経路情報を用いた生成木構成強安定プロトコルに関する研究

長谷川敏之，片山喜章，高橋直久

2004年3月

第1章　はじめに

1.1　研究の背景

本稿では，通信リンクの消滅や出現を考慮した生成木構成強安定プロトコルに関する研究について述べる．

自律的に動作する複数のプロセスとそれらを相互に接続する通信リンクで構成されるネットワークを分散システムという．インターネットも分散システムであるといえる．

分散システムである（複数のプロセスが自律的に動作する）ことによって生じる問題（リーダー選挙問題，排他制御問題等）や分散システムにおいて達成させたい問題（トークン巡回問題，生成木構成問題等）を分散問題という．

その分散システム上で，プロセス同士が通信リンクを介して情報を交換しながら，与えられた分散問題を解くための手順を，分散プロトコルという．

通常の分散プロトコル (分散アルゴリズムともいう) は，プロトコルの実行開始時のシステム状況 (初期 (大域) 状況) を仮定する．たとえば，プロセスの持つ変数の初期値や通信リンクに伝搬中のメッセージが存在しないことなどである．一方，プロトコル開始時の初期 (大域) 状況に一切の仮定をおかず，任意の初期状況からはじめても，問題を解くことができる (解状況 (正当な状況:legitimate state) に達する) 分散プロトコルを自己安定プロトコル [1] と呼ぶ．

自己安定プロトコルは，この性質により，各プロセスとそのプロセスで実行されるプラグラムが無事ならば，プロセスのデータの一部の破壊，リンクを伝播中のメッセージの紛失や改竄などの任意の一時的な故障 (一時故障) によって，正当な状況から任意の状況に変化しても，その状況を初期状況とすると，再び正当な状況に安定する．つまり，一時故障に対して故障耐性をもつ分散プロトコルである．また自己安定プロトコルは，プロトコル実行中に分散システムの形状 (トポロジ) が変化する動的分散システム (dynamic distributed system) でも，十分に長い間トポロジ変化が生じなければ問題を解決できる．

しかし，自己安定プロトコルは，任意の初期状況からはじめても正当な状況に安定するが，正当な状況から分散システムの一部に障害が生じるだけでシステム全体にその影響が及ぶことや修復が明らかに簡単な状況からでも長い時間をかけて再安定することがある．そこで，正当な状況から少数のプロセスの故障によって生じた状況 (小変動状況) から再安定するまでの実行を考慮した自己安定プロトコルが研究されている．小変動状況から再安定までの実行 (再安定実行) において，

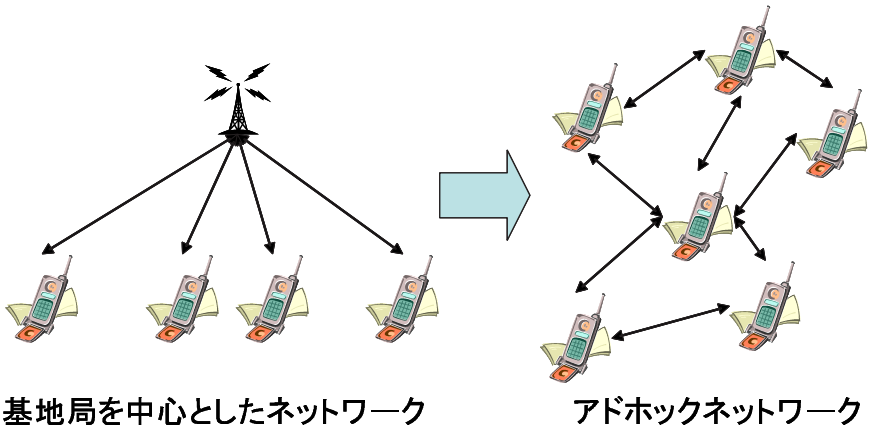

図 1.1: アドホックネットワーク

障害による「影響の伝搬を押さえ」，かつ「速やか」に再安定することを目的としたパラダイムとして，time-adaptive[10] や故障封じ込め (fault-containment) 自己安定プロトコル [7][8] があげられる．一方,「安全に」再安定することを保障する強安定プロトコル (super stabilizing) も研究されている [4],[5].

