とし，プロセス集合を$P = \{p_0, p_1, \ldots p_{n-1}\}$，1方向通信リンクの集合を$L = \{(p_i, p_{(i+1) \bmod n}) \mid 0 \leq i \leq n-1\}$とする．以下では，プロセスの添字から$\bmod n$を省略し，例えば，$p_{(i+1) \bmod n}$を単に$p_{i+1}$と略記する．

プロセスを状態機械としてモデル化し，各プロセスp_iを状態集合S_iと状態遷移関数δ_iの2項組(S_i, δ_i)で表す．1方向リング状の分散システムでは，プロセスp_iはプロセスp_{i-1}から情報を受けとることができる．このプロセス間通信のモデルとしては，メッセージによって情報を伝えるメッセージ通信モデル，共有レジスタを介して情報を伝えるレジスタ通信モデル，隣接プロセスの状態を知ることができる状態通信モデルなどがこれまでに考えられている．本稿では，状態通信モデルを採用し，プロセスp_iは，プロセスp_iとp_{i-1}の状態から次の状態を決定するものとする．つまり，プロセスp_iの状態遷移関数δ_iは，$\delta_i : S_i \times S_{i-1} \rightarrow S_i$である．ここで，$\delta_i(s, t) = s'$は，$p_i$，$p_{i-1}$の状態がそれぞれ$s, t$のときに$p_i$が動作すれば，

p_iの状態がs'に遷移することを意味している．

分散システム全体の大域状況を，各プロセスの状態で表す．つまり，可能なすべての大域状況の集合をCとすると，$C = S_0 \times S_1 \times S_2 \times \ldots \times S_{n-1}$である．以下では，分散システム全体の大域状況のことを，単に，**状況**とよぶ．

$c \in C$を任意の状況，$Q \subseteq P$をプロセスの任意の部分集合とする．Qに属するプロセスが同時に動作することにより，状況がcからc'に変化するとき，$c' = \Delta(c, Q)$と表す．つまり，$c = (s_0, s_1, \ldots s_{n-1})$，$c' = (s'_0, s'_1, \ldots s'_{n-1})$とすると，$p_i \in Q$ならば$s'_i = \delta_i(s_i, s_{i-1})$が成り立ち，$p_i \notin Q$ならば$s'_i = s_i$が成り立つ．

プロセスの空でない集合の無限系列を**スケジュール**と呼ぶ．c_0を状況，$\mathcal{Q} = Q_0, Q_1, \ldots$をスケジュールとする．このとき，状況の無限系列$E = c_0, c_1, c_2 \ldots$が$c_{i+1} = \Delta(c_i, Q_i)$ $(i \geq 0)$を満たすとき，Eを初期状況c_0，スケジュール$\mathcal{Q}$に対する**実行**と呼ぶ．つまり，$Q_0, Q_1, \ldots$に属するプロセスが順に動作するときの状況の変化を表す系列が実行Eである．

スケジュール$\mathcal{Q}$に分散システムのすべてのプロセスが無限回現れるとき，スケジュール$\mathcal{Q}$は**公平**であると言う．本稿では，公平なスケジュールに対する実行のみを考慮し，公平なスケジュールに対する実行を，単に，実行とよぶ．

2.2 相互排除問題

本稿では，相互排除問題を解く自己安定プロトコルについて考察する．1つのプロセスによって占有的に使用される資源を複数のプロセスで共有する場合，資源の使用権（**特権**とよぶ）を持つプロセスが同時に2個以上存在しないようにしなければならない．このような排他的な特権を実現する問題が**相互排除問題**である．本稿では，プロセスを状態機械としてモデル化しているが，プロセスp_iが特権を持つ状態を表すために，状態の部分集合$T_i \subset S_i$が定められているものとする．つまり，プロセスp_iの状態がT_iに属するとき，プロセスp_iが特権を持つという．

一般に，**分散問題**は，実行において各プロセスにどのような動作を要求するかを定める．つまり，分散問題は実行が満たすべき条件によって記述される．この条件を満たす実行をその分散問題の**正当な実行**とよぶ．相互排除問題の正当な実行は次のように定義される．

定義 1（**相互排除問題**）

$E = c_0, c_1, \ldots$を任意の実行とする．実行Eが次の2条件を満たすとき，実行Eは相互排除問題の正当な実行である．

1. 任意の状況c_i $(i \geq 0)$において，特権を持つプロセスは高々1つ．
2. 任意の状況c_i $(i \geq 0)$と任意のプロセスp_j $(0 \leq j \leq n-1)$に関して，p_jが特権を持つ状況c_k $(k \geq i)$が存在する．■

相互排除問題の条件1は排他的な特権を保証し，条件2は飢餓状態に陥るプロセスが存在しないことを保証している．

2.3 自己安定プロトコル

プロトコルは，分散システムのすべてのプロセスに協調作業させるために，各プロセスの動作を定めるものである．つまり，各プロセスp_iについて，その状態集合S_iと状態遷移関数δ_iを定める．本稿で提案するプロトコルでは，プロセスp_0以外のすべてのプロセスは同一の状態機械である．つまり，任意のi, j $(1 \leq i \leq j \leq n-1)$について，$S_i = S_j$かつ$\delta_i = \delta_j$が成り立つ．

