NAIST-IS-DT9961005

博士論文

分散システムの高度な故障耐性を実現する自己安定プロトコルに関する研究

浮穴学慈

2001年2月12日

奈良先端科学技術大学院大学
情報科学研究科 情報処理学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である.

論文番号： NAIST-IS-DT9961005

提出者： 浮穴学慈

審査委員： 藤原 秀雄 教授
伊藤 実 教授
関 浩之 教授
増澤 利光 大阪大学教授

提出日： 2001 年 2 月 12 日

分散システムの高度な故障耐性を実現する自己安定プロトコルに関する研究*

浮穴学慈

内容梗概

複数の自律計算ユニット「プロセス」が通信リンクによって相互接続されたネットワークシステムを分散システムという．近年，局所ネットワーク (LAN) や広域ネットワーク (WAN) などのネットワーク環境が広く整備されるに伴い，複数の利用者に対し同時並行的にサービスを提供できる分散システムの重要性が増している．分散システム上でサービスを提供するためにはプロセス同士の協調動作が必要である．この協調動作に関する問題を分散問題といい，分散問題を効率良く解く通信計算手順「分散プロトコル (分散アルゴリズム)」が盛んに研究されている．

分散システムの特長の１つは，その潜在的な冗長性を活用することにより，故障耐性を有するシステムを構築できる点である．大規模な分散システムでは，システムを構成するプロセスや通信リンクの数が多いため，常にすべてが正常動作していると仮定することは難しい．分散システムの一部に故障が発生してもサービスを継続できる性質「故障耐性」が必要不可欠であり，故障耐性に優れた分散システムを設計するために，故障耐性を有する分散プロトコルが重要である．プロセスや通信リンクの故障に対し，故障耐性を有する分散プロトコルを設計するためのさまざまなアプローチが考えられているが，**自己安定プロトコル**は有望な設計パラダイムの１つである．自己安定プロトコルとは，任意のシステム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．システムの初期化が不要であるだけでなく，メモリ内容が破壊されるなど，システムが一時故障によりどのような大域状況に陥っても，その後プロセスが正常に動作していれば再び解状況に復旧する．そのため，自己安定プロトコルを用いることで，長

*奈良先端科学技術大学院大学 情報科学研究科 情報処理学専攻 博士論文，NAIST-IS-DT9961005, 2001 年 2 月 12 日.

期間稼働するシステムを安定に保ち，一時故障に柔軟に対応することができる．本研究では，このように優れた故障耐性を有する自己安定プロトコルに着目し，効率のよい自己安定プロトコルの設計，および，さらに高度な故障耐性を有する自己安定プロトコルの設計を目指している．

本論文では，まずはじめに，木ネットワークにおいてプロセス間の同期を実現する自己安定プロトコルを利用して，ヒープ順序付き木を構成する自己安定プロトコルについて考察する．ヒープは逐次アルゴリズムにおいて重要なデータ構造であり，ソートや優先順序キューなど，様々な用途に応用されている．このような重要なデータ構造を分散型データ構造として実現することは，分散システムを設計・利用する上で有用であり，様々なデータ構造について分散システム上に実現する分散プロトコルが研究されている．しかし，自己安定分散型データ構造については，まだあまり研究が行われていない．提案するヒープ順序付き木を構成するプロトコルは，安定時間 $O(h)$，各プロセスの領域計算量 $O(K)$ であり，既知の結果と比べ安定時間，領域計算量ともに改善されている．ここで，h は木の高さ，K は入力のサイズを表す．

次に，一時故障だけでなく，永久故障に対する故障耐性も有する自己安定プロトコルについて考察する．自己安定プロトコルは一時故障に対する耐性を有する優れた設計パラダイムであるが，実システムへの適用のためには，永久故障に対する耐性も必要である．まず，永久故障の新たなクラスとして，**非停止永久故障**を定義する．非停止永久故障は，故障プロセスが無限にしばしば故障動作により状態変化する故障であり，停止を許さない故障モデルである．そして，ネットワークの生成木を構成する問題に対し，非停止永久故障プロセスが存在しても問題を解く自己安定プロトコルを提案する．提案するプロトコルの安定時間は高々 $n(n/2+\mathcal{F})d$ である．ここで，n はプロセス数，d はネットワークの最大次数，$\mathcal{F}$ は故障プロセスの状態変化が観測されるまでの時間の上界 (定数) である．

キーワード

分散システム，自己安定，一時故障，永久故障，分散型データ構造，ヒープ，生成木

Studies on Self-Stabilizing Distributed Protocols for attaining High Fault-Tolerance*

Satoshige Ukena

Abstract

A distributed system consists of autonomous processes connected by communication links. In recent years, distributed systems are widely in use. They are given in various forms, e.g. communication networks (local area networks (LAN) and wide area networks (WAN)), multiprocessor computers, etc. Because of benefits that a disrtibuted system can provide parallel and simultaneous services for multiple users, importance of distributed systems is increasing. The services in a distributed system usually need collaboration of processes. A distributed protocol (algorithm) is a procedure of communication and calculation for solving a problem for process cooperation, which is called a distributed problem.

Fault-tolerance is one of the most important advantages of distributed systems. Because a large-scale distributed system consists of a large number of processes and communication links, we cannot assume that all the components are fault-free. Therefore, a property "fault tolerance" is strongly required for a distributed system. The property guarantees that the system can continue to provide services even though several portions of a distributed system are collapsed. Fault-tolerant distributed protocols are necessary for such fault-tolerant systems.

Several approaches have been proposed for attaining fault-tolerance to process or link failures on distributed systems. **Self-stabilization** is a promising paradigm for designing fault-tolerant distributed systems. A self-stabilizing protocol is a distributed protocol that achieves its intended behavior regardless of the initial configuration. This instantly means that no self-stabilizing protocol requires system initialization. And furthermore, even when the system configuration changes to an arbitrary one because of transient faults, e.g. destruction

*Doctor's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9961005, February 12, 2001.

of contents of memories, a self-stabilizing protocol converges and achieves the intended behavior. It follows that a self-stabilizing protocol is resilient to any number and any type of transient faults. A long-term-surviving system will be stable and can flexibly cope with transient faults by using self-stabilizing protocols. In this research, we are aiming at designing efficient self-stabilizing protocols and designing self-stabilizing protocols with more advanced fault-tolerance.

In this thesis, we first investigate a self-stabilizing protocol for constructing a heap-ordered tree. The protocol makes use of a self-stabilizing neighborhood synchronizer. For sequential algorithms, heap is an important data structure used in various applications, e.g. sorting and priority queue, etc. It is useful for designing distributed systems to realize such important data structures as distributed ones. Although many researches have studied on distributed protocols for various data structures, there are only few researches for self-stabilizing distributed data structures. The protocol proposed in this thesis has $O(h)$ convergence time and $O(K)$ space complexities, where h denotes the height of the tree and K denotes the size of each input value. Both of the complexities are improved comparing with previous protocols.

Next, we investigate possibility of self-stabilizing protocols that can also tolerate some type of permanent faults in addition to transient faults. Although a self-stabilizing protocol is an excellent design paradigm for tolerating transient faults, it is also desired for practical systems to tolerate some permanent faults. We newly introduce **non-quiescent permanent faults** and present a self-stabilizing protocol that constructs a spanning tree under existence of the faults. The protocol stabilizes within $n(n/2 + \mathcal{F})d$ rounds, where n is the number of processes, d is the maximum degree of the network, and $\mathcal{F}$ is the upper bound of delay before state change of faulty process is observed by a neighboring process.

Keywords:

distributed system, self-stabilization, transient fault, permanent fault, heap, spanning tree

研究業績

学術論文

1. 浮穴学慈, 長谷川学, 片山喜章, 増澤利光, 藤原秀雄, “木ネットワーク上のヒープ順序構成自己安定プロトコル”, 電子情報通信学会論文誌 (DI), Vol.J84-D-I, No.1, pp.48-57, Jan. 2001.

2. 浮穴学慈, 片山喜章, 増澤利光, 藤原秀雄, “非停止永久故障に耐性を有する自己安定生成木構成プロトコル”, 電子情報通信学会論文誌 (DI), (条件付採録).

研究会報告

1. S. Ukena, Y. Katayama, T. Masuzawa, and H. Fujiwara, “A Self-Stabilizing k-Exclusion Protocol with k-Wait-Freedom”, Technical Report of IEICE, (COMP98-87), pp.33-40, Mar. 1999.

2. 長谷川学, 浮穴学慈, 片山喜章, 増澤利光, 藤原秀雄, “ヒープ順序づき木を構成する自己安定プロトコル”, 電子情報通信学会技術報告, (COMP99-79), pp.1-8, Mar. 2000.

3. 浮穴学慈, 片山喜章, 増澤利光, 藤原秀雄, “非停止永久故障に耐性を有する自己安定生成木構成プロトコル”, 電子情報通信学会技術報告, (COMP2001-31), pp.17-23, Sep. 2001.

目 次

図 目 次

図 目 次

表 目 次

第 1 章

序論

1.1. 自己安定プロトコル

自律動作する複数の計算機が通信リンクによって相互接続されたネットワークシステムを分散システムという．近年、局所ネットワーク (LAN) や広域ネットワーク (WAN) などのネットワーク環境が整備され広く利用されるに伴い，複数の利用者に対し同時並行的にサービスを提供できる分散システムの重要性が増している．分散システムが潜在的に持つ特長としてしばしば以下の点が挙げられる．

- 高機能性: 全体を制御する制御ユニットの必要から生じるボトルネックなしに，計算を並列かつ非同期的に実行可能．
- 拡張性（scalability）: 環境の変化に応じて，システムの規模を柔軟に調整することができる．
- 可用性（availability）: システムの一部が故障しても，その機能をシステムの他の部分が代替することでサービスを継続できるので，決定的な打撃を受けにくい．
- 資源の共有: プロセッサや他の機器類，データベースなどの情報を共有できる．

一般に「分散システム」という語は，様々な文脈で用いられる．広い意味で用いられる場合，インターネットから，マルチプロセッサシステムまで，様々な計算・通信の形態，システムの規模，運用目的を持つ様々なシステムを含む．このため，分散システムに対する単一のモデルを定義することは困難である．

分散システムを理論的に扱うための様々なモデルのうち，本論文では複数のプロセスが通信リンクにより相互接続されたネットワークモデルを用いる．各プロセスは分散システム上で自律動作する計算ユニットであり，例えばインターネットにおいては各計算機，マルチプロセッサシステムにおいては各プロセッサを表す．また，通信リンクによって接続されたプロセス同士は，互いの状態を直接参照できるものとする (状態通信モデル)．

分散システム上でサービスを提供するためには，プロセス同士の協調動作が必要であり，これは各プロセス上で分散管理されている情報の交換を通して行われる．この協調動作に関する問題を分散問題といい，分散問題を効率良く解く通信計算手順「分散プロトコル (分散アルゴリズム)」が盛んに研究されている．

はじめに挙げたように，分散システムの特長の１つは，その潜在的な冗長性を活用することにより，可用性を有するシステムを構築できる点である．大規模な分散システムでは，システムを構築するプロセスや通信リンクの数が多いため，常にすべてが正常動作していると仮定することは難しい．分散システムの一部に故障が発生してもサービスを継続できる性質「故障耐性」が必要不可欠であり，故障耐性に優れた分散システムを設計するためには，故障耐性を有する分散プロトコルが必要である．

自己安定プロトコル（self-stabilizing protocol）とは，分散システムの任意の状況からプロトコルの実行を開始しても，やがて解状況に到達し安定する分散プロトコルである．自己安定プロトコルが任意の状況から解状況に到達し安定するまでの時間を安定時間と呼ぶ．上述の性質により自己安定プロトコルは次の特長を持つ．

- 分散システムの初期化が不要である．各プロセス，各通信リンクに対して初期化を必要としないため，実システムにおいては，システム構成要素の起動順序などを考慮する必要がなく，システム管理の労力を軽減できる．
- どのような種類，規模の一時故障（transient fault）に対しても耐性を持つ．一時故障は，メモリ内容の破壊などによるプロセスの持つ動的データの破壊 (プログラムカウンタを含む) など，一時的な故障である．一時故障によりシステムがどのような状況に陥っても，プロセスがプロトコルの実行を継続すれば自動的に解状況に復旧し安定する．ただし，通常の自己安定プロトコルでは，一時故障に対応するためには安定時間に比べ十分長い間一時故障が起こらないという条件が必要である．
- 分散システムのネットワーク形状の変化に対応できる．プロトコルの実行中に，プロセスの増減や通信リンクの増減などのネットワーク形状の変化が生じる，動的ネットワーク（dynamic network）においても，安定時間に比べ十分長い時間形状変化しなければ，プロトコルは解状況に到達する．ただし，ネットワークサイズなどの大域情報を利用するプロトコルに関しては，形状変化に応じて大域情報を供給する必要がある．

自己安定プロトコルは非マスク型 (分散システムの利用者に故障の影響が一時的に及ぶことを許す) の高度な故障耐性を実現する分散プロトコルであり，自己安定プロトコルを用いることで，長期間稼働するシステムを安定に保ち，一時故障に柔軟に対応することができる．このように優れた性質を持つ自己安定プロトコルは，Dijkstra によってはじめて提案され [6]，現在分散プロトコルの研究で最も盛んに研究されている分野のひとつである．本研究では，このように優れた故障耐性を有する自己安定プロトコルに着目し，効率のよい自己安定プロトコル，

および，さらに高度な故障耐性を有する自己安定プロトコルの設計を目指している．本論文では，次の 2 つの自己安定プロトコルを提案する．

- 木ネットワーク上のヒープ順序構成自己安定プロトコル
- 非停止永久故障に耐性を有する自己安定生成木構成プロトコル

1.2. 木ネットワーク上のヒープ順序構成自己安定プロトコル

ヒープは，逐次アルゴリズムにおいて重要なデータ構造であり，ソートや優先順序キューなど，多くの応用を持つ．このような重要なデータ構造をネットワーク上に分散型データ構造として実現し，分散システムの設計・開発に利用することは有用である．そのため，様々なデータ構造に対し，それらをネットワーク上に実現するための分散プロトコルに関する研究が，数多く行われている．しかし，自己安定分散型データ構造については，まだあまり研究が行われていない．

本論文では，木ネットワークで**ヒープ順序付き木**を構成する自己安定プロトコルを提案する．ここで，ヒープ順序付き木の構成とは，各プロセスが持つ値（入力値）を，ヒープ順序を満たす（つまり，各プロセスがその子プロセスよりも大きい値を持つ）ように並べ替えることである．なお，本論文では木ネットワークを対象とするが，生成木を構成する自己安定プロトコルを併用することにより，提案するプロトコルは一般のネットワークに対しても適用できる．

木ネットワークでヒープ順序付き木を構成する自己安定プロトコルは，Bourgon ら [4] によって提案されている．このプロトコルの安定時間は $O(nh)$ である．ここで，n はシステムのプロセス数，h は木の高さである．また，各プロセスの領域計算量は $O(\delta K)$ である．ここで，δ はプロセスの次数，K は入力値のサイズである．