強安定プロトコルとは，小変動状況からの再安定実行中にある決められた安全条件 (passage 述語) を満たしたまま，再安定するプロトコルである．たとえば，片山ら [6] は，排他制御問題を解く強安定プロトコルを提案している．このプロトコルの passage 述語は,「再安定実行中，故障プロセスを除いてシステム中で特権を持つプロセスは高々1 つ」であり，通常の自己安定プロトコル (再安定実行中，特権がいくつ出現するかわからない) に比べて「安全に」再安定する．また，Dolev ら [4] は，生成木構成問題を解く強安定プロトコルを提案している．ただし，障害の対象としているのは，生成木の枝以外の辺の消滅，出現のみであり，そのときに限り，再安定実行中に生成木の構成が変化しないことを保障している．

ここで実システムに目を移してみる．近年，携帯電話や PDA などモバイル端末の処理・通信速度の向上により，モバイルアドホックネットワークが注目されている．アドホックネットワークは，基地局などの集中管理・制御する機器を介さないで，互いに近隣の端末同士が通信することでネットワークを形成する（図 1.1 参照）．たとえば，地震などの大規模災害によって固定通信回線などの固定インフラが利用できない状況においても，無線端末によるアドホックネットワークによって，迅速な救助活動や安全な避難誘導のための情報収集インフラ構築ができる．

アドホックネットワークは端末や通信リンクの出現，消滅が頻繁に起こる動的なネットワークであることも特徴である．つまり，アドホックネットワークはトポロジが変化する動的分散システムと捉えることができる．自己安定プロトコルは，その性質から，十分に長い時間トポロジ変化がない場合，任意の一時故障 (通信リンクの消滅・出現) にかかわらず正当な状況に安定する．その意味で，アドホック

プロトコル	故障の種類	安全条件
既知の研究 Dolevら	生成木の枝以外の消滅	安全条件①を満たす
	生成木の枝の消滅	×（規定されていない）
提案プロトコル	生成木の枝以外の消滅	安全条件①を満たす
	生成木の枝の消滅	**安全条件②を満たす**

安全条件①＝　「生成木の構成が変更されない」

安全条件②＝　「消滅した生成木を構成するリンクと代替リンクの間に存在するプロセスのみが親子関係を変更させるだけであり、残りのプロセスの親子関係は変化させない」

図 1.2: 既存のプロトコルとの比較

ネットワーク上での自己安定プロトコルは有用であり，その研究は非常に重要である．

そこで，本稿では通信リンクの消滅・削除を考慮した生成木構成強安定プロトコルを提案する．生成木構成とは，分散システム上に生成木 (spanning tree) を構成することであり，生成木を用いて，システム全体への放送や情報収集，モバイルエージェントの巡回など，効率よく行える [2].

強安定プロトコルによる生成木構成問題は，Dolev ら [4] によって提案されている．しかし，生成木に含まれない通信リンクの消滅しか考慮されていない．つまり，もし生成木の枝である通信リンクが消滅した場合，通常の自己安定プロトコルと同様，その影響がシステム全体に影響を及ぼす可能性がある．

本稿では，生成木を構成しない通信リンクが消滅したときは，既存の研究と同様に，生成木は変化しない．そして，生成木を構成する通信リンクが消滅した場合は，十分長い間障害（トポロジ変化）が起きなければ，消滅した生成木を構成するリンクと代替リンクの間に存在するプロセスのみが親子関係を変更させるだけであり，残りのプロセスの親子関係は変化させないで再安定する強安定プロトコルを提案する（図 1.2 参照）.