分散問題Πを解く**自己安定プロトコル**とは，初期状況にかかわらず，Πで要求される動作をいずれ実行するプロトコルであり，次のように定義される．

定義 2（**自己安定プロトコル**）

プロトコル$\mathcal{A}$の任意の状況c_0から始まる任意の実行$E = c_0, c_1, \ldots$において，次の条件を満たす状況c_iが現れるとき，プロトコル$\mathcal{A}$は分散問題Πを解く自己安定プロトコルである．

- 実行Eの状況c_iから始まる接尾部$E' = c_i, c_{i+1}, \ldots$は問題$\Pi$の正当な実行である．

特に，分散問題Πを解く自己安定プロトコル$\mathcal{A}$に対し，次の3条件を満たす状況の集合$\mathcal{L} \subset C$が存在するとき，プロトコル$\mathcal{A}$を$\mathcal{L}$に関する自己安定プロトコルという．また，$\mathcal{L}$に属する状況を**正当な状況**と言う．

1. 正当性：$c \in \mathcal{L}$なる任意の状況cから始まる任意の実行Eは問題Πの正当な実行である．
2. 到達可能性：任意の状況c_0から始まる任意の実行$E = c_0, c_1, \ldots$において，$c_i \in \mathcal{L}$なる状況c_iが現れる．
3. 閉包性：$c_0 \in \mathcal{L}$なる任意の状況c_0から始まる任意の実行を$E = c_0, c_1, \ldots$とする．このとき，実行Eに現れるすべての状況c_i $(i \geq 0)$は，$\mathcal{L}$に属する．つまり，プロトコル$\mathcal{A}$の実行が正当な状況に達すると，それ以降の任意の状況は正当な状況である．

■

本稿では，正当な状況に安定した後の1プロセスの一時故障からの復旧に関して，その実行がある制約を満たす自己安定プロトコルについて考察する．まず，正当な状況から1プロセスの一時故障によって起こりうる状況を定義する．

定義 3（**1故障状況**）

プロトコル$\mathcal{A}$を正当な状況集合$\mathcal{L}$に関する自己安定プロトコルとする．正当な状況$c \in \mathcal{L}$において，1つのプロセスの状態を任意の状態に変えることにより得られる状況c'（ただし，$c' \notin \mathcal{L}$とする）を**1故障状況**とよび，状態を変えたプロセスを**故障プロセス**とよぶ．つまり，1故障状況の集合$\mathcal{F}$は，次のように定義される．

$\mathcal{F} = \{c' \in C - \mathcal{L} \mid \exists c \in \mathcal{L}[diff(c, c') = 1]\}$

ここで，$diff(c, c')$は，状況cとc'において状態の異なるプロセス数を表す．■

次に，1故障状況から始まる実行が，ある制約を満たす自己安定プロトコルを定義する．

定義 4（**強安定プロトコル**）

プロトコル$\mathcal{A}$を正当な状況集合$\mathcal{L}$に関する自己安定プロトコルとする．任意の1故障状況cから始まる任意の

実行 Eが，実行に関する制約 $\mathcal{P}$を満たすとき，プロトコル $\mathcal{A}$は，$\mathcal{P}$に関して**強安定**であるという． ■

相互排除問題を解く自己安定プロトコルでは，1故障状況から始まる実行に現れる任意の状況においても，特権を持つプロセスが高々1つであることが望ましい．しかし，プロセス pが特権を持つ正当な状況において，p以外のプロセス qが一時故障のために特権を持つ状態に変化した場合，結果として生じる状況では，プロセス pと qが特権を持つ．このような場合には，なるべく早く，特権を持つプロセスが高々1つである状況に復帰することが望まれる．Herman [3] は，このような観点から，自己安定相互排除プロトコルの任意の1故障状況 c_0 から始まる実行 $E = c_0, c_1, \ldots$ が満たすべき制約 $\mathcal{P}$として，次の制約を提案している．

1. c_0において故障プロセスが特権を持たない場合：実行 Eに現れる任意の状況 c_i $(i \geq 0)$ において特権を持つプロセスは高々1つ．
2. c_0において故障プロセス（qとする）が特権を持つ場合：実行 Eのスケジュールを $\mathcal{Q} = Q_0, Q_1, \ldots$ とし，$k = \min\{j \mid q \in Q_j\}$ とする．つまり，c_0以降 c_kに至るまで故障プロセス qは動作せず，状況が c_kから c_{k+1}に変化するときに初めて qが動作したものとする．
   (1) c_k以前の任意の状況 c_i $(0 \leq i \leq k)$ において，故障プロセス q以外に特権を持つプロセスは高々1つである．(状況 c_kの定義より，状況 c_i において qは特権を持つ．従って，状況 c_iにおいて，特権を持つプロセスは2個存在するかもしれない．)
   (2) c_{k+1}以降の任意の状況 c_i $(i \geq k+1)$ において，特権を持つプロセスは（qを含め）高々1つである．