Alima[2] は，木ネットワークに対して，安定時間 $O(h)$，各プロセスの領域計算量 $O(\delta + K)$ のヒープ順序付き木を構成する自己安定プロトコルを提案している．しかしこのプロトコルでは隣接プロセス間で 1 度しか値の比較・交換を行わず，正しくヒープ順序付き木を構成できない．このプロトコルを修正したものを繰返し適用すればヒープ順序付き木の構成も可能となるが，そのときの安定時間は $O(h^2)$ になってしまう．

本論文では，安定時間 $O(h)$，各プロセスの領域計算量 $O(K)$ のヒープ順序付き木を構成する自己安定プロトコルを提案する．これは文献 [4]，[2] のプロトコルと比べ，安定時間，領域計算量を共に改善している．

文献 [4]，[2] のプロトコルの安定時間が大きいのは，大域的同期化プロトコル（根がシステム全体の調停者となってプロセスを同期させる）を使用しているためである．そこで，本論文では，隣接プロセス間でのみ同期を実現する**隣接間同**

	安定時間	プロセスあたりの領域計算量
Bourgon ら [4]	$O(nh)$	$O(\delta K)$
本論文	$O(h)$	$O(K)$

n: プロセス数, h: 木の高さ

δ: プロセスの次数, K: 入力値のサイズ

表 1.1 木ネットワーク上のヒープ順序構成自己安定プロトコル

期化プロトコル [13] を利用して，ヒープ順序付き木を構成する自己安定プロトコルを設計する．

同期化プロトコルは，非同期式分散システムで同期式分散システムを模倣することを可能にするプロトコルであり，幅広く研究されている [14, 17, 18]．Johnen らは，木ネットワークにおいて自己安定隣接間同期化プロトコルを提案している [13]．つまり，任意の隣接プロセス間で，同時に両方が動作することなく，いずれか一方ずつ，交互に動作させる仕組みを提供する．このプロトコルの安定時間は 0 であり，1 プロセッサにつき 1 ビットの領域 (領域計算量 $O(1)$) を必要とする．

1.3. 非停止永久故障に耐性を有する自己安定生成木構成プロトコル

自己安定プロトコルは一時故障に対する高度な故障耐性を持つ．しかし，自己安定プロトコルは，故障状況から再安定するまでの実行の間，再び故障が生じないことを仮定している．つまり，現実のシステムのように，一度障害を起こした計算機が，その後も断続的に障害を起こすような場合，自己安定プロトコルは解状況に復帰して安定することを保証しない．このような断続的な障害は永久故障の枠組で扱われる．実システムにおける永久故障に対応するため，一時故障だけでなく，永久故障に対する故障耐性も有する自己安定プロトコルに関する研究が数多く行なわれている [12, 3, 15, 16]．本論文でも，永久故障に対する故障耐性を有する自己安定プロトコルについて考察する．

プロセスの永久故障としては，これまでに，ビザンチン故障，停止故障などのさまざまな故障モデルが考えられている．ビザンチン故障モデルは，故障プロセスの動作に仮定を設けない故障モデルであり，最も大きな故障クラスである．一方，停止故障モデルは，故障したプロセスはそれ以降，一切動作しなくなる故障モデルである．停止故障プロセスが 1 つでも存在すると，コンセンサス問題のような

単純な問題でさえ解けないことが知られている [9]．これは，非同期システムにおいては，プロセスが停止故障していることと単に動作が遅いことが有界時間で区別できないことによる．また，Anagnostou ら [3] は，故障敏感 (failure-sensitive) な問題のクラスを定義し，停止故障プロセスが 1 個でも存在すると，故障敏感な問題を解く自己安定プロトコルが存在しないことを示した．生成木構成問題は故障敏感な問題のクラスに属する．

生成木構成問題は基本的な分散問題のひとつであり，経路情報の管理や同報通信など，さまざまな応用が存在する．通常の分散アルゴリズムの研究では，Gallager ら [11] などにより盛んに研究されてきた．また，自己安定プロトコルにおいては，Dolev ら [8] や Chen ら [5] をはじめ，さまざまな研究が行なわれている．なお，最適な自己安定生成木構成プロトコルは，Aggrawal ら [1] によって提案された，レジスタ通信モデル上で安定時間 $O(diam)$ のものである．$diam$ はネットワークの直径である．このプロトコルでは大域的なプロセス識別子を用いている．

これまで，永久故障を考慮した自己安定生成木構成プロトコルは提案されていない．本論文では，新たに**非停止永久故障**を導入することで，状態通信モデルにおいて，永久故障に耐性を有する自己安定生成木構成プロトコルを提案する．非停止永久故障は，故障プロセスが無限にしばしば，故障動作により状態変化する故障モデルである．停止を許さないという制限の下で最も性質の悪い故障であるといえる．本論文では，非停止永久故障の下で生成木構成問題を解く自己安定プロトコルを提案するが，これは，自己安定生成木構成問題の可解性にとって，停止故障が致命的な影響を与えることを意味する．

本論文で提案するプロトコルは，故障プロセス数に対して制限を置いていない．従って，永久故障に対する優れた故障耐性を実現しているといえる．また，大域的なプロセス識別子は用いない．そのかわりに，故障しない根プロセスを仮定する．これはネットワークが完全に対称であると，決定性プロトコルで問題を解くことが不可能なため導入した仮定であるが，故障耐性の観点からは欠点といえる．また停止故障と区別するため，故障プロセスの状態変化が，すべての正常な隣接プロセスによって無限にしばしば観測されるという仮定を設けている．提案するプロトコルの安定時間は $n(n/2+\mathcal{F})d$ である．n はプロセス数，d はネットワークの最大次数，$\mathcal{F}$ は故障プロセスの状態変化が観測されるまでの時間の上界 (定数) であり，安定時間の評価のために導入した．

第 2 章

木ネットワーク上のヒープ順序構成自己安定プロトコル

本章では，木ネットワークにおいて，プロセス間の同期を実現する自己安定プロトコルを利用して，ヒープ順序を構成する自己安定プロトコルについて考察する．まずはじめに第 2.1節において，本章で扱う分散システム，自己安定プロトコル，隣接間同期化問題，ヒープ順序付き木構成問題について定義を行う．次に，第 2.2節では，Johnen ら [13] の隣接間同期化プロトコルを示す．第 2.3節では，ヒープ順序付き木を構成する自己安定プロトコルを示し，正しさを証明する．最後に，第 2.4で本章のまとめをする．

2.1. 諸定義

2.1.1 分散システム

分散システムは，無向連結グラフ $G = (V, E)$ で定義される．ここで，V は頂点の集合 ($V = \{0, 1, \cdots, n-1\}$ とする)，E は辺の集合とする．それぞれの頂点はプロセスを表し，辺は双方向通信リンクを表す．本章では，木構造のネットワークを扱う．ネットワーク中で根プロセスを r，葉プロセスの集合を L，そして内部プロセス (根プロセスは内部プロセスに含めない) の集合を I で表す．

各プロセス i の隣接プロセスの集合を N_i とする．各プロセス i は隣接するプロセスを区別できるものとする．また，隣接するプロセス数 $|N_i|$ をプロセス i の次数という．各プロセス i は，木ネットワーク上での親と子のプロセスを認識できるとし，親プロセスを P_i，子プロセスの集合を Cld_i で表す．従って P_i, Cld_i から，プロセス i は自分が根，葉，内部プロセスのいずれであるかを認識できる．また，木の高さを h で表す．

各プロセスを状態遷移機械としてモデル化する．状態を変数で表し，状態遷移をガード付アクション (以下，アクションという) で表す．ここで，各プロセスは

自分のもつ変数のみに書き込み可能であるとし，また，隣接プロセスの変数の値を直接参照できる (状態通信モデル) ものとする．各プロセスの各アクションはラベルづけされており，以下のように表す．

$$< label > :: < guard > \rightarrow < statement >$$

i の各アクションのガード $< guard >$ は，i および i の隣接プロセスの変数からなる論理式で表される．プロセス i はガードが真の場合のみ命令文 $< statement >$ を実行する．命令文では，i の持つ変数を更新する．本章では，ガードが評価され，命令文が実行されるまでを 1 原子動作とする．また，この 1 原子動作を 1 ステップと呼び，ガードが真の命令文の実行をアクションの実行と呼ぶ．

各プロセス i の状態集合を S_i で表す．分散システム全体の取り得る状況の集合を $\mathcal{C}$ とすると，$\mathcal{C} = S_0 \times S_1 \times \cdots \times S_{n-1}$ である．つまり，分散システムの各大域状況 (以下単に状況と呼ぶ) c は，$(s_0, s_1, \ldots, s_{n-1})$ で表される．ここで，$s_i \in S_i$ $(0 \leq i \leq n-1)$ である．

プロトコル $\mathcal{P}$ は，各プロセスの動作を規定する．従って，プロトコル $\mathcal{P}$ を，すべての状況の集合 $\mathcal{C}$ 上の 2 項関係 $\mapsto$ とみなすことができる．$c \in \mathcal{C}$ を任意の状況，$Q \subseteq V$ をプロセスの任意の部分集合とする．Q に属するすべてのプロセスが，同時に 1 ステップを行うことにより状況が c から c' となる (つまり $c \mapsto c'$) とき，$c' = \Delta(c, Q)$ と表す．$c = (s_0, s_1, \ldots, s_{n-1})$，$c' = (s'_0, s'_1, \ldots, s'_{n-1})$ とするとき，$i \notin Q$ のとき，あるいは，$i \in Q$ でもそのアクションのガードが偽のとき，$s_i = s'_i$ である．

プロセスの空でない部分集合の無限系列をスケジュールと呼び，$\mathcal{Q} = Q^0, Q^1, Q^2, \ldots$ と表す．このとき，状況の無限系列 $\mathcal{E} = c^0, c^1, c^2, \cdots$ が $c^{i+1} = \Delta(c^i, Q^i)$ $(i \geq 0)$ を満たすとき，$\mathcal{E}$ 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，$\mathcal{E}$ は $Q^0, Q^1, Q^2, \ldots$ に属するプロセスが順に動作するときの状況変化を表す系列である．本章では，弱公平な実行のみを考える．これは，各プロセス i に対し，ある状況 c^j 以降常にガードが真でありながら，c^j 以降 命令文 が実行されないようなアクションは存在しないことを保証する．つまり，連続してガードが真のアクションを持つとき，いつかはその命令文が実行されることを保証する．

本章では，非同期式分散システムを想定しているが，時間計算量の評価は，同期式分散システムで用いられるラウンドを用いて行う．つまり，時間計算量の評価では，各 $i(i \geq 0)$ に対し，$Q^i = V$ となるスケジュール $\mathcal{Q} = Q^0, Q^1, Q^2, \cdots$ に対する実行 $\mathcal{E} = c^0, c^1, \cdots$ を考え，状況 c^i を第 i ラウンド終了時の状況とする．このような実行 $\mathcal{E}$ を同期式実行と呼ぶ．

2.1.2 自己安定プロトコル

自己安定プロトコルは，任意の状況から実行を開始しても，問題の解を求めた状況に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，プロセスの一時的な故障 (プロセスの変数の値，プログラムカウンタの値の破壊) により，分散システムがどのような状況に陥っても，故障したプロセスが復旧すれ

ば，やがて再び解を求めた状況で安定する．つまり，自己安定プロトコルは，一時的な故障に対する高度な故障耐性を有する．

ここでは *attractor* という概念を用いて，自己安定プロトコルを定義する．

定義 2.1 *Closed Attractor*

X, Y を，分散システムの状況の集合 $\mathcal{C}$ に対して定義される述語とする．以下の条件を満たすとき Y は X に対する *closed attractor* であるといい，$X \triangleright Y$ と表す．

1. 述語 X を満たす任意の状況を c^0 とする．c^0 を初期状況とする，任意の実行 $\mathcal{E} = c^0, c^1, \cdots$ において，述語 Y を満たす状況 c^i $(i \geq 0)$ が存在する．
2. 述語 Y を満たす任意の状況を c'^0 とする．c'^0 を初期状況とする，任意の実行 $\mathcal{E}' = c'^0, c'^1, \cdots$ において，すべての状況 c'^i が述語 Y を満たす． □

$X \triangleright Y$ は，述語 X を満たす任意の状況から始まるすべての実行に対して，システムが述語 Y を満たすような状況に到達し，いったんそのような状況に到達すると，その後のすべての状況も述語 Y を満たすことを意味する．次に，この概念を用いて自己安定プロトコルを定義する．

定義 2.2 自己安定プロトコル

$\mathcal{P}$ をプロトコルとし，$\mathcal{P}$ のすべての実行の集合を $\mathcal{A}$ とする．また，$\mathcal{SP}$ を実行の集合 $\mathcal{A}$ に対して定義される述語とする．すべての状況の集合 $\mathcal{C}$ に対して定義される述語 $\mathcal{L}$ が存在し，以下の条件を満たすとき，プロトコル $\mathcal{P}$ は $\mathcal{SP}$ に対して自己安定であるという．

1. 正当性 *:* 述語 $\mathcal{L}$ を満たす任意の状況を α とする．α を初期状況とする，任意の実行 $\mathcal{E}$ が述語 $\mathcal{SP}$ を満たす．
2. 閉包性・収束性 *:* $true \triangleright \mathcal{L}$．つまり，述語 $\mathcal{L}$ が，システムのすべての状況の集合 $\mathcal{C}$ に対する *closed attractor* である． □

定義 2.2 において，述語 $\mathcal{SP}$ は，分散システムに要求される動作を規定するものである．そこで，述語 $\mathcal{SP}$ のことをプロトコル要求と呼ぶ．また述語 $\mathcal{L}$ は，プロトコル要求を満たす実行の初期状況を規定するので，$\mathcal{L}$ を満たす状況を正当な状況と呼ぶ．

プロトコル要求 $\mathcal{SP}$ に対する自己安定プロトコルは，いずれ正当な状況に到達し，それ以降の実行は $\mathcal{SP}$ を満たす．自己安定プロトコルの安定時間を以下のように定義する．

定義 2.3 安定時間

$\mathcal{P}$ をプロトコル要求 $\mathcal{SP}$ に対する自己安定プロトコルとする．$\mathcal{P}$ の任意の同期式実行 $\mathcal{E}$ において，第 t ラウンド終了時の状況が正当な状況のとき，$\mathcal{P}$ の安定時間は t であるという． □

2.1.3 隣接間同期化問題

本章では，木ネットワークにおいて隣接プロセス間の同期を実現する自己安定プロトコルを利用する．この自己安定プロトコルのプロトコル要求 NS は次のように定義される．

定義 2.4 プロトコル要求 *NS*
任意の実行を $\mathcal{E} = c^0, c^1, \cdots$ とする．任意のプロセスを i とし，i が状況 c^s, c^t $(s < t)$ でアクションを実行した *(*アクションを実行する直前の状況を c^s, c^t*)* とする．i が $c^s, \cdots, c^t$ の間で正確に一度だけアクションを実行するなら，i の任意の隣接プロセス j も $c^s, \cdots, c^t$ の間で正確に一度だけアクションを実行する．□

これは，プロトコル要求 NS を満たす任意の実行 $\mathcal{E}$ において，あるプロセス i が実行する連続する 2 つのアクションの間に，i に隣接するすべてのプロセスが正確に一度だけアクションを実行することを意味する．

2.1.4 ヒープ順序付き木構成問題

本章では，木ネットワークの各プロセスが，初期状況で持つ値 (初期値) を並べ換えることにより，ヒープ順序 (各プロセスがその子プロセスよりも大きい値を持つ) を構成する自己安定プロトコルを提案する．ただし，プロセスの初期値は相異なるとする．以下で，ヒープ順序付き木構成問題のプロトコル要求 HP を定義する．

定義 2.5 プロトコル要求 *HP*
各プロセス i は，読出し専用の入力変数 in_i と書込専用の出力変数 out_i を持つ．
実行を $\mathcal{E}$ とする．$\mathcal{E}$ において，in_i の値が変化しなければ，out_i の値は変化せず，以下の条件を満たす．

1. $\{out_i |\ i \in V\} = \{in_i |\ i \in V\}$
2. $\forall i \in V - \{r\} :\ out_i < out_{P_i}$ □

2.2. 隣接間同期化プロトコル

木ネットワークにおける隣接間同期化プロトコル $\mathcal{NSP}$[13] を図 2.1 に示す. 各プロセス i は, 論理型変数 col_i を持つ. 各プロセス i は, 親プロセスと子プロセスの 変数 col を読む. そして,col_i と親プロセスの col_{P_i} が異なる値であり, かつ, すべての子プロセスの col が col_i と同じ場合のみ動作し,col_i の値を反転させる. これより, プロセスとその隣接プロセスが交互に動作することを実現している. なお, 根プロセスは親プロセスとの比較をせず, 葉プロセスは子プロセスとの比較をしない点以外は, 他のプロセスと同一である.