1.2　本稿の構成

以降では，**??**章でモデルを説明し，2 章で生成木を構成する自己安定プロトコルを紹介し，次に 3 章で強安定プロトコルについて述べ，最後に 4 章でまとめと今後の課題について述べる．　input./model-p.tex

第2章　経路情報を用いた自己安定生成木構成プロトコル

生成木構成自己安定プロトコルとは，任意の一時故障が生じても，十分長い間故障が起きなければ，正当な状況（システム上に生成木が構築されている状況）に到達するプロトコルである．以下では，各プロセスが経路情報を交換しながら，生成木を構成する自己安定プロトコルについて述べる．

2.1　プロトコルのアイデア

生成木を構成する自己安定プロトコルは，すでに Chen ら [3] をはじめいくつかの結果が知られている．しかし，それらはいずれも生成木が構成された状況（正当な状況）で各プロセスは経路情報を持たない．提案プロトコルは各プロセスが根 (プロセス p_r) から自分までの生成木上の経路情報 (根から自分までのプロセスの識別子の系列) を持つ生成木構成自己安定プロトコルである[1]．

まず提案プロトコルの具体的な記述の前に，いくつかの用語の定義を行なう．

2.2　プロトコル

以下では，経路情報による生成木構成自己安定プロトコルの詳細を述べる．

2.2.1　経路情報

プロセス識別子の系列が格納できる変数 $\mathcal{P}$ を経路情報と呼ぶ．

経路情報 $\mathcal{P}$ には，任意の長さの識別子の系列 $\mathcal{P} = \langle p_a, p_b, \cdots \rangle$ か，もしくは空系列であることを意味する記号 $\perp$ が代入される．ネットワーク中に唯一存在する，特別なプロセス p_r の経路情報は常に $\mathcal{P}_r = \langle p_r \rangle$ である．

p_r 以外の各プロセス p_i の経路情報 $\mathcal{P}_i$ は，あるプロセス p_j の経路情報 $\mathcal{P}_j$ の最後にプロセス自身の識別子 p_i を追加することにより生成される．
つまり,$\mathcal{P}_j = \langle p_1, \cdots, p_{m-1}, p_j \rangle$ とすると，$\mathcal{P}_i = \langle p_1, \cdots, p_{m-1}, p_j, p_i \rangle$ と生成される．これを $\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$ と表す．

[1]片山ら [8] も，経路情報を保持した (重み最小) 生成木構成プロトコルのひとつである．

定義 1 (親子関係). 任意の二つのプロセス p_i, p_j が，次の条件を満たすとき，プロセス p_i はプロセス p_j の子プロセス，プロセス p_j はプロセス p_i の親プロセスという.

条件 1 $(p_i, p_j) \in L$

条件 2 $\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$

条件 3 $\mathcal{P}_i$ 中に同じ識別子が二個以上存在しない

これを「親子関係が成立」していると呼び，$p_j \leftarrow p_i$ と表す．親子関係が成立しない場合は $p_j \not\leftarrow p_i$ と表す. □

定義 2 (実経路情報). プロセス p_i の経路情報を $\mathcal{P}_i = \langle p_1, p_2, \cdots, p_{m-1}, p_m \rangle$ とすると, 次の条件を全て満たすとき，$\mathcal{P}_i$ は「実経路情報」という.

条件 1 $p_1 = p_r \wedge p_m = p_i$

条件 2 $m = 1$ ならば，$\mathcal{P}_r = \langle p_r \rangle$．それ以外は $p_1 \leftarrow p_2 \leftarrow \cdots \leftarrow p_{m-1} \leftarrow p_m$

□

自己安定プロトコルでは，初期状態に仮定を置かない．よって，初期状態で各プロセスの所持している経路情報 $\mathcal{P}$ はどのような値をとるかは分からない.

また，定義 2 の実経路情報は大域情報によって認識できるものであり，局所情報しか持たないプロセス自身が認識できるものではない．そこで，局所情報（プロセスがもつ内部変数）のみで判定可能な，経路情報に関する性質である無矛盾経路情報を定義する.