上記の制約は，強安定相互排除プロトコルの正当な状況において，1プロセスで一時故障が生じても，故障プロセスが一時的に特権を持つ場合を除き，特権を持つプロセスが高々1つであることを保証している．本稿では，上記の制約に関して強安定な自己安定相互排除プロトコルについて考察する．以下では，上記の制約に関して強安定な自己安定相互排除プロトコルを，単に，**強安定相互排除プロトコル**とよぶ．

Herman [3] は，自己安定相互排除プロトコルの評価尺度として，**巡回ラウンド数**を提案している．巡回ラウンド数は，自己安定相互排除プロトコルの正当な状況から始まる実行において，すべてのプロセスが特権を1回以上持つまでに必要な時間の目安となる評価尺度である．つまり，巡回ラウンド数は，自己安定相互排除プロトコルが正当な状況で動作している場合の性能を表す評価尺度である．自己安定プロトコルは，一時故障に対する故障耐性を持つプロトコルとして注目されているが，分散システムで自己安定プロトコルを長期に実行せさている状況では，ほとんどの期間は正当な状況で動作しており，プロセスの一時故障が生じたときのみ，正当な状況でなくなる．従って，正当な状況で動作している場合の性能は，自己安定プロトコルの重要な評価尺度の1つである．以下に，巡回ラウンド数を定義するが，まず，その定義に必要ないくつかの概念を定義する．

スケジュール $\mathcal{Q} = Q_0, Q_1, \ldots$ において，各 i $(i \geq 0)$ について $|Q_i| = 1$ が成り立つとき，$\mathcal{Q}$を**逐次スケジュール**という．逐次スケジュール $\mathcal{Q}$ に対し，$Q_h = Q_{h+1} = \ldots = Q_k$ が成り立つ $\mathcal{Q}$の極大部分系列 $Q_h, Q_{h+1}, \ldots Q_k$を $\mathcal{Q}$の**フラグメント**とよぶ．逐次スケジュール $\mathcal{Q}$ に対し，次の2条件を満たす極小の接頭部系列 $\mathcal{Q}^1 = Q_0, Q_1, \ldots Q_{j_1}$を $\mathcal{Q}$ の第1**ラウンド**とよぶ．

- $\mathcal{Q}^1$は $\mathcal{Q}$ のフラグメントの連結である．
- $\bigcup_{Q \in \mathcal{Q}^1} Q = P$.

同様に，Q_{j_1+1}から始まり，フラグメントの連結であり，かつ，すべてのプロセスが現れる極小の部分系列を $\mathcal{Q}$の第2ラウンドとよぶ．第3ラウンド以降も同様に定義する．すべての i $(i \geq 1)$ について，逐次スケジュール $\mathcal{Q}$の第 i ラウンドが正確に n 個のフラグメントの連結であるとき，$\mathcal{Q}$ を**線形スケジュール**とよぶ．

線形スケジュール $\mathcal{Q} = Q_0, Q_1, \ldots$ に関する実行 $E = c_0, c_1, \ldots$ を**線形実行**をとよぶ．また，$\mathcal{Q}$ の第 i ラウンドが $\mathcal{Q}^i = Q_h, Q_{h+1}, \ldots Q_k$ のとき，実行 Eの部分系列 $R^i = c_h, c_{h+1}, \ldots c_{k+1}$を実行 Eの第 i **ラウンド**とよび，c_h, c_{k+1}をそれぞれ第 i ラウンドの**開始状況**，**終了状況**とよぶ．

次に，巡回ラウンド数を定義する．

定義 5 (巡回ラウンド数)

プロトコル $\mathcal{A}$を，正当な状況集合 $\mathcal{L}$ に関する自己安定相互排除プロトコルとする．次の条件を満たす最小の k をプロトコル $\mathcal{A}$ の巡回ラウンド数という．

- 任意の正当な状況 c_0に対し，c_0から始まり，第 kラウンドの終了状況までにすべてのプロセスが特権を持つ線形実行が存在する． ■

3 巡回ラウンド数の下界

この節では，巡回ラウンド数 $\lfloor \frac{n}{2} \rfloor$ （ただし，$n \geq 3$）の強安定相互排除プロトコルが存在しないことを証明する．

補題 1 強安定相互排除プロトコルの正当な状況から始まる任意の線形実行を Eとし，Eの任意のラウンドを Rとする．Rで特権を持つプロセスが存在する場合，Rで最初に特権を持つプロセスを p_iとする．このとき，Rで p_i以外のプロセス p_jが特権を持つならば $j = i+1$ が成り立つ．

証明 ラウンド Rでプロセス p_j （ただし，$j \notin \{i, i+1\}$）が特権を持つと仮定し，矛盾を導く．

Rで p_i, p_jが特権を持つ（正当な）状況をそれぞれα, βとする（図1）．Rの開始状況における p_jの状態を a，状況βにおける p_{j-1}の状態を bとする．Rは線形実行のラウンドなので，各プロセスは Rの部分系列で連続して状態遷移する．従って，p_j, p_{j-1}の状態がそれぞれ a, bの状況から p_jだけが連続して状態遷移することにより，p_jが特権を持つ状況βに到達する．このときの p_jの状態遷移の系列をτと表す．