(定数) Cld_i：子の集合，P_i：親のプロセス

(変数) col_i：論理型変数

For the root
S_1 :: $\forall j \in Cld_i$: $col_j = col_i$ $\rightarrow$ $col_i := \neg col_i$

For the internal processes
S_2 :: $col_{P_i} \neq col_i \wedge (\forall j \in Cld_i : col_j = col_i)$
$\rightarrow$ $col_i := col_{P_i}$

For the leaf processes S_3 :: $col_{P_i} \neq col_i$ $\rightarrow$ $col_i := col_{P_i}$

図 2.1 隣接間同期化自己安定プロトコル $\mathcal{NSP}$（プロセス i）

定理 2.1 *[13]* プロトコル $\mathcal{NSP}$ は, 安定時間 0, 各プロセスの領域計算量 $O(1)$ の隣接間同期化自己安定プロトコルである. □

Johnen らは，このプロトコル $\mathcal{NSP}$ が任意の実行において，プロトコル要求 NS を満たすことを示したが，実際にプロセスが動作を始めるまでの時間は評価していない．そこで，このプロトコル $\mathcal{NSP}$ の任意の同期式実行において，第 $2h$ ラウンド終了時以降, 各プロセスが 2 ラウンドに 1 度, 正確にアクションを実行することを示す.

本節では，以降次の表記を用いる．

$d(i)$: プロセス i の深さ (根からの距離)

$col_i(c)$: 状況 c における変数 col_i の値

プロセス集合 $R_\ell = \{i | d(i) \leq \ell\}$ $(0 \leq \ell \leq h)$

プロセス集合 $L_\ell = \{i | d(i) \geq \ell\}$ $(0 \leq \ell \leq h)$

補題 2.1 プロトコル $\mathcal{NSP}$ の任意の同期式実行 $\mathcal{E}$ を考える．任意の $k \geq h - \ell (0 \leq \ell \leq h)$，および，葉以外の任意のプロセス $i \in L_\ell$ について，第 k ラウンド終了時の状況 c^k において，次が成立する．

- i が根プロセスで，第 k ラウンドでアクションを実行しないならば，$\forall j \in Cld_i : col_j(c^k) = col_i(c^k)$
- i が内部プロセスならば，$col_i(c^k) = col_{P_i}(c^k) \vee (\forall j \in Cld_i : col_j(c^k) = col_i(c^k))$

証明
ℓ についての帰納により示す.

- $\ell = h$ の場合．プロセス $i \in L_h$ は葉なので考えなくてよい.
- $\ell + 1 (\leq h)$ について題意が成立すると仮定する.

  つまり，任意の内部プロセス $j \in Cld_i$，および，任意の $k' \geq h - (\ell + 1)$ について，$col_j(c^{k'}) = col_i(c^{k'}) \vee (\forall j' \in Cld_j : col_{j'}(c^{k'}) = col_j(c^{k'}))$ が成立することになる.

  プロセス $i \in L_\ell$ が内部プロセスの場合と根プロセスの場合で，次のように背理法の仮定をおく，

  - 内部プロセス i について，
    $col_i(c^k) \neq col_{P_i}(c^k) \wedge (\exists j \in Cld_i : col_j(c^k) \neq col_i(c^k))$ と仮定する．定理 2.1により，$col_i(c^k) \neq col_{P_i}(c^k)$ は，プロセス i が第 k ラウンドにアクションを実行しないことを意味する.
  - 根プロセス i について，$\exists j \in Cld_i : col_j(c^k) \neq col_i(c^k)$ と仮定する．
    根プロセス i が第 k ラウンドにアクションを実行しないときのみを考えればよい.

  いずれの場合もプロセス i は第 k ラウンドにアクションを実行せず，
  $col_i(c^{k-1}) = col_i(c^k)$ が成立することになり，$col_j(c^k) \neq col_i(c^k)$ であるようなプロセス $j \in Cld_i$ が存在する．ここで，j について次の 2 通りの場合に分けられる.

  - プロセス j が，第 k ラウンドにおいてアクションを実行する場合．プロトコルより $col_j(c^k) = col_i(c^k)$ であるから，プロセス j の仮定に矛盾する.
  - プロセス j が，第 k ラウンドにおいてアクションを実行しない場合．
    アクションを実行しないので $col_j(c^{k-1}) = col_j(c^k)$ である．また前述より，$col_i(c^{k-1}) = col_i(c^k) \neq col_j(c^k)$ なので，$col_j(c^{k-1}) \neq col_i(c^{k-1})$ が成立．j が葉プロセスの場合，状況 c^{k-1} でガードが偽であることと矛盾する．一方，j が内部プロセスの場合，
    $\exists j' \in Cld_j : col'_j(c^{k-1}) \neq col_j(c^{k-1})$ が成立することになり，これは帰納法の仮定に矛盾する.

  従って，プロセス $i \in L_\ell$ が条件を満たすことが言える．以上の帰納法により，題意は満たされた. ■

補題 2.2 プロトコル $\mathcal{NSP}$ の任意の同期式実行 $\mathcal{E}$ を考える．任意の $k \geq h+\ell (0 \leq \ell \leq h)$，および，根以外の任意のプロセス $i \in R_\ell$ について，第 k ラウンド終了時の状況 c^k において，次が成立する．

- i が内部プロセスならば，$col_i(c^k) \neq col_{P_i}(c^k) \vee (\forall j \in Cld_i : col_j(c^k) \neq col_i(c^k))$
- iが葉プロセスで，第 k ラウンドでアクションを実行しないならば，$col_i(c^k) \neq col_{P_i}(c^k)$

証明
ℓ についての帰納により示す．

- $\ell = 0$ のとき．プロセス i は根プロセス r なので考えなくてよい．
- $\ell - 1 (\geq 0)$ について題意が成立すると仮定する．

  つまり，任意の内部プロセス $j \in R_{\ell-1}$，および，任意の $k' \geq h + (\ell - 1)$ について，$col_{P_i}(c^{k'}) \neq col_{P_{P_i}}(c^{k'}) \vee (\forall j \in Cld_{P_i} : col_j(c^{k'}) \neq col_{P_i}(c^{k'}))$ が成立することになる．

  プロセス $i \in R_\ell$ が内部プロセスの場合と葉プロセスの場合で，次のように背理法の仮定をおく，

  - 内部プロセス i について，
    $col_i(c^k) = col_{P_i}(c^k) \wedge (\exists j \in Cld_i : col_j(c^k) = col_i(c^k))$ と仮定する．プロセス $i \in R_\ell$ が第 k ラウンドにアクションを実行すると仮定すると，定理 2.1よりすべての隣接プロセス $j \in N_i$ はアクションを実行せず，プロトコルより $\forall j \in Cld_i : col_j(c^k) \neq col_i(c^k)$ となり矛盾する．従って，プロセス i が第 k ラウンドにアクションを実行しない場合を考える．
  - 葉プロセス i について，$col_i(c^k) = col_{P_i}(c^k)$ と仮定する．補題の仮定より，プロセス i は第 k ラウンドにアクションを実行しない．

  いずれの場合もプロセス i が第 k ラウンドにアクションを実行せず，
  $col_i(c^{k-1}) = col_i(c^k)$ が成立する．また $col_i(c^k) = col_{P_i}(c^k)$ が成立する．ここで，P_i について次の 2 通りの場合に分けられる．

  - プロセス P_i が，第 k ラウンドにおいてアクションを実行する場合．
    プロトコルより $col_{P_i}(c^k) \neq col_i(c^k)$ であるから，背理法の仮定に矛盾する．
  - プロセス P_i が，第 k ラウンドにおいてアクションを実行しない場合．
    アクションを実行しないので $col_{P_i}(c^{k-1}) = col_{P_i}(c^k)$ である．また前述より，$col_i(c^{k-1}) = col_i(c^k) = col_{P_i}(c^k)$ なので，
    $col_{P_i}(c^{k-1}) = col_i(c^{k-1})$ が成立．

P_i が根プロセスの場合，第 $k-1(\geq h)$ ラウンドにアクションを実行したとすると，$\forall j \in Cld_{P_i} : col_j(c^{k-1}) \neq col_{P_i}(c^{k-1})$ となって矛盾する．第 $k-1$ ラウンドにアクションを実行しないとすると，補題 2.1 より $\forall j \in Cld_{P_i} : col_j(c^{k-1}) = col_{P_i}(c^{k-1})$ となり，状況 c^{k-1} でガードが偽であることに矛盾する．

一方 P_i が内部プロセスの場合，帰納法の仮定 $col_{P_i}(c^{k-1}) \neq col_{P_{P_i}}(c^{k-1}) \vee (\forall j \in Cld_{P_i} : col_j(c^{k-1}) \neq col_{P_i}(c^{k-1}))$ から，$col_{P_i}(c^{k-1}) \neq col_{P_{P_i}}(c^{k-1})$ が言える．状況 c^{k-1} において P_i のガードが偽であるから，$\exists j \in Cld_{P_i} : col_j(c^{k-1}) \neq col_{P_i}(c^{k-1})$ であるが，これは補題 2.1に矛盾する．

従って，プロセス $i \in R_\ell$ が条件を満たすことが言える．以上の帰納法により，題意は満たされた． ■

定理 2.2 プロトコル $\mathcal{NSP}$ の任意の同期式実行を $\mathcal{E}$ とする．$\mathcal{E}$ の第 $2h$ ラウンド終了時以降, 各プロセスは 2 ラウンドに 1 度, 正確にアクションを実行する． □

証明
補題 2.1，補題 2.2から，任意のプロセス $i \in R_\ell$ と任意の $k \geq 2h$ について次が成立することが言える．

- i が根プロセスで，第 k ラウンドでアクションを実行しないならば，$\forall j \in Cld_i : col_j(c^k) = col_i(c^k)$
- i が内部プロセスならば，$(col_i(c^k) = col_{P_i}(c^k) \wedge (\forall j \in Cld_i : col_j(c^k) \neq col_i(c^k))) \vee (col_i(c^k) \neq col_{P_i}(c^k) \wedge (\forall j \in Cld_i : col_j(c^k) = col_i(c^k)))$
- i が葉プロセスで，第 k ラウンドでアクションを実行しないならば，$col_i(c^k) \neq col_{P_i}(c^k)$

つまり，第 $k \geq 2h$ ラウンドにおいてアクションを実行しないプロセスのガードはラウンド終了時に真となることが言えるので，題意は成立する． ■

実際に初期状況から $2h$ ラウンドの間，アクションを実行しないプロセスが存在する場合がある．初期状況において，経路 $r, \beta_1, \beta_2, \ldots, \beta_h$ 上のプロセスについて，$col_r(c^0) = col_{\beta_1}(c^0) = \ldots = col_{\beta_h}(c^0)$ が成り立ち，かつ，経路 $r, \gamma_1, \gamma_2, \ldots, \gamma_h$ 上のプロセスについて，$\ell = 1, 3, 5, \ldots$ の場合，$col_{\gamma_\ell}(c^0) \neq col_r(c^0)$ が成り立ち，$\ell = 2, 4, 6, \ldots$ の場合，$col_{\gamma_\ell}(c^0) = col_r(c^0)$ が成り立つ場合には，プロセス β_h は初期状況から $2h$ ラウンドの間アクションを実行しない．

2.3. ヒープ順序付き木を構成する自己安定プロトコル

本節では，木ネットワークにおいてヒープ順序付き木を構成する安定時間 $O(h)$，各プロセスの領域計算量 $O(K)$ の自己安定プロトコル $\mathcal{HPP}$ を提案する．ここで，h は木の高さを，K は入力値のサイズを表す. このプロトコルは，文献 [4] のプロトコルの安定時間 $O(nh)$，領域計算量 $O(\delta K)$ を改良している．

2.3.1 プロトコル $\mathcal{HPP}$ の概略

木ネットワーク上の各プロセスが，それぞれ 1 つずつデータを持つとする．ヒープ順序を実現するためには，より大きな値を根に向かって移動させ，小さな値は葉に向かって移動させればよい．つまり，各プロセスと，その子プロセスの間で値を比較し，子プロセスの持つ値の最大値が親プロセスの持つ値より大きな場合，その親と子プロセス間で値を交換する．これを繰り返せば，やがて木ネットワーク全体でヒープ順序を実現できる．

このとき，複製や損失を起こすことなく，正しくデータ交換を行うために，本プロトコルでは，隣接間同期化プロトコルを用いる．

各プロセス i の入力変数 in_i の値は，プロトコル実行中は変化しないものとする．ヒープ順序を実現するには，前述したようにこの値を移動させなければならず，各プロセス i はそのための作業用変数 w_i, r_i を持っている．自己安定プロトコルでは，初期状況に仮定をおかないため，初期状況において，作業用変数の値がどのプロセスの入力変数 val の値とも一致しないことや，あるプロセスの入力変数の値がどのプロセスの作業変数の値とも一致しない可能性がある．そこで本プロトコルでは，作業用変数の値を並べ換えることにより，ヒープ順序を実現した後，ネットワーク全体にリセットをかけ，各プロセス i の入力変数 in_i の値を作業用変数にコピーし，再び作業用変数に対してヒープ順序の構成を繰り返す. ヒープ順序づき木が構成されると，各プロセス i は作業用変数の値を出力変数 out_i にコピーする．リセット，ヒープ順序付き木構成は繰り返し実行されるが，入力変数の値は変化しないので，2 回目以降では同じヒープ順序付き木が構成される．従って，各プロセス i は out_i に同じ値を書き込むことになり，out_i の値は変化しなくなる．

なお，異なるプロセスの入力変数 in_i の値は異なるものとする (同じ値がある場合，プロセスの識別子との二項組にすることにより，異なる値とみなせる).