定義 3 (無矛盾経路情報). 各プロセス p_i とその隣接プロセス集合 $\mathcal{N}_i$ において，経路情報が $\mathcal{P}_i = \langle p_1, p_2, \cdots, p_{m-1}, p_m \rangle$ とすると，次の条件を全て満たすとき，$\mathcal{P}_i$ は「無矛盾経路情報」という.

1. $(p_1 = p_r) \wedge (p_m = p_i)$：経路情報 $\mathcal{P}$ の最初の識別子は p_r であり，最後の識別子は p_i である
2. $p_{m-1} \in \mathcal{N}_i$: 経路情報 $\mathcal{P}$ の最後から二番目のプロセス識別子（親プロセス）が隣接プロセスの識別子である（つまり，$(p_{m-1}, p_m) \in L$ である）
3. $\mathcal{P}_i$ に同じ識別子が二個以上存在しない

□

2.2.2 部分木群

経路情報をもつプロセスの集合による任意の状況におけるネットワークは，定義 1 の親子関係が成立しているプロセス集合とそのプロセス間のリンクによって，いくつかの部分木が構成されていると考えることができる．

定義 4 (経路情報による部分木群)．親子関係が成立しているプロセスとリンクで構成される部分ネットワーク集合を部分木 $\mathcal{T}$ とする．部分木 $\mathcal{T}_i \in \mathcal{T}$ の根（以下サブルート），つまり親プロセスを持たないプロセスを $R(\mathcal{T}_i)$ と表し，部分木 $\mathcal{T}_i$ に属するプロセスの集合を $P(\mathcal{T}_i)$ と表す．ただし，p_r をサブルートとする部分木を $\mathcal{T}_0$（つまり $R(\mathcal{T}_0) = p_r$）とする．

また，どの隣接プロセスとも親子関係が成立してないプロセスを単一部分木と呼ぶ．

部分木 $\mathcal{T}_x, \mathcal{T}_y$ について，$p_i \in P(\mathcal{T}_x), p_j \in P(\mathcal{T}_y), (p_i, p_j) \in L$ であるような任意の二つのプロセス p_i, p_j が存在するとき，部分木 $\mathcal{T}_x, \mathcal{T}_y$ は隣接しているという． □

2.2.3 プロトコル中で使用する変数・関数・手続き

各プロセス p_i が，プロトコル中で使用する変数および関数について，以下のように定義する．

- 経路情報 $\mathcal{P}_i = \langle p_1, p_2, \cdots \rangle$:
  経路情報を格納するための変数．ただし，その大きさは有限であると仮定する．

- レジスタ通信で用いるレジスタ構造体 R
  レジスタ構造体 R を次のようなデータ構造に定義する．

  $$R = (ID, \mathcal{P})$$

  ただし，ID にはプロセスの識別子 $p_1, \cdots, p_n$ のうちどれか 1 つ，もしくは空であることを意味する記号 $\perp$ が代入され，$\mathcal{P}$ には経路情報が格納される．レジスタ構造体に含まれる $ID, \mathcal{P}$ は，それぞれ $R.ID, R.\mathcal{P}$ で参照可能である．

- レジスタバッファ $\mathcal{R}_i = \{R_{ji}, R_{ki}, \cdots\}$:
  通信読み込み命令 read() が返したレジスタ構造体の集合を格納する変数

- Random1($\mathcal{R}_x$):
  レジスタバッファ $\mathcal{R}_x$ 内のレジスタ構造体の集合から，ランダムにひとつだけレジスタ構造体を返す関数．

- Parent($\mathcal{P}_x, \mathcal{R}_x$): レジスタバッファ$\mathcal{R}_x$内のレジスタ構造体集合において，経路情報$\mathcal{P}_x = \langle p_1, p_2, \cdots, p_{m-1}, p_m \rangle$における$p_{m-1}$と一致する$ID$を持つレジスタ構造体$R$（つまり$R.ID = p_{m-1}$）を返す関数．もし存在しないのなら$\perp$を返す．また，$\mathcal{P}_i$の長さ$m$が$m \leq 1$の場合も$\perp$を返す．