一方，Rは線形実行のラウンドなので，状況αにおける p_jの状態は a である．状況αにおいて，p_{j-1}の状態を bに変更して得られる状況をα'とする (状況αで p_{j-1}の一時故障によって得られる)．α'において，p_iは特権を持っており，p_j, p_{j-1}の状態はそれぞれ a, b である．p_jの状態遷

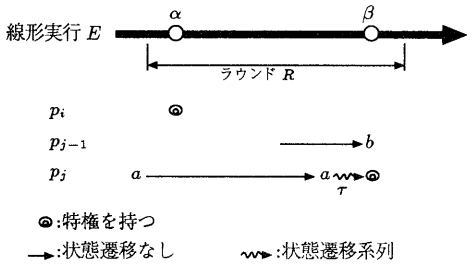

図 1: 線形実行 E (補題 1 の証明)

移は，p_jとp_{j-1}の状態のみで決まるので，α'からp_jだけが状態遷移系列τをすることにより，p_i, p_jが特権を持つ状況に到達する．このことは，プロトコルが強安定相互排除プロトコルであることに矛盾する． ■

定理 1 巡回ラウンド数$\lfloor \frac{n}{2} \rfloor$（ただし，$n \geq 3$）の強安定相互排除プロトコルは存在しない．

証明 巡回ラウンド数$\lfloor \frac{n}{2} \rfloor$の強安定相互排除プロトコルが存在すると仮定し，矛盾を導く．このとき，任意の正当な状況cに対し，cから始まり，第$\lfloor \frac{n}{2} \rfloor$ラウンドの終了状況までにすべてのプロセスが特権を持つ線形実行$E(c)$が存在する．

(a) nが奇数のとき：線形実行$E(C)$において，第$\frac{n-1}{2}$ラウンドの終了状況までに，すべてのプロセスが特権を持つ．従って，3個以上の異なるプロセスが特権を持つラウンドが存在する．一方，補題 1 より，線形実行の各ラウンドで特権を持つのは高々2個のプロセスであり，矛盾を生じる．

(b) nが偶数のとき（図 2）：$E(C)$の第$\frac{n}{2}$ラウンドの終了状況までにすべてのプロセスが特権を持つこと，および，各ラウンドで特権を持つのは高々2個のプロセスであること（補題 1）より，$E(C)$の第$\frac{n}{2}$ラウンドまでの各ラウンドで正確に2個のプロセスが特権を持つ．$E(C)$の第1ラウンドRで特権を持つプロセスをp_i, p_{i+1}とし，p_i, p_{i+1}がそれぞれ特権を持つ状況をα, βとする．

同様に，$E(\beta)$の第1ラウンドR'でも正確に2個のプロセスp_{i+1}, p_{i+2}が特権を持つ．p_{i+2}が特権を持つ状況をγとし，βでのp_{i+2}の状態をa，γでのp_{i+1}の状態をbとする．R'は線形実行のラウンドなので，各プロセスはR'の部分系列で連続して状態遷移する．従って，p_{i+2}, p_{i+1}の状態がそれぞれa, bの状況からp_{i+2}だけが連続して状態遷移することにより，p_{i+2}が特権を持つ状況γに到達する．このときのp_{i+2}の状態遷移の系列をτと表す．

状況αでのp_{i+2}の状態をcとすると，$c \neq a$であることを示す．$c = a$ならば，αにおいて，p_{i+1}の状態をbに変更して得られる状況をα'とする．状況α'において，p_iは特権を持っており，p_{i+2}, p_{i+1}の状態はそれぞれa, bである．p_{i+2}の状態遷移は，p_{i+2}とp_{i+1}の状態のみで決まるので，α'からp_{i+2}だけが状態遷移系列τをすることにより，p_i, p_{i+2}が特権を持つ状況に到達する．このことはプロトコルが強安定相互排除プロトコルであることに矛盾する．従って，状況αでのp_{i+2}の状態をcとすると，$c \neq a$である．

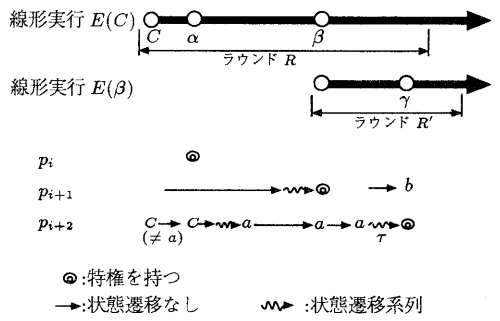

図 2: 線形実行 $E(C), E(\beta)$ (定理 1 の証明)