2.3.2 プロトコル $\mathcal{HPP}$

ヒープ順序付き木を構成する自己安定プロトコルを図 2.2, 2.3 に示す．各プロセスのアクションにおけるガードは，隣接間同期化プロトコルのものと同じである．プロセスの動作は隣接間同期化プロトコルの動作を含んでおり，これによりプロトコル $\mathcal{HPP}$ は同期化して動作する．

次に，アクション中に含まれる定数，変数，手続きを説明する．

(定数)

P_i ： i の親
Cld_i ： i の子の集合
in_i ： i への入力値 (値はユニークで実行中変化しない)

(変数)

col_i ： 同期機構 $\mathcal{NSP}$ に基づく論理型変数
out_i ： 出力変数
w_i ： ヒープを構築するための作業用変数
r_i ： 値の交換後，子へ返す値を格納する変数
$change_i$ ： i を根とする部分木で作業変数の値の変化の有無を示す論理型変数
$reset_i$ ： 手続き **RESET** を行うかどうかを示す論理型変数

(手続き)

RESET (図 2.3)
すべてのプロセスで作業用変数の値の交換がなくなった (ヒープ順序付き木が構成された) ときに，リセットをかけるために根プロセスが実行を開始する手続き．各プロセス i は，作業用変数 w_i の値を出力変数 out_i に代入し，入力変数 in_i の値を作業用変数に代入する．

HEAPIFY (図 2.3)
ヒープ順序を実現する手続き．親プロセスと子プロセス間での値の交換を行う．各プロセス i は，すべての子プロセスの作業用変数を読み，それらの最大値を求める．最大値が自分の作業用変数の値より大きい場合は，まず自分の作業用変数の値を r_i に代入し，次に最大値を自分の作業用変数に代入する．なお，値の交換が起こらなかった場合は，r_i に特別な記号 $\perp$ を代入する．子プロセスは，親プロセスの作業用変数を読み，その値が自分の作業用変数と同じ値の場合に，値の交換が起きたと認識し，親の r の値を自分の作業用変数に代入する．

For the root
$H_1 :: \forall j \in Cld_i : \ col_j = col_i$
$\rightarrow$ **HEAPIFY**;
  if $(r_i = \perp \wedge \ (\forall j \in Cld_i : change_j = false))$
    /* **heap order accomplished** */
    $reset_i := true$; **RESET**; $change_i := true$;
  else
    $reset_i := false$; $change_i := true$;
  $col_i := \neg col_i$;

For the internal processes
$H_2 :: col_{P_i} \neq col_i \ \wedge \ (\forall j \in Cld_i : \ col_j = col_i)$
$\rightarrow$ **if** $(reset_{P_i} = false)$
    $reset_i := false$; **HEAPIFY**;
    if $(r_i = \perp \wedge \ (\forall j \in Cld_i : change_j = false))$
      $change_i := false$;
    else /* **heap disorder in the subtree rooted** i */
      $change_i := true$;
  else
    $reset_i := true$; **RESET**; $change_i := true$;
  $col_i := col_{P_i}$;

For the leaf processes
$H_3 :: col_{P_i} \neq col_i$
$\rightarrow$ **if** $(reset_{P_i} = false)$
    $reset_i := false$; **HEAPIFY**; $change_i := false$;
  else
    $reset_i := true$; **RESET**; $change_i := true$;
  $col_i := col_{P_i}$;

図 2.2 プロトコル $\mathcal{HPP}$（プロセス i）

RESET:
$out_i := w_i$; $w_i := in_i$; $r_i = \perp$;

HEAPIFY:
For the root
$max_i := \max\{w_j | \ j \in Cld_i\}$;
if $(w_i < max_i)$ /* **a child has a bigger value** */
$r_i := w_i$; $w_i := max_i$; /* **exchange values** */
else
$r_i := \perp$;

For the internal processes
if $(w_i = w_{P_i} \ \wedge \ r_{P_i} \neq \perp)$ /* **parent of** i **picked up the value of** i */
$w_i := r_{P_i}$; /* **copy the returned value** */
$max_i := \max\{w_j | \ j \in Cld_i\}$;
if $(w_i < max_i)$
$r_i := w_i$; $w_i := max_i$;
else
$r_i := \perp$;

For the leaf processes
if $(w_i = w_{P_i} \ \wedge \ r_{P_i} \neq \perp)$
$w_i := r_{P_i}$;
$r_i := \perp$;

図 2.3 プロトコル $\mathcal{HPP}$ の手続き **RESET**，**HEAPIFY**（プロセス i）

以下では，プロトコル $\mathcal{HPP}$ は，安定時間 $O(h)$，領域計算量 $O(K)$ の自己安定ヒープ順序づき木構成プロトコルであることを示す．
以下では，次の表記を用いる．

$d(i)$: プロセス i の深さ (根からの距離)．

$\mathrm{Acted}(c^i)$: 任意の実行 $\mathcal{E} = c^0, c^1, \ldots$ について，状況 c^{i-1}, c^i $(i \geq 1)$ の間にアクションを実行したプロセスの集合．

$var_i(c)$: 状況 c における，変数 var_i の値．

プロトコル $\mathcal{HPP}$ では，隣接間同期化プロトコルを利用している．同期化プロトコルは，非同期式システムで同期式システムを模倣するためのものである．この同期式実行を用いて $\mathcal{HPP}$ の正当性を証明する．まず，同期式実行として理想実行を定義する．また，自己安定システムでは初期状況に仮定を置かないため，

プロセスの変数には通常の動作からは得られないような値が入っている可能性がある．初期状況において，このような値を持つ変数が存在しないように，理想実行を定義する．

定義 2.6 (理想実行) 実行 $\mathcal{E} = c^0, c^1, \ldots$ が以下の条件を満たすとき，$\mathcal{E}$ を理想実行という．

- 任意の状況 $c \neq c^0$ について，
  $\mathrm{Acted}(c) = \{ i \in V \mid d(i) \bmod 2 \neq 0 \} \vee \mathrm{Acted}(c) = \{ i \in V \mid d(i) \bmod 2 = 0 \}$ が成立．
  便宜上，$\mathrm{Acted}(c^0) = V - \mathrm{Acted}(c^1)$ と定義する．
- 初期状況 c^0 において，以下が成立．
  - $\forall i \in V :\ reset_i(c^0) = false$
  - $\forall i \in V :\ r_i(c^0) < w_i(c^0) \vee r_i(c^0) = \perp$
  - $\forall i \in \mathrm{Acted}(c^0) :\ r_i(c^0) \neq \perp \Longrightarrow\ \exists j \in Cld_i :\ w_j(c^0) = w_i(c^0)$
  - $\forall i \notin \mathrm{Acted}(c^0) :\ r_i(c^0) \neq \perp \Longrightarrow \exists j \in Cld_i : w_j(c^0) = r_i(c^0) \vee r_j(c^0) = r_i(c^0)$ □

次にプロトコル $\mathcal{HPP}$ の任意の実行 $\mathcal{E}$ に対応する理想実行を定める．そのために，各プロセスの $\mathcal{E}$ に現れる状態の n 項組として以下で実行断面を定義する．この実行断面を状況であると思えば，実行断面の系列において，理想実行と一致するような接尾部が存在する．

定義 2.7 任意の実行を $\mathcal{E} = c^0, c^1, \ldots$ とする．任意のプロセス i，任意の状況 c^j について，以下を定義する．

$\mathrm{Enable}(i, c^j)$: 状況 c^j において，ガードが真の i のアクションが存在するかどうかを表す真偽値．

$\mathrm{LastActed}(i, c^j)$: $k \leq j \ \wedge\ i \in \mathrm{Acted}(c^k)$ を満たす状況 c^k のうち最後のもの．

$\mathrm{NxtActed}(i, c^j)$: $k > j \ \wedge\ i \in \mathrm{Acted}(c^k)$ を満たす状況 c^k のうち最初のもの．

$\mathrm{NxtEnable}(i, c^j)$: $k \geq j \ \wedge\ i \in \mathrm{Enable}(i, c^k)$ を満たす状況 c^k のうち最初のもの．
□

定義 2.8 任意の実行 $\mathcal{E}$ について，すべてのプロセスが少なくとも *1* 回アクションを実行した状況を c とする．根 r に関する状況の系列 $c_r^0, c_r^1, c_r^2, \ldots, c_r^t, \ldots$ を次のように定義する．ただし，便宜上 $c_r^{-1} = c$ とする．

- $\mathrm{Enable}(r, c_r^{t-1}) \Longrightarrow c_r^t = \mathrm{NxtActed}(r, c_r^{t-1})$
- $\neg\mathrm{Enable}(r, c_r^{t-1}) \Longrightarrow c_r^t = \mathrm{NxtEnable}(r, c_r^{t-1})$

上記の系列 $c_r^0, c_r^1, \ldots$ に対し，実行断面 $(\sigma^0(c_0^t), \ldots, \sigma^{n-1}(c_{n-1}^t))$ $(t = 0, 1, \ldots)$ を次のように定義する．ただし，$c' = (s_0, \ldots, s_{n-1})$ に対し，$\sigma^i(c') = s_i$ とする．

- $P_i \in \mathrm{Acted}(c_{P_i}^t) \Longrightarrow c_i^t = \mathrm{NxtEnable}(i, c_{P_i}^t)$
- $P_i \notin \mathrm{Acted}(c_{P_i}^t) \Longrightarrow c_i^t = \mathrm{LastActed}(i, c_{P_i}^t)$

定義 2.9 任意の実行 $\mathcal{E}$ と任意のプロセス $i \in V$ について，以下で定義される i の状態の系列 $s_i^0, s_i^1, \ldots$ を $\mathcal{E}$ の i への射影といい，$\mathrm{Proj}(\mathcal{E}, i)$ と表す．

- s_i^0 は $\mathcal{E}$ の初期状況における i の状態．
- s_i^j は $\mathcal{E}$ においてプロセス i が j 番目のアクションを実行した直後の i の状態． □

補題 2.3 任意の実行 $\mathcal{E}$ について，次を満たす理想実行 $\mathcal{E}_{\mathcal{I}}$ が存在する．
各プロセス $i \in V$ について，$\mathrm{Proj}(\mathcal{E}_i', i) = \mathrm{Proj}(\mathcal{E}_{\mathcal{I}}, i)$ であるような $\mathcal{E}$ の接尾部 $\mathcal{E}_i'$ が存在する．
さらに，$\mathcal{E}$ が同期式実行なら，$\mathcal{E}_i'$ の初期状況は $\mathcal{E}$ の $4h$ ラウンド以内に現れる．

証明 定理 2.2 より，実行 $\mathcal{E}$ においてすべてのプロセスはいずれアクションを行い，$\mathcal{E}$ が同期実行ならば $2h$ ラウンド以内にすべてのプロセスが少なくとも 1 回アクションを行う．さらに，すべてのプロセス $i \in V$ について $reset_i(c) = false$ が成り立つ状況 c が存在し，$\mathcal{E}$ が同期実行ならば c は初期状況から $4h$ ラウンド以内に現れる．
$\mathcal{E}$ の実行断面 $(\sigma_0(c_0^t), \ldots, \sigma_{n-1}(c_{n-1}^t))$ のなかで，根からの距離が h の任意のプロセス i について，c_i^t が c から到達可能であるような実行断面が存在する．$\mathcal{E}$ が同期実行ならば $c_0^t = c_1^t = \cdots = c_{n-1}^t = c$ である．
上記の実行断面に含まれる $c_0^t, c_1^t, \ldots, c_{n-1}^t$ について，$\mathcal{E}$ の c_i^t から始まる接尾部を $\mathcal{E}_i'$ とし，$\mathrm{Proj}(\mathcal{E}_i', i) = s_i^0, s_i^1, \ldots$ とすると，実行 $\mathcal{E}_{\mathcal{I}} = c^0, c^1, \ldots$ (ただし，$c^\ell = (s_0^\ell, s_1^\ell, \ldots, s_{n-1}^\ell)$ $(\ell = 0, 1, 2, \ldots)$) は理想実行であり，題意を満たす． ■

補題 2.3 より，プロトコルの正当性と時間計算量の評価は，理想実行についてのみ考えればよい．

定義 2.10 任意の理想実行 $\mathcal{E}$ における 任意の状況を c とする．値の集合 $Vals(c)$ を次のように定義する．

$Vals(c) = W(c) \cup R(c)$
ただし，

- $W(c) = \{\langle w_i(c), i\rangle \mid i \in \mathrm{Acted}(c) \vee\ w_i(c) \neq w_{P_i}(c)\ \vee\ r_{P_i}(c) = \perp\}$
- $R(c) = \{\langle r_{P_i}(c), i\rangle \mid i \notin \mathrm{Acted}(c) \wedge\ w_i(c) = w_{P_i}(c)\ \wedge\ r_{P_i}(c) \neq \perp\}$ □

各プロセス i に対し，$\langle v,i\rangle \in \mathit{Vals}(c)$ となる v は正確に一つ定まる．この v を i が状況 c で持つ値と考える．($\langle v,i\rangle \in R(c)$ のとき，実際には v は r_{P_i} の値であるが，r_{P_i} が i に戻す値なので，i が持つ値を v と考える．) この値の移動は次のように定義する．

定義 2.11 (値 v の移動) 任意の連続する状況を c, c'，任意のプロセスを i とし，$\langle v,i\rangle \in \mathit{Vals}(c)$ とする．$\langle v,i\rangle$ は以下の $\langle v',j\rangle \in \mathit{Vals}(c')$ に移動したといい，$\langle v,i\rangle \Rightarrow_{c'} \langle v',j\rangle$ と表す．

- $i \notin \mathrm{Acted}(c')$ の場合
  - *(a)* $r_{P_i}(c') \neq \bot \wedge w_i(c') = w_{P_i}(c')$ *(P_i が i と値を交換)* の場合，$\langle v',j\rangle = \langle v,P_i\rangle$
  - *(b)* その他の場合，$\langle v',j\rangle = \langle v,i\rangle$
- $i \in \mathrm{Acted}(c')$ の場合
  - *(c)* $r_i(c') \neq \bot$ *(i が子と値を交換)* の場合，$w_k(c') = w_i(c')$ となる $k \in \mathit{Cld}_i$ に対し，$\langle v',j\rangle = \langle v,k\rangle$
  - *(d)* $\mathit{reset}_i(c') = \mathit{true}$ *(i が* **RESET** *を実行)* の場合，$\langle v',j\rangle = \langle \mathit{in}_i,i\rangle$
  - *(e)* その他の場合，$\langle v',j\rangle = \langle v,i\rangle$ □

定義 2.11 において，(c) の場合，同じ値を持つ子が複数存在すると $\langle v,i\rangle \Rightarrow_{c'} \langle v,k\rangle$ なる k が一意に定まらない．ただし，**RESET** を一度行うとプロセスの持つ値は相異なるものとなり，$\langle v,i\rangle \Rightarrow_{c'} \langle v',j\rangle$ なる $\langle v',j\rangle$ が一意に定まる．