- Consistent($\mathcal{P}_i$):
  経路情報$\mathcal{P}_i$が無矛盾経路かどうかを判定する関数．

  具体的には，$\mathcal{P}_i$が長さmの経路情報$\mathcal{P}_i = \langle p_1, p_2, \cdots, p_m \rangle$において，プロセス$p_i$とその隣接プロセス集合$\mathcal{N}_i$に関して次の条件を全て満たすとき，真を返し，それ以外は偽を返す．

  1. $p_1 = p_r \wedge p_m = p_i$
  2. $p_{m-1} \in \mathcal{N}_i$
  3. $\mathcal{P}_i$に同じ識別子が二個以上存在しない

2.2.4 経路情報を用いた生成木構成自己安定プロトコル

経路情報を用いた自己安定プロトコルを図2.1に示す．ただし，プロセスが所持する定数や変数以外でのプロトコル中で利用されている変数（R_{tmp}）のライフサイクルは1原子動作中のみである．つまり，1原子動作が終わるたび，$\perp$が代入することで，初期化する．

```
root process(p_r):
do
  P_r := ⟨p_r⟩;
  write(N_r, (p_r, P_r));
od
non-root process(p_i ≠ p_r):
do
    R_i:=read();
    R_tmp := Parent(P_i, R_i);
S1  if R_tmp = ⊥ then {R_tmp := Random1(R_i)};
S2  P_i := ⟨R_tmp.P, p_i⟩;
S3  if ¬Consitent(P_i) then {P_i := ⊥;}
    write(N_i, (p_i, P_i));
od
```

図 2.1: 経路情報を用いた生成木構成自己安定プロトコル

2.3 正当性の証明

提案プロトコルが自己安定プロトコルであることを証明する．
まず経路情報によりプロセスの状態を定義する．

定義 5 (プロセスの状態)**.** 各プロセス p_i が所持する経路情報 $\mathcal{P}_i$ より，プロセス p_i を次のように定義する．

1. 実経路状態:$\mathcal{P}_i$ が実経路情報である
2. 偽経路状態:$\mathcal{P}_i$ が実経路情報ではない

なお，特別なノード p_r の経路情報 $\mathcal{P}_r = \langle p_r \rangle$ は常に実経路状態とする． □

正当な状況を次のように定義する．

定義 6 (正当な状況)**.** 全プロセス p_i が実経路状態である □

以後の提案プロトコルの証明において次の前提を設ける．

前提

全てのプロセスは少なくとも 1 原子動作を実行している（第 2 ラウンド以降である）．

補題 1. 少なくとも *1* 回プロトコルを実行したプロセス p_i の経路情報は無矛盾経路情報か，もしくは $\mathcal{P}_i = \perp$ である．

証明. プロセス p_i がプロトコルを実行すると，プロトコルの S3 において，関数 Consistent が新しく更新した経路情報が無矛盾経路情報であるかどうか判定する．
もし，関数 Consistent が偽を返すなら，$\mathcal{P}_i = \perp$ である．
逆に関数 Consistent が真を返すなら，$\mathcal{P}_i$ は無矛盾経路情報であるので，補題は成り立つ． □

補題 2. 任意の状況において，親子関係の成立したプロセスによる閉路は存在しない

証明. 背理法で証明する．
k（≥ 2）個のプロセス $p_1, \cdots, p_k$ が存在して，親子関係の成立したプロセス間リンクによって形成される閉路 $p_1 \leftarrow p_2 \leftarrow \cdots \leftarrow p_k \leftarrow p_1 \leftarrow \cdots$ が存在すると仮定する．
プロセス p_k の経路情報 $\mathcal{P}_k$ は，$p_{k-1} \leftarrow p_k$ より，$\mathcal{P}_k = \langle \mathcal{P}_{k-1}, p_k \rangle$ である．そして，$\mathcal{P}_k$ に同じ識別子は二個以上存在しないので，$\mathcal{P}_{k-1}$ に p_k は存在してはいけない．そして，プロセス p_{k-1} の経路情報 $\mathcal{P}_{k-1}$ は，$p_{k-2} \leftarrow p_{k-1}$ より，$\mathcal{P}_{k-1} = \langle \mathcal{P}_{k-2}, p_{k-1} \rangle$