状況αとβでp_{i+2}の状態が異なることより，ラウンドRにおいて，αとβの間で，p_{i+2}は状態遷移している．各プロセスはRの部分系列で連続して状態遷移すること，および，p_{i+1}は状況βで特権を持つことより，Rにおいて，p_{i+2}の状態遷移はp_{i+1}の状態遷移より前に起きる．p_{i+2}が状態遷移するときのp_{i+1}の状態は，状況αでのp_{i+1}の状態と等しいので，p_{i+2}のこの状態遷移は状況αからも起こりうる．この状態遷移が起こると，p_iが特権を持ち（αと同じ状態），p_{i+2}の状態がaの状況σに到達する．正当な状況の閉包性より，状況σも正当な状況である．状況σにおいて，p_{i+1}の状態をbに変更して得られる状況をσ'とする．状況σ'において，p_iは特権を持っており，p_{i+2}, p_{i+1}の状態はそれぞれa, bである．σ'からp_{i+2}だけが状態遷移系列τをすることにより，p_i, p_{i+2}が特権を持つ状況に到達する．このことは，プロトコルが強安定相互排除プロトコルであることに矛盾する． ■

4 強安定相互排除プロトコル

この節では，巡回ラウンド数$\lfloor \frac{n}{2} \rfloor + 1$の強安定相互排除プロトコルを示す．このプロトコルは，Dijkstra [1] の自己安定相互排除プロトコルを強安定プロトコルに拡張したものである．そこで，まずDijkstraのプロトコルを紹介する．ただし，簡単のため，各プロセスの動作を状態遷移関数ではなく，プログラムとして表す．

4.1 Dijkstra のプロトコル

Dijkstra は1方向リングで相互排除問題を解く自己安定プロトコルを示している．このプロトコルでは，1方向リング上で**トークン**を周回させ，トークン持つプロセスが特権を持つ．プロセスp_0は**特別なプロセス**とよばれ，他のプロセスと異なる動作をするが，p_0以外のプロセスはすべて同じ動作をする．

トークンを実現するために，各プロセスp_iは変数s_iを持ち，その値は$\{0, 1, \ldots K-1\}$（ただし，$K > n$）のいずれかである．プロセスp_0は，$s_0 = s_{n-1}$のときトークンを持ち，トークンを転送するために$s_0 := ++s_0$を実行する．ただし，$++s_0$は$(s_0+1) \mathbf{mod} K$を表す．一方，p_0以外のプロセスp_iは$s_i \neq s_{i-1}$のときトークンを持ち，$s_i := s_{i-1}$を実行する．Dijkstra のプロトコルを 図 3 に

$1 \leq i \leq n-1$ の場合

S1 $s_i := s_{i-1}$

$i = 0$ の場合

S1 **if**$(s_0 = s_{n-1})$

S2 $s_0 := ++ s_0$

図 3: Dijkstra のプロトコル (プロセス p_i の動作)

示す.

4.2 Herman のプロトコル

Herman[3] は, Dijkstra のプロトコルを拡張し, 巡回ラウンド数 n の強安定相互排除プロトコルを示している.

Dijkstra のプロトコルでは, 2 個のトークンが存在する 1 故障状況があり, この状況から始まる実行では, これらのトークンがそれぞれリング上を移動し, 故障プロセスでない 2 個のプロセスが特権を持つ状況になる. つまり, Dijkstra のプロトコルは強安定相互排除プロトコルではない.

一方, Herman のプロトコルは, 強安定性を満たすために, 1 個の主トークンと n 個の副トークンを用いている. すべてのトークンは基本的に Dijkstra のプロトコルにしたがってリング上を周回する. 主トークンの特別なプロセス (Dijkstra のプロトコルの p_0 に相当) は p_0 である. また i 番目 ($0 \leq i \leq n-1$) の副トークンの特別なプロセスは p_i であり, これを p_i の副トークンと呼ぶ.

このプロトコルの基本的な動作は次のとおりである. プロセス p_i は, 主トークンを受け取るとそれを保持したまま p_i の副トークンを回す. その副トークンがリングを 1 周して戻ってきたとき p_i は特権を持つ. また, 主トークンを p_{i+1} に渡すことにより, p_i は特権を手放す. このため, すべてのプロセスが特権を持つためには, 副トークンが n 周する必要があり, 巡回ラウンド数は n となる. また, $n+1$ 個のトークンを利用するので, 各プロセスの状態数は $O(n^{n+1})$ である.

このプロトコルでは, 主トークンと副トークンを利用することにより, 強安定性を満たしている. 直感的には次のように考えられる. 正当な状況からの 1 故障により, 2 個の主トークンがある状況になったとする. 主トークンを持つ 2 つのプロセスはすぐには特権を持たず, まず副トークンを回す. これらの副トークンがリングを 1 周するときに, 各プロセスの状態を変更し, いずれか一方の主トークンを消去する (両方の主トークンが消去されることはない). したがって, いずれかの副トークンがリングを 1 周するまでに主トークンはただ 1 つになり, 故障してない 2 つのプロセスが同時に特権を持つ状況に到達することはない.

4.3 プロトコル $\mathcal{A}$ の概略

本稿で提案する強安定相互排除プロトコル $\mathcal{A}$ の巡回ラウンド数, (各プロセスの) 状態数はそれぞれ, $\lfloor \frac{n}{2} \rfloor + 1$, $O(n^2)$ であり, Herman のプロトコルと比べ, 巡回ラウンド数, 状態数を改善している.