定義 2.12 任意の $\langle v,i\rangle \in \mathit{Vals}(c)$ に対し $\mathrm{Wrong}(\langle v,i\rangle,c)$ を次のように定義する．

$$\mathrm{Wrong}(\langle v,i\rangle,c) = \{\langle v',j\rangle \in \mathit{Vals}(c) \mid v' < v \ \wedge \ \text{“}j \text{ は } i \text{ の祖先”}\}$$ □

$|\mathrm{Wrong}(\langle v,i\rangle,c)|$ は，値 $\langle v,i\rangle$ に対しヒープ順序に違反する値の個数である．従って，すべてのプロセス i について，$|\mathrm{Wrong}(\langle v,i\rangle,c)| = 0$ が成り立てば，ヒープ順序が実現できたことになる．以下，補題 2.4 から補題 2.9 までは，$\forall i \in V$: $\mathit{reset}_i = \mathit{false}$ を満たす状況から高々 $2h$ ラウンドで作業変数上でのヒープ順序が成立することを示す．以下では，$\mathrm{Wrong}(\langle v,i\rangle,c) = \{\langle v_1,i_1\rangle,\langle v_2,i_2\rangle,\ldots,\langle v_t,i_t\rangle\}$ について，$d_s = d(i_s)$ $(1 \le s \le t)$ とおき，一般性を失うことなく $d_s > d_{s+1}$ $(1 \le s \le t-1)$ を仮定する．また，$\langle v,i\rangle \Rightarrow_{c'} \langle v,j\rangle$ としたとき，上に加えて，$\mathrm{Wrong}(\langle v,j\rangle,c') = \{\langle v'_1,i'_1\rangle,\langle v'_2,i'_2\rangle,\ldots,\langle v'_{t'},i'_{t'}\rangle\}$，$d'_s = d(i'_s)$ $(1 \le s \le t')$ とおき，$d'_s > d'_{s+1}$ $(1 \le s \le t'-1)$ を仮定する．

基本的に，各 $\langle v_s,i_s\rangle \in \mathrm{Wrong}(\langle v,i\rangle,c)$ は，ヒープ順序に違反しなくなるまで並列的に葉の方向に移動する．しかし，$d_{s-1} = d_s + 1$ のとき，つまり $i_{s-1} \in \mathit{Cld}_{i_s}$ であるときには，さらに，各 $j \in \mathit{Cld}_{i_s}$ とその値 $\langle v^j,j\rangle$ について $v^j < v_s$ であれば $\langle v_s,i_s\rangle$ は移動できない．状況 c において各 $\langle v_s,i_s\rangle$ $(1 \le s \le m)$ が並列的に移

動できるような最大の m を $\mathrm{Top}(\langle v,i\rangle,c)$ と表す．形式的には以下のように定義でき，補題 2.9 では，h ラウンド以内にすべての値が並列的に移動するようになり，$2h$ ラウンド以内にヒープ順序に違反する値がなくなることを示している．

定義 2.13 任意の状況 c, 任意の $\langle v,i\rangle \in \mathit{Vals}(c)$ について，
$\mathrm{Wrong}(\langle v,i\rangle,c)=\{\langle v_1,i_1\rangle,\langle v_2,i_2\rangle,\ldots,\langle v_t,i_t\rangle\}$ とする．$\mathrm{Top}(\langle v,i\rangle,c)$ を次のように定義する．

- $\mathrm{Wrong}(\langle v,i\rangle,c)=\emptyset$ の場合
  $\mathrm{Top}(\langle v,i\rangle,c)=0$
- $\mathrm{Wrong}(\langle v,i\rangle,c)\neq\emptyset$ の場合
  $\mathrm{Top}(\langle v,i\rangle,c)=\max(\{m|\ \forall s,1<s\leq m\leq t: d_{s-1}\geq d_s+2\}\cup\{1\})$ □

補題 2.4 任意の連続する状況を c,c' とし，$\langle v,i\rangle \Rightarrow_{c'} \langle v,j\rangle$ とする．$t=|\mathrm{Wrong}(\langle v,i\rangle,c)|$, $t'=|\mathrm{Wrong}(\langle v,j\rangle,c')|$ とすると，$t\geq t' \ \wedge\ d_t\leq d'_{t'}$ が成立する．

証明 $x\Rightarrow_{c'} y$ である値 $x\in\mathit{Vals}(c)$, $y\in\mathit{Vals}(c')$ について，以下の 3 つの集合を考える．

- $W^0=\{(x,y)|\ x\in\mathrm{Wrong}(\langle v,i\rangle,c),\ y\in\mathrm{Wrong}(\langle v,j\rangle,c')\}$
- $W^-=\{(x,y)|\ x\in\mathrm{Wrong}(\langle v,i\rangle,c),\ y\notin\mathrm{Wrong}(\langle v,j\rangle,c')\}$
- $W^+=\{(x,y)|\ x\notin\mathrm{Wrong}(\langle v,i\rangle,c),\ y\in\mathrm{Wrong}(\langle v,j\rangle,c')\}$

$t\geq t'$ については $|W^-|\geq|W^+|$ を示せば良い．ここで，定義 2.11より W^+ の各要素 $(\langle v'_{s'},\ell\rangle,\langle v'_{s'},i'_{s'}\rangle)\in W^+$ $(\ell\in\mathit{Cld}_{i'_{s'}})$ に対して，$\langle v_s,i_s\rangle\Rightarrow_{c'}\langle v_s,\ell\rangle$ $(v_s<v'_{s'},i_s=i'_{s'})$ である $(\langle v_s,i_s\rangle,\langle v_s,\ell\rangle)\in W^-$ が存在する．(つまり，$c\mapsto c'$ で v_s と v'_s が交換された) W^+ の異なる要素に対して，対応する W^- の要素も異なるので $|W^-|\geq|W^+|$ が成立．また，$d_t\leq d'_{t'}$ は明らか． ■

補題 2.4 は，値 v に対して $|\mathrm{Wrong}(\langle v,i\rangle,c)|$ が単調非増加であることを意味している．以下補題 2.9 まで，手続き **HEAPIFY** によって $|\mathrm{Wrong}(\langle v,i\rangle,c)|$ はいずれ減少していくことを示す．

補題 2.5 任意の連続する状況を c,c' とし，$\langle v,i\rangle\Rightarrow_{c'}\langle v,j\rangle$ とする．$m=\mathrm{Top}(\langle v,i\rangle,c)$ とすると，経路 $i,\ldots,i_m$ 上の任意の頂点 ℓ について，次が成り立つ．ただし，$\langle v^\ell,\ell\rangle\in\mathit{Vals}(c')$ とする．
$\langle v^\ell,\ell\rangle\in\mathrm{Wrong}(\langle v,j\rangle,c') \implies \ell\notin\mathrm{Acted}(c')$

証明 経路 $i,\ldots,\ell$ 上の ℓ の子を $j_\ell\in\mathit{Cld}_\ell$ とおく．背理法により $\ell\in\mathrm{Acted}(c')$ と仮定する．定義 2.11 より，$\langle v^\ell,\ell\rangle\Rightarrow_{c'}\langle v^\ell,\ell\rangle$ の場合と
$\exists j'_\ell\in\mathit{Cld}_\ell:\langle v^\ell,j'_\ell\rangle\Rightarrow_{c'}\langle v^\ell,\ell\rangle$ の場合とに分けられる．

- $\langle v^\ell, \ell\rangle \Rightarrow_{c'} \langle v^\ell, \ell\rangle$ の場合.
  $j_\ell \notin \mathrm{Acted}(c')$ であり，値の交換がされないので，$\langle v^{j_\ell}, j_\ell\rangle \Rightarrow_{c'} \langle v^{j_\ell}, j_\ell\rangle$ である．プロトコルと定義 2.11 より，$v^{j_\ell} \leq v^\ell$ が成立．一方，補題の仮定より $\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, j\rangle, c')$ だから，$v^\ell < v$ であり，また m の定義から，$\langle v^\ell, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \quad \vee \ \langle v^{j_\ell}, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \leq v^\ell \vee v \leq v^{j_\ell}$ が成立．従って矛盾が生じる．

- $\exists j'_\ell \in \mathit{Cld}_\ell : \langle v^\ell, j'_\ell\rangle \Rightarrow_{c'} \langle v^\ell, \ell\rangle$ の場合.

  - $j'_\ell = j_\ell$ の場合．ℓ と j_ℓ の間で値が交換されるので，$\langle v^{j_\ell}, \ell\rangle \Rightarrow_{c'} \langle v^{j_\ell}, j_\ell\rangle$ である．プロトコルと定義 2.11 より，$v^{j_\ell} < v^\ell$ が成立．また，補題の仮定より $\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, j\rangle, c')$ だから，$v^\ell < v$ である，一方 m の定義から，$\langle v^{j_\ell}, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \vee \ \langle v^\ell, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \leq v^\ell \vee v \leq v^{j_\ell}$ が成立．従って矛盾が生じる．
  - $j'_\ell \neq j_\ell$ の場合．ℓ と j'_ℓ の間で値が交換され，j_ℓ との間では値が交換されないので，$\langle v^{j'_\ell}, \ell\rangle \Rightarrow_{c'} \langle v^{j'_\ell}, j'_\ell\rangle$, $\langle v^{j_\ell}, j_\ell\rangle \Rightarrow_{c'} \langle v^{j_\ell}, j_\ell\rangle$ である．プロトコルと定義 2.11 より，$v^{j'_\ell} < v^\ell \wedge v^{j_\ell} < v^\ell$ が成立．一方，補題の仮定より $\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, j\rangle, c')$ だから，$v^\ell < v$ が成立．また m の定義から，$\langle v^{j'_\ell}, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \vee \ \langle v^{j_\ell}, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \leq v^{j'_\ell} \vee v \leq v^{j_\ell}$ が成立．従って矛盾が生じる．

いずれの場合も矛盾が生じる． ■

補題 2.6 初期状況以外の任意の状況を c とし，$m = \mathrm{Top}(\langle v, i\rangle, c)$ とする．経路 $i, \ldots, i_m$ 上の任意の頂点 ℓ について，次が成り立つ．ただし，$\langle v^\ell, \ell\rangle \in \mathit{Vals}(c)$ とする．
$\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, i\rangle, c) \implies \ell \notin \mathrm{Acted}(c)$

証明 補題 2.5 と同様にして示される．経路 $i, \ldots, \ell$ 上の ℓ の子を $j_\ell \in \mathit{Cld}_\ell$ とおく．帰納法により $\ell \in \mathrm{Acted}(c)$ と仮定する．定義 2.11 より，$\langle v^\ell, \ell\rangle \Rightarrow_c \langle v^\ell, \ell\rangle$ の場合と $\exists j'_\ell \in \mathit{Cld}_\ell : \langle v^\ell, j'_\ell\rangle \Rightarrow_c \langle v^\ell, \ell\rangle$ の場合とに分けられる．

- $\langle v^\ell, \ell\rangle \Rightarrow_c \langle v^\ell, \ell\rangle$ の場合.
  $j_\ell \notin \mathrm{Acted}(c)$ であり，値の交換がされないので，$\langle v^{j_\ell}, j_\ell\rangle \Rightarrow_c \langle v^{j_\ell}, j_\ell\rangle$ である．プロトコルと定義 2.11 より，$v^{j_\ell} \leq v^\ell$ が成立．一方，補題の仮定より $\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, i\rangle, c)$ だから，$v^\ell < v$ であり，また m の定義から，$\langle v^\ell, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \quad \vee \ \langle v^{j_\ell}, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \leq v^\ell \vee v \leq v^{j_\ell}$ が成立．従って矛盾が生じる．

- $\exists j'_\ell \in \mathit{Cld}_\ell : \langle v^\ell, j'_\ell\rangle \Rightarrow_c \langle v^\ell, \ell\rangle$ の場合.

- $j'_\ell = j_\ell$ の場合．ℓ と j_ℓ の間で値が交換されるので，$\langle v^{j_\ell}, \ell\rangle \Rightarrow_c \langle v^{j_\ell}, j_\ell\rangle$ である．プロトコルと定義 2.11 より，$v^{j_\ell} < v^\ell$ が成立．また，補題の仮定より $\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, i\rangle, c)$ だから，$v^\ell < v$ である，一方 m の定義から，$\langle v^\ell, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \vee \langle v^{j_\ell}, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \le v^\ell \vee v \le v^{j_\ell}$ が成立．従って矛盾が生じる．
- $j'_\ell \neq j_\ell$ の場合．ℓ と j'_ℓ の間で値が交換され，j_ℓ との間では値が交換されないので，$\langle v^{j'_\ell}, \ell\rangle \Rightarrow_c \langle v^{j'_\ell}, j'_\ell\rangle$, $\langle v^{j_\ell}, j_\ell\rangle \Rightarrow_c \langle v^{j_\ell}, j_\ell\rangle$ である．プロトコルと定義 2.11 より，$v^{j'_\ell} < v^\ell \wedge v^{j_\ell} < v^\ell$ が成立．また，補題の仮定より $\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, i\rangle, c)$ だから，$v^\ell < v$ が成立．一方 m の定義から，$\langle v^\ell, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \vee \langle v^{j_\ell}, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \le v^\ell \vee v \le v^{j_\ell}$ が成立．従って矛盾が生じる．

いずれの場合も矛盾が生じる． ■

系 2.1 初期状況を除く任意の状況を c とし，$m = \mathrm{Top}(\langle v, i\rangle, c)$ とすると $i_m \notin \mathrm{Acted}(c)$ が成立する． ■

補題 2.7 任意の連続する状況を c, c' とし，$\langle v, i\rangle \Rightarrow_{c'} \langle v, j\rangle$ とする．$m = \mathrm{Top}(\langle v, i\rangle, c)$, $m' = \mathrm{Top}(\langle v, j\rangle, c')$ とすると，次が成立する．
$0 < m < t \ \wedge \ i_m \in \mathrm{Acted}(c') \implies d_m > d'_{m'}$

証明 補題の仮定より，$i_{m+1} = P_{i_m} \notin \mathrm{Acted}(c')$ なので，定義 2.11 より，$\langle v_{m+1}, i_{m+1}\rangle \Rightarrow_{c'} \langle v_{m+1}, i_{m+1}\rangle$ と $\langle v_{m+1}, i_{m+1}\rangle \Rightarrow_{c'} \langle v_{m+1}, P_{i_{m+1}}\rangle$ の場合とに分けられる．

- $\langle v_{m+1}, i_{m+1}\rangle \Rightarrow_{c'} \langle v_{m+1}, i_{m+1}\rangle$ の場合，m の定義より，$\langle v_{m+1}, i_{m+1}\rangle \in \mathrm{Wrong}(\langle v, j\rangle, c')$ は明らか．
- $\langle v_{m+1}, i_{m+1}\rangle \Rightarrow_{c'} \langle v_{m+1}, P_{i_{m+1}}\rangle$ の場合，$\langle v', P_{i_{m+1}}\rangle \Rightarrow_{c'} \langle v', i_{m+1}\rangle$ とおける．$v' < v_{m+1} < v$ なので，$\langle v', i_{m+1}\rangle \in \mathrm{Wrong}(\langle v, j\rangle, c')$ である．