である．そして，$\mathcal{P}_{k-1}$ に同じ識別子は二個以上存在しなくて，かつ $\mathcal{P}_{k-1}$ に p_k は存在してはいけないので，$\mathcal{P}_{k-2}$ に p_k, p_{k-1} は存在してはいけない．

これを繰り返していくと，プロセス p_1 の経路情報は，$p_k \leftarrow p_1 \leftarrow p_2 \leftarrow \cdots \leftarrow p_k$ より，$\mathcal{P}_1 = \langle \mathcal{P}_k, p_1 \rangle$ である．そして，$\mathcal{P}_k, \mathcal{P}_1$ に同じ識別子は二個以上存在しなくて，かつ $\mathcal{P}_1$ に $p_2, p_3, \cdots, p_k$ は存在してはいけないので，$\mathcal{P}_k$ に $p_1, \cdots, p_k$ が存在してはいけない．

しかし，プロセス p_k の経路情報 $\mathcal{P}_k$ は，$p_{k-1} \leftarrow p_k$ より，$\mathcal{P}_k = \langle \mathcal{P}_{k-1}, p_k \rangle$ であるので，「$\mathcal{P}_k$ に $p_1, \cdots, p_k$ が存在してはいけない」というのは矛盾しているので，補題が成り立つ．□

補題 3. 部分木 $\mathcal{T}_0$ に属するプロセスの経路情報だけが実経路情報である．

証明. 1. 「部分木 $\mathcal{T}_0$ に属するプロセスの経路情報は全て実経路情報である」を帰納法で証明するため，部分木 $\mathcal{T}_0$ を考える．

$R(\mathcal{T}_0) = p_r$ の経路情報 $\mathcal{P}_r$ は必ず $\mathcal{P}_r = \langle p_r \rangle$ である．経路情報 $\mathcal{P}_r$ の長さは 1 であり，実経路情報である．

ここで，部分木 $\mathcal{T}_0$ に属するプロセスの長さ $m(\geq 1)$ の経路情報は実経路情報であると仮定して，長さ $m+1$ の経路情報を持つプロセス $p_i \in P(\mathcal{T}_0)$ を考えると，プロセス p_i の経路情報 $\mathcal{P}_i$ は $\mathcal{P}_i = \langle p_1, p_2, \cdots, p_m, p_{m+1} \rangle$ である．ここで，$p_i \in P(\mathcal{T}_0)$ より，必ず親プロセス $p_j \in \mathcal{N}_i$ は存在する．$p_j \leftarrow p_i$ より，$\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$ であり，$\mathcal{P}_i$ に同じ識別子が二個以上存在しない．そして，プロセス $p_j \in P(\mathcal{T}_0)$ の経路情報 $\mathcal{P}_j$ は，$\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$ より，$\mathcal{P}_j = \langle p_1, p_2, \cdots, p_m \rangle$ の長さ m の経路情報であり，仮定より実経路情報であるので $p_1 = p_r, p_m = p_j$ である．そして，$\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$ より $p_{m+1} = p_i$ である．よって経路情報 $\mathcal{P}_i$ は，$p_1 = p_r, p_{m+1} = p_i$ かつ $p_1 \leftarrow \cdots \leftarrow p_m \leftarrow p_{m+1}$ より，実経路情報である．

よって，経路情報が長さ $m+1$ でも実経路情報であるので，「部分木 $\mathcal{T}_0$ に属するプロセスの経路情報は全て実経路情報である」は成り立つ．