プロトコル $\mathcal{A}$ は, Herman のものと同様, 主トークンと副トークンを利用し, 強安定性を実現している. ただし, 1 つの副トークンをすべてのプロセスで共用することにより, 状態数を $O(n^2)$ にしている. また, 補題 1 より, 正当な状況から始まる線形実行の 1 ラウンドで特権を持てるプロセスは高々 2 個であり, しかも, それらはリング上で連続している. そこで, プロトコル $\mathcal{A}$ では, 各ラウンドで 2 つの連続したプロセスが特権を持てるようにし, 巡回ラウンド数を $\lfloor \frac{n}{2} \rfloor + 1$ にしている.

プロトコル $\mathcal{A}$ の基本的な動作を説明する. プロトコル $\mathcal{A}$ では, 線形実行の各ラウンドで特権を持つのは, ある j について, プロセス p_{2j} と p_{2j+1} である. そのため, $P_B = \{p_{2j} \mid 0 \leq j < \frac{n}{2}\}$ のプロセスと $P_W = \{p_{2j+1} \mid 0 \leq j < \frac{n}{2}\}$ のプロセスの動作は大きく異なる. プロセスの P_B, P_W への分類には, プロセスを黒と白に塗り分ける自己安定色分けプロトコルを用いる. 以下では, P_B, P_W のプロセスを, それぞれ**黒プロセス**, **白プロセス**とよぶ.

黒プロセス p_i の動作は, Herman のプロトコルとほぼ同様である. p_i は主トークンを得ると, 副トークンの到着を待つ (正当な状況から始まる実行では, 主トークンと一緒に副トークンも到着する). 副トークンが到着すると, p_i は主トークンを保持したまま副トークンを回し, **待機状態**になる. その副トークンがリングを 1 周して p_i に戻ってきたとき, p_i は特権を持つ. また, 主トークンと副トークンを p_{i+1} に渡すことにより, p_i は特権を手放す.

一方, 白プロセス p_i は, 副トークンを受けとると, これを p_{i+1} に渡すが, このとき p_{i-1} が待機状態であれば, p_i も待機状態になる. そして, p_{i-1} が特権を手放すとき, 主トークンと副トークンを同時に受け取り, p_i は特権を持つ. つまり, p_i は p_{i-1} に続いて特権を得ることになり, 巡回ラウンド数が $\lfloor \frac{n}{2} \rfloor + 1$ となる. また, p_i は主トークンと副トークンを p_{i+1} に渡すことにより, 特権を手放す.

次にプロトコル $\mathcal{A}$ の強安定性について説明する. 正当な状況では, 主トークン, 副トークンとも正確に 1 個ずつ存在する. 1 プロセスの一時故障により, 主トークン, 副トークンが 1 個ずつ生成されることがある. このため, 異なる 2 つの非故障プロセスが特権を持つ (主トークンと副トークンを同時に持つ) 可能性があるが, 実際には, 1 故障状況から始まる実行には, そのような状況は現れない. その理由を簡単に説明する.

まず, 1 故障状況について考える. 特権を持つプロセスは待機状態のプロセスに限られるが, 正当な状況では待機状態のプロセスは高々 2 個存在し, それらは連続している (このプロセスを p_{2j}, p_{2j+1} とする). 1 故障状況で異なる 2 つの非故障プロセスが特権を持つならば, そのプロセスは p_{2j} と p_{2j+1} である. 一方, 故障プロセスを p_i ($i \notin \{2j, 2j+1\}$) とすると, 故障によって (主/副) トークンが生成されるのは, p_i と p_{i+1} に限られる. 従って, 1 故障状況で p_{2j+1} が特権を持つことはない.

一方, 副トークンが移動する場合, 副トークンの移動に伴い, 各プロセスの状態は変更される. このため, 副トークンが主トークンに到達するまでに, いずれかの主トークン, あるいは, 副トークンが消滅する. これにより, 1 故障状況から到達できる他の状況で, 異なる 2 つの非故障プロセスが特権を持たないことが保証される.

これまでに述べた方法で，強安定相互排除プロトコルを実現できる．しかし，このままでは巡回ラウンド数が$\lceil \frac{n}{2} \rceil + 1$となる．定理1で示した巡回ラウンド数の下界は$\lfloor \frac{n}{2} \rfloor + 1$であり，$n$が奇数のときには，巡回ラウンド数最適といえない．実際，以下に述べる方法で，$\lfloor \frac{n}{2} \rfloor + 1$のプロトコルを構成できる．

nが奇数の場合に，上記の方法で巡回ラウンド数最適とならないのは，正当な状況から始まる任意の線形実行において，p_{n-1}だけが特権を持つラウンドが生じるためである．つまり，プロセスp_{2j}とp_{2j+1}を組にしたために，全体で$\lceil \frac{n}{2} \rceil$個の組が生じるためである．一方，$p_{2j+1}$と$p_{2j+2}$を組にすると，$p_0$だけが特権を持つラウンドが生じてしまう．そこで，この組を動的に変化させることにより，巡回ラウンド数最適な強安定相互排除プロトコルが得られる．各プロセスは受け取った主トークンの色により，自分の動作(P_BあるいはP_W)を入れ換える．つまり，主トークンが1周してp_0を通過するたびに色が変化し，それによってリング上のプロセスペアが動的に変化する．このような仕組みは比較的容易に実現できる．