いずれの場合も $\langle v', i_{m+1}\rangle \in \mathrm{Wrong}(\langle v, j\rangle, c')$ である．従って，ある m'' に対して $i'_{m''} = i_{m+1}$ となり，$d'_{m''} < d_m$ が成立．また経路 $j, \ldots, i_m$ 上の任意の頂点 $\ell \in \mathrm{Acted}(c')$ について，補題 2.5 より，$\langle v^\ell, \ell\rangle \notin \mathrm{Wrong}(\langle v, j\rangle, c')$ だから，$\forall s, 1 \le s \le m'' : d'_{s+1} \le d'_s - 2$ が成立．ここで，$d'_{m'} \le d'_{m''}$ だから，$d'_{m'} < d_m$ が成立する． ■

補題 2.8 任意の連続する状況を c, c' とし，$\langle v, i\rangle \Rightarrow_{c'} \langle v, j\rangle$ とする．$m = \mathrm{Top}(\langle v, i\rangle, c)$, $m' = \mathrm{Top}(\langle v, j\rangle, c')$ とすると，次が成立する．
$m = t > 0 \ \wedge \ i_t \in \mathrm{Acted}(c') \implies m' = t' \ \wedge \ d_t < d'_{t'}$

証明 $d_t = d'_{t'}$ つまり $i_t = i'_{t'}$ と仮定する．補題の仮定から，補題 2.5 より $\langle v'_{t'}, i'_{t'}\rangle \notin \mathrm{Wrong}(\langle v, j\rangle, c')$ が成立．これは，t' の定義に矛盾する．従って，$d_t \neq d'_{t'}$ が成り立ち，補題 2.4 より，$d_t \leq d'_{t'}$ だから $d_t < d'_{t'}$ が成り立つ．同時に，補題 2.5 より $m' = t'$ が成り立つ． ■

補題 2.9 任意の理想実行を $\mathcal{E}$ とする．任意の ℓ $(0 \leq \ell \leq h)$, 任意の $\langle v, i\rangle \in Vals(c)$ に対して，初期状況から $2h - \ell$ ラウンド以内に $|\mathrm{Wrong}(\langle v, i\rangle, c)| \leq \ell$ が成立する．

証明 まず，任意の状況 $c^\ell (0 \leq \ell \leq h)$ における Wrong，Top の値をそれぞれ，$t^\ell = |\mathrm{Wrong}(\langle v, i\rangle, c^\ell)|$, $m^\ell = \mathrm{Top}(\langle v, i\rangle, c^\ell)$ とし，それに応じて d_{t^ℓ},d_{m^ℓ} を定義する．
まず，任意の状況 $\ell \leq h$ において $d_{m^\ell} - d_{t^\ell} \leq h - \ell$ を満たすことを帰納的に示す．

- $\ell = 0$ のとき，$d_{m^0} - d_{t^0} \leq h$ が満たされることは自明である．
- $\ell - 1 (\geq 0)$ において，$d_{m^{\ell-1}} - d_{t^{\ell-1}} \leq h - (\ell - 1)$ が満たされるならば，補題 2.4より，$t^{\ell-1} \geq t^\ell \wedge d_{t^{\ell-1}} \leq d_{t^\ell}$，補題 2.7より，
  $0 < m^{\ell-1} < t^{\ell-1} \wedge i_{m^{\ell-1}} \in \mathrm{Acted}(c^\ell) \Rightarrow d_{m^{\ell-1}} > d_{m^\ell}$，系 2.1と理想実行の仮定より，$i_{m^{\ell-1}} \in \mathrm{Acted}(c^\ell)$ であるから，$d_{m^\ell} - d_{t^\ell} \leq h - \ell$ が満たされる．

従ってある状況 $c^{\ell''} (\ell'' \leq h)$ が存在し，$m^{\ell''} = t^{\ell''}$ が満たされる．第 ℓ'' ラウンド終了時以降，ヒープ順序に違反するすべての値が並行的に移動する．
次に，任意のラウンド $h + \ell (\leq 2h)$ 終了時の状況において，$d_{t^{h+\ell}} \geq \ell$ を満たすことを帰納的に示す．

- $\ell = 0$ のとき，$d_{t^h} \geq 0$ は自明である．
- $\ell - 1 (\geq 0)$ において，$d_{t^{h+(\ell-1)}} \geq h + (\ell - 1)$ が満たされるならば，補題 2.8 より，$m^{h+(\ell-1)} = t^{h+(\ell-1)} > 0 \wedge i_{t^{h+(\ell-1)}} \in \mathrm{Acted}(c^{h+\ell}) \Rightarrow m^{h+\ell} = t^{h+\ell} \wedge d_{t^{h+(\ell-1)}} < d_{t^{h+\ell}}$，系 2.1と理想実行の仮定より，$i_{m^{h+(\ell-1)}} \in \mathrm{Acted}(c^{h+\ell})$ であるから，$d_{t^{h+\ell}} \geq \ell$ が満たされる．

従って，$t^{h+\ell} \leq h - d_{t^{h+\ell}} \leq h - \ell$ が言える．ℓ を $h - \ell$ で置き換えることで題意が示された． ■

補題 2.10 任意の理想実行 $\mathcal{E}$ について，高々 $8h$ ラウンドで

- $\forall i \in V : \ out_i < out_{P_i}$
- $\{in_i |\ i \in V\} = \{out_i |\ i \in V\}$

が成立し，以降の任意の状況で成立する．

証明
補題 2.9より，$2h$ ラウンド以内に，初期状況 c^0 における値の集合 $Vals(c^0)$ に対するヒープ順序が構成され，$\forall i \in V:\ w_i < w_{P_i}$ が成立する．その後，$h-1$ ラウンド以内に $\forall j \in Cld_r:\ change_j = false$ が成立し，根 r が **RESET** を行い $reset_r = true$ とする．$reset_r = true$ が成立してから，根 r を除くすべてのプロセスが **RESET** を行うまでに h ラウンド要する．このとき $\{in_i|\ i \in V\} = \{w_i|\ i \in V\}$ が成立．ここからさらに，上と同様にヒープ順序が構成され，すべてのプロセスが **RESET** を行うまでに $4h$ ラウンド要する．このとき，

- $\forall i \in V:\ out_i < out_{P_i}$
- $\{in_i|\ i \in V\} = \{out_i|\ i \in V\} = \{w_i|\ i \in V\}$

が成立し，これ以降の任意の状況で成立する． ■

補題 2.3, 2.10 より，次の定理 2.3 が成り立つ．

定理 2.3 プロトコル $\mathcal{HPP}$ の安定時間は $O(h)$ である． ■

定理 2.4 各プロセスの領域計算量は $O(K)$[1] *(K は入力のサイズ)*

証明　木ネットワークを構成するための情報として，各プロセスは局所的に隣接プロセスとの接続関係を保持していればよい．(各プロセスは識別子を持たない．) 従って，木ネットワークを構成するために必要な情報は，プロセスあたり δ である．ここで，δ はプロセスの次数．(さらに言えば，状態遷移モデルでは隣接プロセスの状態を一度に知ることができるため，プロトコルの中では $\log \delta$ bit のカウンタを必要としない．) $K \gg \delta$ と考えて良いので，各プロセスの領域計算量は $O(K)$ である． ■

従って次が言える．

系 2.2 プロトコル $\mathcal{HPP}$ は，安定時間 $O(h)$，領域計算量 $O(K)$ の自己安定ヒープ順序付き木構成プロトコルである． ■

[1]定数 Cld_i は分散システムによって用意されている定数であり，プロトコルの変数ではないので，領域計算量に含めていない．

2.4. 本章のまとめ

本章では，木ネットワークでヒープ順序を実現する自己安定プロトコルを考察した．このプロトコルは隣接間同期化プロトコルを利用して，安定時間 $O(h)$，領域計算量 $O(K)$ でヒープ順序を実現する． ここで，h は木の高さ，K は入力値のサイズを表す． これは既知の結果と比べ，安定時間，領域計算量を共に改善している．

今後の課題としては，分散ヒープ順序付き木に対して，挿入，および，最大値の削除を行う仕組みを実現することが挙げられる．

第 3 章

非停止永久故障に耐性を有する自己安定生成木構成プロトコル

本章では，一時故障だけでなく永久故障に耐性を持つ自己安定プロトコルを考察する．はじめに，第 3.1節において，本章で扱う分散システムのモデル，新たに導入した非停止永久故障，および，非停止永久故障の存在下での生成木構成問題の定義を行なう．第 3.2節では，非停止永久故障の存在下で生成木構成問題を解く自己安定プロトコルを提案し，その正当性を証明する．また，提案するプロトコルが安定するまでの時間計算量を評価する．最後に第 3.3節で結論と今後の課題について述べる．

3.1. 諸定義

3.1.1 分散システム

分散システムは n 個のプロセスとそれらを相互に結ぶ通信リンクからなり，無向グラフ $G = (V, E)$ によって表される．頂点集合 $V = \{p_0, p_1, \ldots, p_{n-1}\}$ はプロセスの集合を表し，辺集合 E は通信リンクの集合を表す．ここで $(p_i, p_j) \in E$ であるとき，プロセス p_i はプロセス p_j に隣接するという．プロセス p_i の隣接プロセスの集合を $N_i (\subseteq V)$ と表す．本章において，各プロセス p_i は大域的な識別子を持たないとする．各プロセス p_i は，隣接プロセス $p_j \in N_i$ を局所的なポート番号を用いて識別する．すなわち，ポート番号の集合 $\widehat{N}_i = \{0, 1, \ldots, |N_i| - 1\}$ に対して，1 対 1 対応の関数 $\Lambda_i : N_i \to \widehat{N}_i$ によって，各隣接プロセスに対応するポート番号が定まっている．

本章においても，簡単のため，各プロセスは隣接プロセスの状態を直接読むことができる状態通信モデルを対象とする．ただし提案するプロトコルは，隣接プロセスが共有レジスタを用いて通信するレジスタ通信モデル上のプロトコルへと容易に変換することができる．

各プロセス p_i は状態機械であり，状態集合 S_i，状態遷移関数 α_i の組 (S_i, α_i) で定義される．状態通信モデルでは，プロセス p_i の状態遷移関数 α_i は $\alpha_i : S_i \times (\prod_{p_j \in N_i} S_j) \to S_i$ である[1]．

3.1.2 分散システムの実行

分散システム全体の大域的な状況は，全プロセスの状態の n 項組で表す．つまり，すべての可能な状況の集合を $\mathcal{C}$ とすると，$\mathcal{C} = \prod_{p_i \in V} S_i$ である．ある状況 $c \in \mathcal{C}$ においてプロセスの部分集合 $Q \subseteq V$ が同時に動作し，システムが状況 c から状況 $c'(\in \mathcal{C})$ に変化したとする．これを，$c' = \sigma(c)$ と表す．ここで，$\Delta = (\alpha_0, \ldots, \alpha_{n-1})$ とすると，$\sigma = (\Delta, Q)$ と表され，これをステップと呼ぶ．上記において，$c = (s_0, \ldots, s_{n-1})$，$c' = (s'_0, \ldots, s'_{n-1})$ とすると，$p_i \in Q$ ならば $s'_i = \alpha_i(s_i; c/N_i)$ である．ここで c/N_i は，状況 c からプロセス $p_{j_\ell} \in N_i$ $(0 \leq \ell \leq |N_i| - 1)$ の状態 s_{j_ℓ} を集めた $|N_i|$ 項組 $(s_{j_0}, s_{j_1}, \ldots, s_{j_{|N_i|-1}})$ を表す．一方，$p_i \notin Q$ ならば $s'_i = s_i$ である．

スケジュールはプロセスの空ではない部分集合の無限系列 $Q^0, Q^1, \ldots$ $(Q^\ell \subseteq V)$ である．状況 c^0 とスケジュール $Q^0, Q^1, \ldots$ が与えられたとき，c^0 から始まるスケジュール $Q^0, Q^1, \ldots$ によるシステムの実行 $\mathcal{E}$ は，状況の無限系列 $c^0, c^1, \ldots$ で表される．ただし各 ℓ について $c^\ell = (s^\ell_0, \ldots, s^\ell_{n-1})$ とすると，$c^{\ell+1} = \sigma^\ell(c^\ell)$, $\sigma^\ell = (\Delta, Q^\ell)$ を満たす．$p_i \in Q^\ell$ のとき，プロセス p_i はステップ σ^ℓ において動作したと呼び，$s^\ell_i \neq s^{\ell+1}_i$ のとき，p_i は状態変化したと呼ぶ．ここで実行 $\mathcal{E} = c^0, c^1, \ldots$ における状況 c^0 を実行 $\mathcal{E}$ の**初期状況**と呼ぶ．

本章では自己安定プロトコルについて考察するので，初期状況に対し仮定を置かない．また，無限スケジュールを考えるが，すべてのスケジュールが公平であると仮定する．すなわち，各プロセスはスケジュールに無限にしばしば現れるものとする．

以下では，実行 $\mathcal{E} = c^0, c^1, \ldots$ の部分系列 $c^k, c^{k+1}, \ldots, c^{k'}$ を，実行断片とよび $frag(\mathcal{E}; k, k')$ と表す．また，$frag(\mathcal{E}; k, \infty)$ は接尾部 $c^k, c^{k+1}, \ldots$ を表す．

3.1.3 非停止永久故障

本章では，プロセスの故障を考える．故障プロセスとは，分散システムで定められた状態遷移関数に従わない状態遷移を行なうプロセスである．形式的には，故障プロセスを以下のように定義する．

状況 c^0 とスケジュール $Q^0, Q^1, \ldots$ に対して，状況の無限系列 $\mathcal{E} = c^0, c^1, \ldots$ を考える．ここで，$c^\ell = (s^\ell_0, \ldots, s^\ell_{n-1})$ と表し，$p_i \notin Q^\ell$ ならば $s^{\ell+1}_i = s^\ell_i$ とする．$p_i \in Q^\ell$ に対して，$s^{\ell+1}_i \neq \alpha_i(s^\ell_i; c^\ell/N_i)$ のとき，p_i は実行 $\mathcal{E}$ のステップ σ^ℓ に故障

[1] 厳密には，p_i は隣接プロセスをポート番号で識別するので，$\alpha_i : S_i \times (\prod_{j \in \widehat{N}_i} S_{j'}) \to S_i$（ただし，$p_{j'} = \Lambda_i^{-1}(j)$）である．

動作したという．実行 $\mathcal{E}$ において故障動作したプロセスを故障プロセスという．以下では，故障プロセスの集合を F と表す．

これまでに，さまざまな故障モデルが考察されている．ビザンチン故障は，故障動作に関して何も仮定しない故障モデルであり，最も大きな故障クラスである．また，停止故障は，実行のある時点以降，状態が変化しない故障モデルである．停止故障プロセスが 1 つでも存在すると，コンセンサス問題のような単純な問題でさえ解けないことが知られている [9]．これは，プロセスが停止故障していることと単に動作が遅いことが有界時間で区別できないことによる．そこで本章では，停止を許さない故障のモデルとして，故障プロセスが無限にしばしば，故障動作により状態変化する非停止永久故障を導入する．非停止永久故障は，以下のように定義される．

