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

浮穴　学慈[†]　長谷川　学[†]　片山　喜章[†]　増澤　利光[†]
藤原　秀雄[†]

A Self-Stabilizing Max-Heap Protocol in Tree Networks

Satoshige UKENA[†], Manabu HASEGAWA[†], Yoshiaki KATAYAMA[†],
Toshimitsu MASUZAWA[†], and Hideo FUJIWARA[†]

あらまし　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，故障耐性のあるプロトコルが重要である．故障耐性を実現する有力な手法の一つに，自己安定プロトコルがある．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．本論文では，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，ヒープ順序付き木を構成する自己安定プロトコルを提案する．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．

キーワード　分散システム，木ネットワーク，自己安定プロトコル，隣接間同期化，ヒープ

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なる分散システムにおいて，プロセスが協調して問題を解くプロトコルの研究が盛んに行われている．特に，一部のプロセスの故障にかかわらず，問題を解くことのできる故障耐性を有するプロトコルが重要視されている．

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であると仮定する．つまり，各プロセスは，あらかじめ決められた初期状態から実行を開始する．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定めない．つまり，自己安定プロトコルは，任意の大域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では，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など）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，自動的に再び解を求めた状況で安定する．したがって自己安定プロトコルは，長期にわたって分散システムの状況を安定に保ち，プロセスの一時的な故障に柔軟に対応することが求められる分散システムの実現に適している．自己安定プロトコルは 1974 年に Dijkstra [3] によって初めて導入された概念であるが，一時故障に対して優れた故障耐性をもつことから，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．

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，多くの応用をもつ．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プロトコル（分散アルゴリズム）に関する研究が，数多く行われている．

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，各プロセスがもつ値（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もつ）ように並べ換えることである．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

[†] 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduated School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0101 Japan

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 [2] 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 [1] 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 [1], [2] のプロトコルと比べ，安定時間，領域計算量をともに改善している．文献 [1], [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期化プロトコル [4] 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，分散システムにおいて，非同期式システムで同期式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 [5]〜[7]．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 [4]．これは，木ネットワークで隣接プロセス間の同期を実現するものである．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．

本論文の構成は，以下のとおりである．**2.** では，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．**3.** では，隣接間同期化プロトコルを示す．**4.** 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提案する．最後に，**5.** で本論文での結論について述べる．

2. 諸 定 義

2.1 分散システム

分散システムは，無向連結グラフ $D = (V, E)$ 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．本論文では，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．したがって P_i, Cld_i から，プロセス i は根，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付けされており，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 及び 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 のもつ変数を更新する．本論文では，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と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，$s_i \in S_i$ $(0 \leqq i \leqq n-1)$ である．

プロトコル $\mathcal{P}$ は，各プロセスの動作を規定する．したがって，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，$Q \subseteqq V$ 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，$c' = \triangle(c, Q)$ 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，$\mathcal{Q} = Q_1, Q_2, \ldots$ と表す．このとき，状況の無限系列 $E = c_0, c_1, c_2, \cdots$ が $c_{i+1} = \triangle(c_i, Q_{i+1})$ $(i \geqq 0)$ 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，E は $Q_1, Q_2, \ldots$ 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．本論文では，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もつとき，いつかはその命令文が実行されることを保証する．

本論文では，非同期式システムを想定しているが，時間計算量の評価は，同期式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 $i (i \geqq 1)$ に対し，$Q_i = V$ となるスケジュール $\mathcal{Q} = Q_1, Q_2, \cdots$ に対する実行 $E = c_0, c_1, \cdots$ 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 E を同期式実行と呼ぶ．

2.2 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 $E = c_0, c_1, \cdots$ において，述語 Y を満たす状況 c_i $(i \geqq 0)$ 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 $E' = c'_0, c'_1, \cdots$ 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 $\mathcal{P}$ とする．また，$\mathcal{SP}$ を実行の集合 $\mathcal{P}$ 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 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．$\mathcal{P}$ の任意の実行 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． □

2.3 隣接間同期化問題

本論文では，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 $E = c_0, c_1, \cdots$ 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 1 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 1 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 E において，あるプロセス p が実行する連続する二つのアクションの間に，p に隣接するすべてのプロセスが正確に 1 度だけアクションを実行することを意味する．

2.4 ヒープ順序付き木構成問題

本論文では，木ネットワークの各プロセスが，初期状況でもつ値（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もつ）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 と書込み専用の出力変数 out_i をもつ．

実行を E とする．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） $\forall i \in V - \{r\}:\ out_i < out_{P_i}$ □

3. 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 $\mathcal{NSP}$ [4] を図 1 に示す．各プロセス i は，論理型変数 col_i をもつ．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

［定理 3.1］[4] 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． □

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

Fig. 1 Self-stabilizing neighborhood synchronizer protocol $\mathcal{NSP}$ for process i.