2. 「部分木 $\mathcal{T}_0$ 以外のサブルートの経路情報は実経路情報ではない」を背理法で証明するため，「部分木 $\mathcal{T}_0$ 以外のサブルートの経路情報は実経路情報である」と仮定する．

ここで，部分木 $\mathcal{T}_k$($\mathcal{T}_0$ を除く) のサブルート $p_i = R(\mathcal{T}_k)$ を考える．そうすると，プロセス p_i は，部分木 $\mathcal{T}_k$($\mathcal{T}_0$ を除く) に属するので $p_i \neq p_r$ である．そしてサブルートなので，プロセス p_i の隣接プロセス $p_j \in \mathcal{N}_i$ において $p_j \not\leftarrow p_i$ である．

- プロセス p_i の経路情報 $\mathcal{P}_i$ の長さが 1 なら，$p_i \neq p_r$ より，定義 2 の条件 1 は満たさない．
- プロセス p_i の経路情報 $\mathcal{P}_i$ の長さが $m(>1)$ なら，$p_j \not\leftarrow p_i$ より，定義 2 の条件 2 を満たさない．

- プロセス p_i の経路情報 $\mathcal{P}_i$ の長さが 0 なら，$\mathcal{P}_i = \perp$ より，定義 2 の条件 1,2 を満たさない．

以上より，プロセス $p_i(= R(\mathcal{T}_k))$ の経路情報 $\mathcal{P}_i$ は実経路情報ではない．

よって，仮定と矛盾するので,「部分木 $\mathcal{T}_0$ 以外のサブルートの経路情報は実経路情報ではない」は成り立つ．

3. 「部分木 $\mathcal{T}_0$ 以外のプロセスの経路情報は実経路情報ではない」を証明する．

部分木 $\mathcal{T}_k(\mathcal{T}_0$ を除く) を考える．

まず部分木 $\mathcal{T}_k$ のサブルートであるプロセス $p_j(= R(\mathcal{T}_k))$ において，その経路情報 $\mathcal{P}_j$ は先ほどの証明より実経路情報ではない．

ここで，部分木 $\mathcal{T}_k$ に属するプロセス p_i の経路情報 $\mathcal{P}_i = \langle p_1, p_2, \cdots, p_i \rangle$ である．よって，プロセス p_i は部分木 $\mathcal{T}_k$ に属しているので，すくなくとも $p_j \leftarrow \cdots \leftarrow p_i$ である．しかし，サブルート p_j は親プロセスを持たない $(p_x \not\leftarrow p_j, p_x \in \mathcal{N}_j)$ ので，実経路情報の定義 2 の条件 2 の $p_1 \leftarrow p_2 \leftarrow \cdots \leftarrow p_m$ は満たしていない．よって,「部分木 $\mathcal{T}_0$ 以外のプロセスの経路情報は実経路情報ではない」は成り立つ．

よって,「部分木 $\mathcal{T}_0$ に属するプロセスの経路情報は実経路情報であり，部分木 $\mathcal{T}_0$ に属さないプロセスの経路情報は実経路情報ではない.」ので，補題は正しい． □

補題 4. 初期状況で全てのプロセスが持っている経路情報に含まれる「ネットワークに存在しないプロセス識別子」の種類の数 α は増加しない．

証明. 初期状況では,「ネットワークに存在しないプロセス識別子」を各プロセスが経路情報に格納している可能性があるので，その総数を $\alpha(\geq 0)$ とする．プロトコル S2 より「ネットワークに存在しないプロセス識別子」は追加できないので，実行中に α が増加しないのは明らかである．

以上より，補題は成り立つ. □

次に，経路情報のレベルを以下のように定義する．

定義 7 (経路情報のレベル)**.** プロセス p_i が持つ経路情報 $\mathcal{P}_i$ 中の長さを $Length(p_i)$ とする．その経路情報 $\mathcal{P}_i$ に存在する「ネットワークに存在しないプロセス識別子」の総数を α_i とする．そのときのレベル $L(p_i)$ は $L(p_i) = Length(p_i) - \alpha_i$ と定義する． □