4.4 プロトコル $\mathcal{A}$

プロトコルAを図4に示す．簡単のため，主トークンが色を持たないプロトコル(巡回ラウンド数$\lceil \frac{n}{2} \rceil + 1$)を示す．前節に述べた方法で，このプロトコルから巡回ラウンド数$\lfloor \frac{n}{2} \rfloor + 1$の強安定相互排除プロトコルが得られる．以下に，プロトコル$\mathcal{A}$で用いる変数の説明を行う．

- *state:* プロセスの色を表す変数．0,1のいずれかの値を取り，0は黒，1を白とする．
- *wait:* 特権を持つ待機状態を表す変数．0,1のいずれかの値を取り，0は通常の状態，1は待機状態．
- *major:* 主トークンを実現するための変数．0からnまでの整数値を取る．主トークンを持つプロセスp_iは次のように定義できる．
  - $1 \leq i \leq n-1$の場合．
    $++major_i = major_{i-1}$.
  - $i = 0$の場合．
    $major_i = major_{i-1}$.
- *minor:* 副トークンを実現するための変数．0からnまでの整数値を取る．副トークンを持つプロセスp_iは次のように定義できる．
  - $1 \leq i \leq n-1$の場合．
    $++minor_i = minor_{i-1}$.
  - $i = 0$の場合．
    $minor_i = minor_{i-1}$.

4.5 プロトコル $\mathcal{A}$ の正当性

まず，プロトコル$\mathcal{A}$の正当な状況$\mathcal{L}$を定義する．次のことを順に示すことでプロトコル$\mathcal{A}$の正当性を示す．

プロトコル$\mathcal{A}$の正当な状況を定義する．

定義 6 (正当な状況 $\mathcal{L}$)
cを任意の状況とする．cにおいて，以下の5条件がすべて満たされるとき，かつ，そのときに限り，cは正当な状況である．

1. 各$i(0 \leq i \leq n-1)$について，$state_i = i \bmod 2$が成り立つ．

$1 \leq i \leq n-1$の場合

```
S1  if(state_{i-1} = 1)
S2      state_i := 0
S3      if(++major_i = major_{i-1} ∧
                       ++minor_i = minor_{i-1})
S4          if(!wait_i)
S5                  wait_i := 1

S6          else
S7                  perform CS
S8                  wait_i := 0
S9                  major_i := major_{i-1}

S10     else if(!(++major_i = major_{i-1} ∧
                       ++minor_i = minor_{i-1}))

S11         wait_i := 0
S12         major_i := major_{i-1}


S13 else
S14     state_i := 1
S15     if(wait_{i-1} = 1 ∧ major_i = major_{i-1} ∧
                       ++minor_i = minor_{i-1})
S16         wait_i := 1

S17     else if(wait_i = 1 ∧ ++major_i = major_{i-1} ∧
                       ++minor_i = minor_{i-1})

S18         perform CS
S19         wait_i := 0

S20     else if(!(wait_{i-1} = 1 ∧ wait_i = 1 ∧
                       major_i = major_{i-1}))
S21         wait_i := false
S22     major_i := major_{i-1}

S23 minor_i := minor_{i-1}
S24 goto S1
```

$i = 0$の場合

```
S1  state_i := 0
S2  if(minor_i = minor_{i-1})
S3      if(major_i = major_{i-1})
S4          if(!wait_i)
S5              wait_i := 1

S6          else
S7              Perform CS
S8              wait_i := 0
S9              major_i := ++major_{i-1}

S10     minor_i := ++minor_{i-1}

S11 if(!(minor_i = ++minor_{i-1} ∧
                       major_i = major_{i-1}))
S12     wait_i := 0

S13 goto S1
```

図4: プロトコル$\mathcal{A}$(プロセスp_iの動作)

2. ある $i(0 \leq i \leq n-1)$ が存在し，任意の $j(0 \leq j \leq i)$ について $major_j = major_0$ が成立し，任意の $k(i+1 \leq k \leq n-1)$ について $major_k = ++major_0$ が成立する．
3. ある $i(0 \leq i \leq n-1)$ が存在し，任意の $j(0 \leq j \leq i)$ について $minor_j = minor_0$ が成立し，任意の $k(i+1 \leq k \leq n-1)$ について $minor_k = ++minor_0$ が成立する．

4. 主トークンを持つプロセス p_i と副トークンを持つプロセス p_j が異なる場合，$state_i = 0$．
5. • 主トークンと副トークンを1つのプロセス p_i が持っている場合．
   * $state_i = 0$ の場合．
     (a) $wait_i = wait_{i+1} = 1$ かつ，任意の $j(j \neq i, i+1)$ について $wait_j$=0.
     (b) 任意の j について $wait_j = 0$.
   * $state_i = 1$ の場合．
     (c) $wait_i = 1$ かつ，任意の $j(j \neq i)$ について $wait_j = 0$.
   • 主トークンも持つプロセス P_i と副トークンを持つプロセス P_j が異なる場合．
   * $j = i+1$ の場合．
     (d) $wait_i = 1$ かつ，任意の $k(k \neq i)$ について $wait_k = 0$.
   * $j \neq i+1$ の場合．
     (e) $wait_i = wait_{i+1} = 1$ が成立し，任意の $k(k \neq i, i+1)$ について $wait_k = 0$.