更に，プロトコル $\mathcal{NSP}$ に対して，次の定理が成り立つ．

［定理 3.2］ プロトコル $\mathcal{NSP}$ の任意の同期式実行を E とする．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． □

4. ヒープ順序付き木を構成する自己安定プロトコル

本章では，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 [2] 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．

4.1 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，それぞれ一つずつデータをもつとする．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，子プロセスのもつ値の最大値が親プロセスのも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

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　プロトコル $\mathcal{HPP}$（プロセス i）
Fig. 2　Protocol $\mathcal{HPP}$ for process i.

ず，各プロセス i はそのための作業用変数 w_i, r_i をもっている．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．ヒープ順序付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．したがって，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 2 項組にすることにより，異なる値とみなせる）．

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

HEAPIFY:
For the root
  $max_i := \max\{w_j \mid j \in Cld_i\}$;
  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 */**
  $max_i := \max\{w_j \mid j \in Cld_i\}$;
  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 3　プロトコル $\mathcal{HPP}$ の手続き **RESET**, **HEAPIFY**（プロセス i）
Fig. 3　Procedure **RESET**, **HEAPIFY** for process i in protocol $\mathcal{HPP}$.

4.2 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，図 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用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 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 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．

［定理 4.1］ プロトコル $\mathcal{HPP}$ は，安定時間 $O(h)$，領域計算量(注 1)$O(K)$ の自己安定ヒープ順序付き木構成プロトコルである． □

定理 4.1 の証明の概略は，付録に掲載する．

5. むすび

本論文では，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提案した．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，領域計算量をともに改善している．

謝辞 日ごろから有益な御討論を頂く奈良先端科学技術大学院大学の井上美智子助手に感謝致します．本研究の一部は，文部省科学研究費補助金（特定領域研究 (B)(2)10205218），及び，中部電力基礎技術研究所研究助成による．

（注 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．

文　献

[1] L. Alima, “Self-Stabilizing max-heap,” Proc. 4th Workshop on Self-stabilizing Systems, pp.94–101, 1999.

[2] B. Bourgon and A. Datta, “A self-stabilizing distributed heap maintenance protocol,” Proc. 2nd Workshop on Self-Stabilizing Systems, pp.5.1–5.13, 1995.

[3] E. Dijkstra, “Self-stabilizing systems in spite of distributed control,” CACM, vol.17, no.11, pp.643–644, 1974.

[4] C. Johnen, L. Alima, A. Datta, and S. Tixeuil, “Self-stabilizing neighborhood synchronizer in tree networks,” Proc. ICDCS, pp.487–494, 1999.

[5] N. Lynch, Distributed Algorithms, Morgan Kaufmann, 1996.

[6] M. Raynal and J. Helary, Synchronization and Control of Distributed Systems and Programs, John Wiley & Sons, 1990.

[7] G. Tel, Introduction to Distributed Algorithms, Cambridge University Press, 1994.

付　録

本付録では，定理 4.1 が成り立つことを示す．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 $E = c_0, c_1, \ldots$ について，状況 c_{i-1}, c_i $(i \geqq 1)$ 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おかないため，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，このような値をもつ変数が存在しないように，理想実行を定義する．

［定義 A.1］（理想実行） 実行 $E = c_0, c_1, \ldots$ が以下の条件を満たすとき，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 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 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 A.2］ 任意の実行を $E = c_0, c_1, \ldots$ 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．

$\mathrm{LastActed}(i, c_j)$: $k \leqq j \ \wedge\ i \in \mathrm{Acted}(c_k)$ 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．

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

［定義 A.3］ 任意の実行 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 $c_0^r, c_1^r, c_2^r, \ldots, c_t^r, \ldots$ を次のように定義する．ただし，便宜上 $c_{-1}^r = c$ とする．

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

上記の系列 $c_0^r, c_1^r, \ldots$ に対し，実行断面 $(\sigma_0(c_t^0), \ldots, \sigma_{n-1}(c_t^{n-1}))$ $(t = 0, 1, \ldots)$ 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．

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

［定義 A.4］ 任意の実行 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 $s_0^i, s_1^i, \ldots$ を E の i への射影といい，$\mathrm{Proj}(E, i)$ と表す．

- s_0^i は E の初期状況における i の状態．
- s_j^i は 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 A.1］ 任意の実行 E について，次を満たす理想実行 $E_{\mathcal{I}}$ が存在する．

各プロセス $i \in V$ について，
$\mathrm{Proj}(E_i', i) = \mathrm{Proj}(E_{\mathcal{I}}, i)$
であるような E の接尾部 E_i' が存在する．