定義 3.1 (非停止永久故障)
スケジュール $Q^0, Q^1, \ldots$ に対して，状況の無限系列 $\mathcal{E} = c^0, c^1, \ldots$ を考える．ここで，$c^\ell = (s_0^\ell, \ldots, s_{n-1}^\ell)$ と表し，$p_i \notin Q^\ell$ ならば $s_i^{\ell+1} = s_i^\ell$ とする．p_i に対して，$p_i \in Q^\ell$ かつ $s_i^{\ell+1} \neq \alpha_i(s_i^\ell; c^\ell/N_i)$ かつ $s_i^{\ell+1} \neq s_i^\ell$ なる ℓ が無限個存在するとき，p_i は実行 $\mathcal{E}$ において非停止永久故障したという． ■

本章では，故障プロセスの故障として非停止永久故障のみを考える．ただし，故障プロセスが無限にしばしば，故障動作により状態変化したとしても，隣接プロセスがその状態変化を観測できなければ，この故障プロセスの故障は停止故障と同じになってしまう．例えば，正常なプロセス $p_i \in N_f$ が 2 度動作する間に，故障プロセス p_f が $s_f^\ell \to s_f^{\ell'} \to s_f^{\ell''}$ のように複数回状態遷移したとする．$s_f^\ell \neq s_f^{\ell'}$ であっても，$s_f^\ell = s_f^{\ell''}$ であれば，p_i は故障プロセスの状態変化を観測できない．そこで以下では，故障プロセスの故障動作による状態変化が任意の正常な隣接プロセスによって無限にしばしば観測されるという仮定を設ける．ここで，状態変化の観測は次のように定義される．

定義 3.2 (状態変化の観測)
任意の実行 $\mathcal{E} = c^0, c^1, \ldots$ を考える．以下をみたす実行断片 $frag(\mathcal{E}; k, k')$ $(k < k')$ が存在するとき，ステップ $\sigma^{k'-1}$ において，プロセス $p_j \in N_i$ がプロセス p_i の状態変化を観測するという．

- $s_i^k \neq s_i^{k'-1}$
- $p_j \in Q^k,\ p_j \in Q^{k'-1},\ p_j \notin Q^\ell\ (k < \ell < k' - 1)$ ■

3.1.4 非停止永久故障下の自己安定生成木構成問題

分散システム $G = (V, E)$ において，非停止永久故障プロセスの集合を F とする．本章では，G から故障プロセスを取り除いたネットワーク $G - F$ の生成木を構成する自己安定プロトコルを提案する．ここでは，$G - F$ の生成木を構成す

る自己安定プロトコルを定義する．ただし以下では，プロセス p_0 が根として指定されているものとする．また，p_0 は故障しないと仮定し，ネットワーク $G-F$ は連結であると仮定する．

各プロセス p_i は変数 $parent_i \in \widehat{N_i} \cup \{\perp\}$ を持つ．状況 c における変数 $parent_i$ の値を $parent_i(c)$ と表し，$T(c) = (V, A(c))$ を，V を頂点集合，$A(c) = \{(p_i, p_j) | parent_i(c) = \Lambda_i(p_j)\}$ を有向辺集合とするグラフとする $(\Lambda_i : N_i \to \widehat{N_i})$．また $T(c) - V'$ は，部分グラフ $(V - V', A(c) - (V' \times V'))$ を表す $(V' \subseteq V)$．

定義 3.3 頂点の部分集合 $V'(\subseteq V - \{p_0\})$ が与えられたとき，$T(c) - V'$ が根プロセス p_0 を根とする木であるとは，以下の条件を満たすことをいう．

- 根プロセス p_0 の出次数は *0*（$parent_0(c) = \perp$）．
- 任意の頂点 $p_i(\in V - V' - \{p_0\})$ の出次数は *1* $(parent_i(c) \neq \perp)$．
- $T(c) - V'$ において，任意の頂点 $p_i(\in V - V')$ から根 p_0 に到達可能． ■

定義 3.4 (自己安定生成木構成プロトコル)
任意の故障プロセス集合 F に対する，プロトコル $\mathcal{P}$ の任意の実行 $\mathcal{E}$ が次の条件を満たす接尾部 $frag(\mathcal{E}; k, \infty)$ を持つとき，プロトコル $\mathcal{P}$ を非停止永久故障耐性を有する自己安定生成木構成プロトコルという．

- 任意の状況 c^ℓ $(\ell \geq k)$ において，$T(c^\ell) - F$ が根プロセス p_0 を根とする木．
- 任意の状況の組 $c^\ell, c^{\ell'}$ $(k \leq \ell \leq \ell')$ について，$T(c^\ell) - F = T(c^{\ell'}) - F$． ■

3.1.5 ラウンド

生成木を構成して安定するまでの時間計算量を**安定時間**という．非同期分散システムにおいては，プロセスの動作速度，すなわち動作してから次に動作するまでのステップ数の上界がないため，安定時間をステップ数で評価できない．そこで，本章では以下に定義する**ラウンド** 数を用いて安定時間を評価する．本章で用いるラウンドは，第 2章で用いているラウンドとは定義が異なる．

定義 3.5 (ラウンド)
任意の実行 $\mathcal{E}$ が与えられたとき，

- 第 *0* ラウンドは，初期状況から始まる実行断片において，各プロセスが少なくとも *1* 回はスケジュールに現れるような最小の実行断片 $frag(\mathcal{E}; 0, \ell_0)$ である．
- 第 k ラウンド $frag(\mathcal{E}; \ell_{k-1}, \ell_k)$ が定義されたとき，第 $k+1$ ラウンドは状況 c^{ℓ_k} から始まる実行断片において，各プロセスが少なくとも *1* 回はスケジュールに現れるような最小の実行断片 $frag(\mathcal{E}; \ell_k, \ell_{k+1})$ である． ■

3.2. プロトコル

本節では，非停止永久故障耐性を有する自己安定生成木構成プロトコルを示し，その正当性の証明と安定時間の評価を行う．

3.2.1 プロトコルの概略

各正常プロセス p_i の状態遷移関数 α_i は以下で示される手続き全体により表される．つまり，正常プロセス p_i がスケジュールに現れるとき，直前の状況における隣接プロセスの状態 (諸変数の値) に従って 1 ステップで手続き全体が処理され，新たな p_i の状態が決められる．

各プロセス p_i は，p_i における局所的なポート番号の集合 $\widehat{N}_i$ を定数として持つ．また，3.1.4節において定義された変数 $parent_i$(親へのポート番号を格納) に加え，$dist_i$ と old_i の 2 変数を持つ．変数 $dist_i$ には安定状況において根からの距離が格納され，変数 old_i は p_i が前回動作したときの $parent_{parent_i}$ の値を保持するための変数である．

各プロセスは自分が根であるか否かを関数 $Root$ を用いることで判別できる．根以外の各プロセス p_i は，$dist_{parent_i}$ および $parent_{parent_i}$ の値を前回動作したときの値と比較することで，親が状態変化したかどうかを観測する．親が状態変化したことを観測すると，p_i は現在の親とは異なる隣接プロセスを新たな親として選ぶ．なぜなら，状態変化したプロセスは故障プロセスである可能性があるからである．なお，新しい親を選ぶ際には，あらかじめ決められた順序で繰返し隣接プロセスを選ぶ関数 $RRobin$ を使用する．これにより，p_i の隣接プロセスに故障プロセスが存在し，無限にしばしば状態変化が観測される場合には，故障プロセスを避けて親を選ぶことになる．

定数	$\widehat{N}_i$	隣接プロセスを表すポート番号の集合.
変数	$dist_i$	根からの距離.
	old_i	前回の親の親の値を保持するための変数.
関数	$Root$	プロセスが根 p_0 であれば真，根以外であれば偽を返す.
	$RRobin$	与えられた集合 X からラウンドロビンで集合の要素を返す．つまり X を順序集合とみなし，ℓ 回目に呼ばれたとき，$\ell \bmod |X|$ 番目の要素を返す.

---

if $(Root())$
　$parent_i := \perp$;
　$dist_i := 0$;
else if $((old_i, dist_i) \neq (parent_{parent_i}, dist_{parent_i} + 1))$
　$parent_i := RRobin(\widehat{N}_i)$;
　$(old_i, dist_i) := (parent_{parent_i}, dist_{parent_i} + 1)$;

図 3.1 非停止永久故障に耐性を有する自己安定生成木構成プロトコル

3.2.2 正当性

本節では提案したプロトコルが問題の解条件を満たすことを示す．問題の定義において導入した記法 $parent_i(c)$,$T(c)$ に加え，$dist_i(c)$,$old_i(c)$ は，それぞれ状況 c における各プロセス p_i の変数 $dist_i$,old_i の値を表すものとする．

補題 3.1 任意の実行 $\mathcal{E} = c^0, c^1, \ldots$ を考える．状況 c^k におけるグラフ $T(c^k)$ が有向閉路を含み，有向閉路に正常プロセスが含まれるならば，その有向閉路に含まれる少なくとも 1 個の正常プロセスが接尾部 $frag(\mathcal{E}; k, \infty)$ において状態変化し親を変更する．

証明　$T(c^k)$ に含まれる有向閉路を，プロセスの系列を用いて $p_{j_0}, \ldots, p_{j_{m-1}}, p_{j_m}(= p_{j_0})$ と表す．ただし，$parent_{j_\ell}(c^k) = \Lambda_{j_\ell}(p_{j_{\ell+1}})$ $(0 \leq \ell \leq m-1)$ とする．有向閉路に故障プロセスが含まれる場合は，故障の定義とプロトコルより，故障プロセスを親とする正常プロセス p_{j_ℓ} はいずれ状態変化する．このとき，ネットワーク $G-F$ の連結性の仮定より p_{j_ℓ} の次数は 2 以上なので，p_{j_ℓ} は親を変更する．有向閉路に含まれるすべてのプロセスが正常プロセスの場合，各プロセス $p_{j_\ell}(\notin F)$ $(0 \leq \ell \leq m-1)$ が c^k 以降状態変化しないのであれば，各 ℓ について $dist_{j_\ell}(c^k) = dist_{j_{\ell+1}}(c^k) + 1$ が成立．つまり $dist_{j_0}(c^k) > dist_{j_{m-1}}(c^k) > dist_{j_0}(c^k)$ が成立することになり矛盾する．従って，$dist_{j_\ell}(c^k) \neq dist_{j_{\ell+1}}(c^k) + 1$ であるようなプロセス p_{j_ℓ} が存在し，次に動作するとき状態変化する．ここで，p_{j_ℓ} の次数が 2 以上か，または，根である場合，p_{j_ℓ} が親を $p_{j_{\ell+1}}$ 以外に変更する．p_{j_ℓ} の次数が 1 で，かつ，根ではない場合．p_{j_ℓ} の次数が 1 となるのは長さ 2 の有向閉路 $p_{j_0}, p_{j_1}, p_{j_0}$ のときのみである．p_{j_ℓ} は $dist_{j_\ell}$ の値のみを変更し，$p_{j_{\ell-1}}$ が親を p_{j_ℓ} 以外に変更する． ■

補題 3.2 任意の実行 $\mathcal{E}$ において，無限にしばしば状態変化するプロセス集合を M と表す．実行 $\mathcal{E}$ に接尾部 $frag(\mathcal{E}; k, \infty)$ が存在し，任意の $\ell \geq k$ について，$T(c^\ell) - M$ は連結である．

証明　$V - M$ に含まれるプロセスが状態変化しない接尾部 $frag(\mathcal{E}; k, \infty)$ を考える．背理法により $T(c^\ell) - M (\ell \geq k)$ が 2 個以上の連結成分を含むと仮定し，根プロセス p_0 を含まない連結成分の 1 つを $\overline{T_r} = (\overline{V_r}, \overline{A_r})$ と表す．補題 3.1より $\overline{T_r}$ は閉路を含まない．一方，$\overline{T_r}$ の各プロセスの出次数が 1 だから，$T(c^\ell)$ において，$p_i \in \overline{V_r}$ かつ $p_j \in V - \overline{V_r}$ である p_i, p_j の組のうち，$(p_i, p_j) \in A(c^\ell)$ であるようなものが存在し，$\overline{T_r}$ が連結成分なので $p_j \in M$ である．p_i の次数が 2 以上の場合，プロトコルより，いずれ p_i は $parent_i$ を変更するので $p_i \notin M$ に矛盾する．p_i の次数が 1 の場合，ネットワーク $G - F$ の連結性の仮定より $p_j \notin F$ である．プロトコルより p_j はいずれ p_i を親とし，状態変化しなくなる．これは $p_j \in M$ に矛盾する．

従って $T(c^\ell) - M$ は連結であることがいえる． ■

補題 3.3 任意の実行 $\mathcal{E}$ において，無限にしばしば状態変化するプロセス集合を M と表す．このとき $F = M$ が成立する．

証明　非停止永久故障の定義より $F \subseteq M$ である．$V - M$ に含まれるプロセスが状態変化しないような E の接尾部 $frag(\mathcal{E}; k, \infty)$ を考え，背理法により $(V - F) \cap M \neq \emptyset$ と仮定する．仮定より $G - F$ は連結なので，$p_i \in M - F$ かつ $p_j \in V - M$ であるようなプロセスの組 p_i, p_j $((p_i, p_j) \in E)$ が存在する．仮定より p_i は無限にしばしば親を変更するが，プロトコルより，いずれ $V - M$ に属するプロセスを親とし状態変化しなくなることになり矛盾する．従って，$(V - F) \cap M = \emptyset$ であることがいえ，$F = M$ がいえる． ■

補題 3.2および補題 3.3 から，定理 3.1がいえる．

定理 3.1 任意の実行 $\mathcal{E}$ において，いずれグラフ $T(c) - F$ は根プロセス p_0 を根とする木となり安定する． ■

3.2.3 安定時間

次に提案プロトコルが安定するまでの時間計算量である安定時間を評価する．故障プロセスの状態変化が観測されない間，正常なプロセスが故障プロセスを避けて木を構成することができないことは自明である．時間計算量を評価するため，各故障プロセス p_f について，p_f の全ての隣接プロセスが p_f の状態変化を観測するまでに要するラウンド数の上界を $\mathcal{F}$ と仮定する．定数 $\mathcal{F}$ は故障の見つかりにくさを示す指標と言うことができる．またネットワークの最大次数を d とする．

補題 3.4 任意の実行 $\mathcal{E}$ において，2 個のプロセス $p_i, p_j \notin F$ を考える．p_j が状態変化した直後の状況を c^ℓ とし，c^ℓ は第 $k (k \geq 2)$ ラウンドに属するとする．$T(c^\ell)$ において $(p_i, p_j) \in A(c^\ell)$ ならば，p_i は c^ℓ 以降第 $k + 1$ ラウンド終了までに状態変化する．