■

プロトコル $\mathcal{A}$ では，Dijkstra の自己安定相互排除プロトコルを用いて，主トークン，副トークンを実現してる．従って，プロトコル $\mathcal{A}$ が $\mathcal{L}$ に関する自己安定プロトコルであることは，Dijkstra のプロトコルと同様にして証明できる．ここで，$\mathcal{L}$ は定義6で定義される正当な状況の集合を表す．

以下では，プロトコル $\mathcal{A}$ の強安定性の証明の方針のみを示す．プロトコル $\mathcal{A}$ の概略で述べたように，1故障状況では，主トークン，副トークンはそれぞれ2個存在することがある．Dijkstra のプロトコルの性質から，プロトコル $\mathcal{A}$ の1故障状態から始まる任意の実行に現れる任意の状況において，主トークン，副トークンは，それぞれ，高々2個であることを証明できる．

主トークン，副トークンが2個ずつある状況でも，同時に2個の非故障プロセスが特権を持つ状況に到達しないことは，トークンと次に定義するギャップに関する性質から証明できる．

正当な状況では，p_0 の変数 $minor_0$（副トークンを実現する変数）について，$minor_0 = minor_{n-1}$，あるいは，$minor_0 = ++minor_{n-1}$ が成り立つ．このいずれの条件も成り立たないとき，p_0 にギャップがあるという．p_0 以外のプロセスについても，同様に，ギャップを定義する．定義より，ギャップがない状況では，副トークンは高々1個存在する．

プロトコル $\mathcal{A}$ の1故障状況から始まる任意の実行に現れるトークンとギャップについて，以下の補題が成立する．

補題 2 副トークンがギャップに到達，もしくは，ギャップが副トークンに到達すると，その副トークンは消滅する．■

補題 3 プロセス p_k が主トークンとギャップを持つとする．p_k が動作しギャップが p_{k+1} に移るとする．このとき，この動作によって，主トークンも p_{k+1} に移る．■

補題 4 主トークンを持つプロセス p_k が動作し，主トークンが p_{k+1} に移るとする．このとき，この動作前には p_k は副トークンかギャップを持っており，この動作によって，その副トークン，あるいは，ギャップも p_{k+1} に移る．■

これらの補題を用いて，同時に2個の非故障プロセスが特権を持つ（主トークンと副トークンの両方を持つ）状況に到達しないことを証明できる．また，(色情報を持つトークンを用いた) プロトコル $\mathcal{A}$ の巡回ラウンド数と (各プロセスの) 状態数は，容易に評価でき，次の定理が成り立つ．

定理 2 （色情報を持つトークンを用いた）プロトコル $\mathcal{A}$ は強安定相互排除プロトコルであり，その巡回ラウンド数，(各プロセスの) 状態数は，それぞれ，$\lfloor \frac{n}{2} \rfloor + 1$，$O(n^2)$ である．■

5 むすび

本稿では，1方向リングにおける相互排除問題について，巡回ラウンド数 $\lfloor \frac{n}{2} \rfloor$ の強安定相互排除プロトコルは存在しないことを証明した．

また，巡回ラウンド数 $\lfloor \frac{n}{2} \rfloor + 1$ の強安定相互排除プロトコルを提案した．本稿ではプロセス間の通信モデルに状態通信モデルを用いたが，本稿のプロトコルはレジスタ通信モデルにも適応できる．

謝辞 日頃より，分散アルゴリズムについて熱心にご討論いただく本学藤原研究室の井上美智子助手，守屋宣氏に感謝致します．また，井上智生助手を始めとする本学藤原研究室の諸氏に感謝致します．なお，本研究の一部は，文部省科学研究費補助金 (奨励 (A)08780279)，財団法人電気通信普及財団の助成による．

参考文献

[1] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *CACM*, 17(11):643–644, 1974.

[2] S. Dolev and T. Herman. Superstabilizing protocols for dynamic distributed systems. In *Proc. 2nd Workshop on Self-stabilizing Systems*, pages 3.1–3.15, 1995.

[3] T. Herman. Superstabilizing mutual exclusion. In *Proc. International Conf. on Parallel and Distributed Systems*, 1995.

[4] 桝田, 片山, 増澤, 都倉. C-daemon でのリング方向付けのための自己安定アルゴリズムについて. 信学技報, COMP93-96, 1994.