更に，E が同期式実行なら，E_i' の初期状況は E の $4h$ ラウンド以内に現れる．

証明の概略 定理 3.2 より，実行 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，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．更に，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，E が同期実行ならば c は初期状況から $2h$ ラウンド以内に現れる．

E の実行断面 $(\sigma_0(c_t^0), \ldots, \sigma_{n-1}(c_t^{n-1}))$ の中で，根からの距離が h の任意のプロセス i について，c_t^i が c から到達可能であるような実行断面が存在する．E が同期実行ならば $c_t^0 = c_t^1 = \cdots = c_t^{n-1}$ である．

上記の実行断面に含まれる $c_t^0, c_t^1, \ldots, c_t^{n-1}$ について，E の c_t^i から始まる接尾部を E_i' とし，$\mathrm{Proj}(E_i', i) = s_0^i, s_1^i, \ldots$ とすると，実行 $E_{\mathcal{I}} = c_0, c_1, \ldots$ (ただし，$c_\ell = (s_\ell^0, s_\ell^1, \ldots, s_\ell^{n-1})$ $(\ell = 0, 1, 2, \ldots)$) は理想実行であり，題意を満たす． □

補題 A.1 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 A.5］ 任意の理想実行 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，

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

各プロセス i に対し，$\langle v, i\rangle \in Vals(c)$ となる v は正確に一つ定まる．この v を i が状況 c でもつ値と考える（$\langle v, i\rangle \in R(c)$ 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 がもつ値を v と考える）．この値の移動は次のように定義する．

［定義 A.6］（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，$\langle v, i\rangle \in Vals(c)$ とする．

$\langle v, i\rangle$ は以下の $\langle v', j\rangle \in Vals(c')$ 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) $r_{P_i}(c') \neq \perp \wedge w_i(c') = w_{P_i}(c')$（$P_i$ 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) $r_i(c') \neq \perp$（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，$\langle v', j\rangle = \langle in_i, i\rangle$

(e) その他の場合，$\langle v', j\rangle = \langle v, i\rangle$ □

定義 A.6 において，(c) の場合，同じ値をも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 1 度行うとプロセスのも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 A.7］ 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．

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

$|\mathrm{Wrong}(\langle v, i\rangle, c)|$ 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．したがって，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 A.2 から補題 A.7 までは，$\forall i \in V : \mathit{reset}_i = \mathit{false}$ を満たす状況からたかだか $2h$ ラウンドで作業用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，$d_s = d(i_s)$ $(1 \leqq s \leqq t)$ とおき，一般性を失うことなく $d_s > d_{s+1}$ $(1 \leqq s \leqq t-1)$ 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，$\mathrm{Wrong}(\langle v, j\rangle, c') = \{\langle v'_1, i'_1\rangle, \langle v'_2, i'_2\rangle, \ldots, \langle v'_{t'}, i'_{t'}\rangle\}$，$d'_s = d(i'_s)$ $(1 \leqq s \leqq t')$ とおき，$d'_s > d'_{s+1}$ $(1 \leqq s \leqq t'-1)$ 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，更に，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 $\langle v_s, i_s\rangle$ $(1 \leqq s \leqq m)$ 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 A.7 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ることを示す．

［定義 A.8］ 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

$\mathrm{Top}(\langle v, i\rangle, c) = \max(\{m |\ \forall s, 1 < s \leqq m \leqq t : d_{s-1} \geqq d_s + 2\} \cup \{1\})$

［補題 A.2］ 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，$t \geqq t' \wedge d_t \leqq d'_{t'}$ 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，以下の三つの集合を考える．

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

$t \geqq t'$ については $|W^-| \geqq |W^+|$ を示せばよい．ここで，定義 A.6 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 $|W^-| \geqq |W^+|$ が成立．また，$d_t \leqq d'_{t'}$ は明らか． □

補題 A.2 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 A.7 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 A.3］ 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 A.6 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 A.6 より，$v^{j_\ell} \leqq v^\ell$ 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，$\langle v^\ell, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \vee \langle v^{j_\ell}, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \leqq v^\ell \vee v \leqq v^{j_\ell}$ が成立．したがって矛盾が生じる．

- $\exists j'_\ell \in \mathit{Cld}_\ell : \langle v^\ell, j'_\ell\rangle \Rightarrow_{c'} \langle v^\ell, \ell\rangle$ 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 A.6 より，$v^{j_\ell} < v^\ell$ 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，$v^\ell < v$ であり，また m の定義から，$\langle v^{j_\ell}, \ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c) \vee \langle v^\ell, j_\ell\rangle \notin \mathrm{Wrong}(\langle v, i\rangle, c)$ であり，$v \leqq v^\ell \vee v \leqq v^{j_\ell}$ が成立．したがって矛盾が生じる．