証明　実行 $\mathcal{E}$ において，$\ell' < \ell < \ell''$,$p_i \in Q^{\ell'}$,$p_i \in Q^{\ell''-1}$ であるような最小の実行断片 $frag(\mathcal{E};\ell',\ell'') = c^{\ell'}, c^{\ell'+1}, \ldots, c^{\ell''}$ を考える．$k \geq 2$ なので，$p_i \in Q^{\ell'}$ であるような ℓ' は存在する．また，$\sigma^{\ell''-1} = (\Delta, Q^{\ell''-1})$ は第 k または第 $k+1$ ラウンドに含まれることになる．

$c^{\ell'}$ において p_j の親が p_i でない場合と p_i である場合に分けられる．

- $parent_j(c^{\ell'}) = \Lambda_j(p_{j'})$ $(p_{j'} \neq p_i)$ の場合．

  場合分けの仮定より $|N_j| \geq 2$ なので，補題の仮定より，$parent_j(c^{\ell-1}) \neq parent_j(c^{\ell})$ である．$parent_j(c^{\ell'}) = parent_j(c^{\ell''-1}) = \Lambda_j(p_{j'})$ と仮定すると，プロトコルより，実行断片 $frag(\mathcal{E};\ell'+1,\ell''-1)$ において，$parent_j(c^{\ell'''}) = \Lambda_j(p_i)$ である状況 $c^{\ell'''}$ $(\ell' < \ell''' \leq \ell''-1)$ が存在することになる．実行断片 $frag(\mathcal{E};\ell''',\ell''-1)$ において p_i は状態変化しないので，プロトコルより p_j も状態変化しない．つまり，$parent_j(c^{\ell''-1}) = \Lambda_j(p_i)$ となり矛盾する．従って，$parent_j(c^{\ell'}) \neq parent_j(c^{\ell''-1})$ がいえ，p_i は $\sigma^{\ell''-1}$ において，p_j の状態変化を観測可能である．

- $parent_j(c^{\ell'}) = \Lambda_j(p_i)$ の場合．$parent_j(c^{\ell'}) = parent_j(c^{\ell''-1}) = \Lambda_j(p_i)$ と仮定する．p_i は $\sigma^{\ell'}$ で p_j を親としてから，$frag(\mathcal{E};\ell'+1,\ell''-1)$ において動作しないので，$dist_j(c^{\ell'}) + 1 = dist_i(c^{\ell'+1})$ が成立する．ここで，実行断片 $frag(\mathcal{E};\ell',\ell''-1)$ において，p_j がステップ $\sigma^{\ell-1}$ の 1 回のみ動作し，かつ，$\ell = \ell'+1$ の場合は，補題の仮定から $s_j^{\ell-1} \neq s_j^{\ell}$ なので，$dist_j(c^{\ell'}) \neq dist_j(c^{\ell''})$ が成立する．それ以外の場合は p_j は $frag(\mathcal{E};\ell'+1,\ell''-1)$ で動作し p_i を親とするので，$dist_j(c^{\ell''-1}) = dist_i(c^{\ell'+1})+1$ が成立し，従って $dist_j(c^{\ell'})+2 = dist_j(c^{\ell''})$ がいえる．どちらの場合も $dist_j(c^{\ell'}) \neq dist_j(c^{\ell''})$ がいえる．従って，p_i は $\sigma^{\ell''-1}$ において，p_j の状態変化を観測可能である．

いずれの場合も p_i は $k+1$ ラウンド終了までに p_j の状態変化を観測可能であり，p_i は状態変化する．■

定理 3.2 $V-F$ に属するプロセスが状態変化しなくなるまでに要するラウンド数は，高々$n(n/2+\mathcal{F})d$ である．*(d はネットワークの最大次数)*

証明　実行 $\mathcal{E}$ の $V-F$ が状態変化しない接尾部 (定理 3.1により定義可能) において，各状況 c におけるグラフを $T(c)-F=T$ とおく．また，T において根 p_0 からの距離が h であるプロセス集合を V_h と表し，V_h のプロセスが第 $h(n-h/2+\mathcal{F})d$ ラウンド以降は状態変化しないことを，h による帰納法により証明する．初期状況から高々1 ラウンドで根 $r \in V_0$ は状態変化しなくなる．$\bigcup_{\ell=0}^{h-1} V_\ell$ に含まれる各プロセスが状態変化しない接尾部 $frag(\mathcal{E};k_h,\infty)$ において，プロセス $p_i \in V_h$ は高々d 回しか親を変更しない．ここで，p_i が $\ell(1 \leq \ell \leq d)$ 回状態変化した直後の状況を $c^{k_h^\ell}$ とする．(便宜上 $k_h^0 = k_h$ とおく)

状況 $c^{k_h^\ell}$ から，p_i が 1 回状態変化するまでに要するラウンド数は高々$(n-h)+\mathcal{F}$ であるが，これは状況 $c^{k_h^\ell}$ において，p_i から親を辿って得られる経路を，以下の 3 通りの場合に分けることで証明される．

- 経路上に故障プロセス $p_f \in F$ を含む場合．故障プロセス p_f を親とする正常プロセスが p_f の状態変化を観測し，状態変化するまでに $\mathcal{F}$ ラウンド要する．補題 3.4より，経路上の正常プロセス p_j が状態変化してから，それを親とする正常プロセス $p_{j'}$ が状態変化するまで高々1ラウンド要する．経路長は高々$n-h$ なので，p_i が状態変化するまで，高々$(n-h)+\mathcal{F}$ ラウンド要する．

- 経路上に F に属するプロセスを含まず，経路が閉路に接続する場合．高々1ラウンドで閉路に含まれるプロセスのどれかが状態変化する．上の場合と同様の議論により，p_i が状態変化するまで，高々$n-h$ ラウンド要する．

- 経路上に F に属するプロセスを含まず，経路の終点が根 p_0 である場合．経路上のプロセスのうち高々1ラウンドで状態変化するプロセスが存在する場合，上と同様の議論により，p_i が状態変化するまで，高々$n-h$ ラウンド要する．経路上のすべてのプロセスが1ラウンド以内に状態変化しない場合，経路に含まれるプロセス $p_j \in V_{h-1}$ は状況 $c^{k_h^0}$ 以降状態変化しないので，p_j を親とするプロセスは状況 $c^{k_h^\ell}$ 以降状態変化しない．経路上のあるプロセス $p_{j'}$ が状態変化しないならば，$p_{j'}$ を親とするプロセス $p_{j''}$ は状況 $c^{k_h^\ell}$ 以降状態変化しない．従って，経路上のどのプロセスも状況 $c^{k_h^\ell}$ 以降状態変化しないが，T の定義により経路は T に含まれることになる． ■

系 3.1 提案プロトコルは，非停止永久故障の下で生成木を構成する自己安定プロトコルであり，高々$n(n/2+\mathcal{F})d$ ラウンドで安定する． ■

3.3. 本章のまとめ

本章では新たに故障モデルとして非停止永久故障を定義し，その下で生成木を構成する自己安定プロトコルを考察した．新たな故障モデルとして非停止永久故障を定義した．非停止永久故障は，停止を許さないという制限の下で最も性質の悪い故障であるといえる．そして，非停止永久故障の下で生成木構成問題を解く自己安定プロトコルを提案した．これは，自己安定生成木構成問題の可解性にとって，停止故障が致命的な影響を与えることを意味する．

問題を解くにあたって，故障プロセスの状態変化が，すべての正常な隣接プロセスによって無限にしばしば観測されるという仮定を設けている．停止故障と区別するためには，故障プロセスの状態変化を無限にしばしば観測する正常な隣接プロセスが，少なくとも1個は存在するという仮定が必要である．本章の仮定を緩和し，この最低限の仮定の下で自己安定生成木構成問題が解けるかどうか，また，そのままでは解けない場合には，故障プロセス数などについてどのような制限を設ければ問題が解けるかを明確にすることは今後の課題である．

さらに，非停止永久故障の下で，生成木構成問題以外の静的問題（解状況が変化しない問題）が解けるかどうか，相互排除問題などの動的問題（解状況が変化する問題）が解けるかどうかの考察も今後の課題である．

提案したプロトコルの安定時間は高々 $n(n/2+\mathcal{F})d$ ラウンドである．ここで，n はプロセス数，d はネットワークの最大次数，$\mathcal{F}$ は故障プロセスの状態変化が観測されるまでの時間の上界である．永久故障を考慮しない最適な自己安定生成木構成プロトコル [1] の安定時間 $O(diam)$ と比較すると，安定時間を改善できる可能性が考えられる．ここで $diam$ はネットワークの直径である．計算量の下界を求め，安定時間を改善することも今後の課題である．

第 4 章

結論

本論文では，分散システムの一時故障に対して故障耐性を有する自己安定プロトコルについて，効率のよい自己安定プロトコルの設計を行い，さらに高度な故障耐性を有する自己安定プロトコルの設計のため，新たな永久故障モデルの導入を行った．

まずはじめに，木ネットワークにおいてプロセス間の同期を実現する自己安定プロトコルを利用して，ヒープ順序付き木を構成する自己安定プロトコルを取り上げた．このプロトコルは隣接間同期化プロトコルを利用して，安定時間 $O(h)$，領域計算量 $O(K)$ でヒープ順序を実現する． ここで，h は木の高さ，K は入力値のサイズを表す．これは既知の結果と比べ，安定時間，領域計算量を共に改善している．今後の課題としては，分散ヒープ順序づき木に対して，挿入，および，最大値の削除を行う仕組みを実現することが挙げられる．

次に，一時故障だけでなく永久故障に耐性を有する自己安定プロトコルを取り上げた．まず，新たな故障モデルとして非停止永久故障を定義した．非停止永久故障は，停止を許さないという制限の下で最も性質の悪い故障であるといえる．そして，非停止永久故障の下で生成木構成問題を解く自己安定プロトコルを提案した．これは，自己安定生成木構成問題の可解性にとって，停止故障が致命的な影響を与えることを意味する．

問題を解くにあたって，故障プロセスの状態変化が，すべての正常な隣接プロセスによって無限にしばしば観測されるという仮定を設けている．停止故障と区別するためには，故障プロセスの状態変化を無限にしばしば観測する正常な隣接プロセスが，少なくとも 1 個は存在するという仮定が必要である．本論文の仮定を緩和し，この最低限の仮定の下で自己安定生成木構成問題が解けるかどうか，また，そのままでは解けない場合には，故障プロセス数などについてどのような制限を設ければ問題が解けるかを明確にすることは今後の課題である．

さらに，非停止永久故障の下で，生成木構成問題以外の静的問題（解状況が変化しない問題）が解けるかどうか，相互排除問題などの動的問題（解状況が変化する問題）が解けるかどうかの考察も今後の課題である．

提案したプロトコルの安定時間は高々 $n(n/2+\mathcal{F})d$ ラウンドである．ここで，n はプロセス数，d はネットワークの最大次数，$\mathcal{F}$ は故障プロセスの状態変化が

観測されるまでの時間の上界である．永久故障を考慮しない最適な自己安定生成木構成プロトコル [1] の安定時間 $O(diam)$ と比較すると，安定時間を改善できる可能性が考えられる．ここで $diam$ はネットワークの直径である．計算量の下界を求め，安定時間を改善することも今後の課題である．

謝 辞

本研究の全過程を通じて，適切な御指導と御助言を賜わりました藤原秀雄教授に深く感謝の意を表します．

本論文の内容に関し貴重な御助言を戴きました伊藤実教授，関浩之教授に深く感謝致します．

本研究の全過程を通じて，様々な面で懇切丁寧に直接的な御指導，御助言を戴きました大阪大学の増澤利光教授に深く感謝致します．

本研究において，貴重な御助言を戴きました九州大学の福田晃教授に深く感謝致します．

本研究の全過程を通じて，御討論，御支援を戴きました井上美智子助教授に深く感謝致します．

本研究の全過程を通じて，様々な御指導，御支援を戴きました片山喜章助手に深く感謝致します．

本研究を進めるにあたり，様々な御支援を戴きました広島市立大学の井上智生助教授，本学の大竹哲史助手に感謝致します．

本研究を進めるにあたり，熱心な御協力を頂きました長谷川学氏，御討論を頂きました守屋宣氏に感謝します．また，様々な活動を支えていただきました情報論理学講座の皆様に感謝します．

最後に長期間の学生生活を支えてくれた両親に感謝し，謝辞と致します．ありがとうございました．

参考文献

[1] S. Aggarwal and S. Kutten: "Time-optimal self-stabilizing spanning tree algorithms," In *Proceedings of 13th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 15–17, 1993.

[2] L. Alima: "Self-stabilizing max-heap," In *Proceedings of 4th workshop on self-stabilizing systems*, 1999.

[3] E. Anagnostou and V. Hadzilacos: "Tolerating transient and permanent failures," In *WDAG93 Distributed Algorithms 7th International Workshop Proceedings (LNCS725)*, pages 174–188, 1993.

[4] B. Bourgon and A. Datta: "A self-stabilizing distributed heap maintenance protocol," In *Proceedings of 2nd workshop on self-stabilizing systems*, 1995.

[5] NS. Chen, HP. Yu, and ST. Huang: "A self-stabilizing algorithm for constructing spanning trees," *Information Processing Letters*, 39(3):147–151, 1991.

[6] E.W. Dijkstra: "Self stabilizing systems in spite of distributed control," *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

[7] S. Dolev: "Self-stabilization," MIT Press, 2000: ISBN 0-262-04178-2.

[8] S. Dolev, A. Israeli, and S. Moran: "Uniform self-stabilizing leader election," In *Proceedings of the 5th Workshop on Distributed Algorithms*, pages 167–180, 1991.

[9] M.J. Fischer, N.A. Lynch, and M.S. Paterson: "Impossibility of distributed consensus with one faulty process," In *Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 1–7, 1983.

[10] E. Fromentin, M. Raynal, and F. Tronel: "On classes of problems in asynchronous distributed systems with process crashes," In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, pages 470–477, 1999.

[11] R. Gallager, P. Humblet, and P. Spira: "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.

[12] A. Gopal and K. Perry: "Unifying self-stabilization and fault-tolerance," In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC'92)*, pages 195–206, 1993.

[13] C. Johnen, L. Alima, A. Datta, and S. Tixeuil: "Self-stabilizing neighborhood synchronizer in tree network," In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, pages 487–494, 1999.

[14] N. Lynch: "Distributed Algorithms," Morgan Kaufmann, 1996.

[15] T. Masuzawa: "A fault-tolerant and self-stabilizing protocol for the topology problem," In *Proceedings of the 2nd Workshop on Self-Stabilizing Systems*, pages 1.1–1.15, 1995.

[16] H. Matsui, M. Inoue, T. Masuzawa, and H. Fujiwara: "Fault-tolerant and self-stabilizing protocols using an unreliable failure detector," *IEICE Transactions on Information and Systems*, E83-D(10):1831–1840, 2000.

[17] M. Raynal and J. Helary: "Synchronization and control of distributed systems and programs," John Wiley and Sons, Chichester, 1990.

[18] G. Tel: "Introduction to distributed algorithms," Cambridge university press, 1994.