– $j'_\ell \neq j_\ell$ 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 A.6 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，$v \leqq v^{j'_\ell} \vee v \leqq v^{j_\ell}$ が成立．したがって矛盾が生じる．

いずれの場合も矛盾が生じる． □

［補題 A.4］ 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．

$$\langle v^\ell, \ell\rangle \in \mathrm{Wrong}(\langle v, i\rangle, c) \implies \ell \notin \mathrm{Acted}(c)$$

（証明の概略） 補題 A.3 と同様にして示される（ただし，補題 A.3 とは m の定義が異なることに注意）． □

［系 A.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 A.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．$m = \mathrm{Top}(\langle v, i\rangle, c)$，$m' = \mathrm{Top}(\langle v, j\rangle, c')$ 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 A.6 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．したがって，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 A.3 より，$\langle v^\ell, \ell\rangle \notin \mathrm{Wrong}(\langle v, j\rangle, c')$ だから，$\forall s, 1 \leqq s \leqq m'' : d'_{s+1} \leqq d'_s - 2$ が成立．ここで，$d'_{m'} \leqq d'_{m''}$ だから，$d'_{m'} < d_m$ が成立する． □

［補題 A.6］ 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 A.3 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．したがって，$d_t \neq d'_{t'}$ が成り立ち，補題 A.2 より，$d_t \leqq d'_{t'}$ だから $d_t < d'_{t'}$ が成り立つ．同時に，補題 A.3 より $m' = t'$ が成り立つ． □

［補題 A.7］ 任意の理想実行を E とする．任意の ℓ $(0 \leqq \ell \leqq h)$, 任意の $\langle v, i\rangle \in \mathit{Vals}(c)$ に対して，初期状況から $2h - \ell$ ラウンド以内に $|\mathrm{Wrong}(\langle v, i\rangle, c)| \leqq \ell$ が成立する．

（証明の概略） 補題 A.2，A.5，系 A.1 より，h ラウンド以内に $|\mathrm{Wrong}(\langle v, i\rangle, c)| = \mathrm{Top}(\langle v, i\rangle, c)$ を満たす状況 c に到達することがいえる．したがって，補題 A.6，系 A.1 より，状況 c から $h - \ell$ ラウンド以内に $|\mathrm{Wrong}(\langle v, i\rangle, c)| \leqq \ell$ を満たすことを帰納的に示すことができる． □

［補題 A.8］ 任意の理想実行 E について，たかだか $8h$ ラウンドで

- $\forall i \in V :\ \mathit{out}_i < \mathit{out}_{P_i}$
- $\{\mathit{in}_i |\ i \in V\} = \{\mathit{out}_i |\ i \in V\}$

が成立し，以降の任意の状況で成立する．

（証明） 補題 A.7 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．ここから更に，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 A.1, A.8 より，定理 4.1 が成り立つ．

（平成 12 年 3 月 27 日受付，7 月 17 日再受付）

浮穴　学慈　（学生員）

平 7 阪大・理・物理卒．平 9 奈良先端大大学院博士前期課程了．現在，同大学院博士後期課程在学中．分散アルゴリズムの研究に従事．

長谷川　学

平 8 神戸大・工・電気電子卒．現在，奈良先端大大学院博士前期課程在学中．分散アルゴリズムの研究に従事．

片山　喜章　（正員）

平 2 阪大・基礎工・情報卒．平 6 同大大学院博士後期課程中退．同年奈良先端科学技術大学院大学情報科学研究科助手．平 7 同大情報科学センター助手．分散プロトコルなどに関する研究に従事．情報処理学会会員．

増澤　利光　（正員）

昭 57 阪大・基礎工・情報卒．昭 62 同大大学院博士後期課程了．同年同大情報処理教育センター助手．同大基礎工助教授を経て，平 6 奈良先端大情報科学研究科助教授，現在に至る．平 5 コーネル大学客員準教授（文部省在外研究員）．分散アルゴリズム，並列アルゴリズム，テスト容易化設計，テスト容易化高位合成の研究に従事．工博．ACM，IEEE，EATCS，情報処理学会各会員．

藤原　秀雄　（正員）

昭 44 阪大・工・電子卒．昭 49 同大大学院博士後期課程了．阪大工学部助手，明治大理工学部教授を経て，現在奈良先端大情報科学研究科教授．昭 56 ウォータールー大客員助教授．昭 59 マッギル大客員準教授．論理設計，高信頼設計，設計自動化，テスト容易化設計，テスト生成，並列処理，計算複雑度に関する研究に従事．著書に “Logic Testing and Design for Testability”（The MIT Press）など．大川出版賞受賞．工博．情報処理学会会員．IEEE Fellow, IEEE Golden Core Member.