補題 5. 全てのプロセスの経路情報のレベルは $n+1$ 以上にならない．

証明. まずプロセス p_j の経路情報が $L(\mathcal{P}_j) = n$ であると仮定する．ここで，プロセス $p_i(\in \mathcal{N}_j)$ がプロトコル S1 において，p_j を選んだと仮定する．プロトコル S2 において，$\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$ であり，$\mathcal{P}_i$ のレベルは $n+1$ になるので，経路情報 $\mathcal{P}_i$ に

同じ識別子が二個存在するのは明らかである．よって，$\mathcal{P}_i$ は矛盾経路となるので，プロトコル S3 において必ず $\mathcal{P}_i = \perp$ となる．

以上より，補題が成り立つ． □

補題 6. 部分木 $\mathcal{T}_0$ を構成するプロセスは，その経路情報を変更しない．

証明. ここで，部分木 $\mathcal{T}_0$ を考える．

経路情報の長さが 1 であるプロセス $p_r = R(\mathcal{T}_0)$ は，プロトコルにより常に $\mathcal{P}_r = \langle p_r \rangle$ であり，経路情報 $\mathcal{P}_r$ は変更されない．

長さ $m(\geq 1)$ の経路情報を持つプロセスの経路情報は変更されないと仮定して，長さ $m+1$ の経路情報を持つプロセス $p_i \in P(\mathcal{T}_0)$ を考える．よって，プロセス p_i の経路情報 $\mathcal{P}_i$ は $\mathcal{P}_i = \langle p_1, p_2, \cdots, p_m, p_{m+1} \rangle$ であり，補題 3 より実経路情報である．よって，実経路情報より，$p_1 = p_r, p_{m+1} = p_i$ かつ $p_1 \leftarrow p_2 \leftarrow \cdots \leftarrow p_m \leftarrow p_{m+1}$ なので，プロセス p_i の親プロセス $p_j (= p_m)$ が存在する．そして，プロセス p_j の経路情報 $\mathcal{P}_j$ は $\mathcal{P}_j = \langle p_1, p_2, \cdots, p_m \rangle$ であり，補題 3 より実経路情報であり，$p_1 = p_r, p_m = p_j$ である．また仮定より，長さ m である経路情報 $\mathcal{P}_j$ は変更されない．

よって，プロセス p_i が 1 原子動作を行なうと，

1. まず関数 Parent は，$\mathcal{P}_i = \langle p_1, p_2, \cdots, p_m, p_{m+1} \rangle$，$p_{m+1-1} = p_m = p_j$ かつ $p_m \leftarrow p_{m+1}$ なので $p_j \in \mathcal{N}_i$ より，$R.ID = p_j$ のレジスタ構造体 R を返す．よって，プロトコル S1 において，$R_{tmp}.ID \neq \perp$ であるので，$R_{tmp}.ID = p_j$.

2. プロトコル S2 において，$R_{tmp}.ID = p_j$ より，$\mathcal{P}_i = \langle R_{tmp}.\mathcal{P}, p_i \rangle = \langle \mathcal{P}_j, p_i \rangle$.

3. プロトコル S2 より，プロセス p_i の新しく更新された経路情報 $\mathcal{P}_i$ は，仮定より経路情報 $\mathcal{P}_j$ は変更されないので $\mathcal{P}_i = \langle p_1, p_2, \cdots, p_m, p_i \rangle$ であり，プロセス p_i の 1 原子動作前の経路情報 $\mathcal{P}_i$ と変更はないので実経路情報である．

   よって，$\mathcal{P}_i = \langle p_1, p_2, \cdots, p_m, p_{m+1} \rangle$ であるとすると，実経路情報より $p_1 = p_r, p_{m+1}$，$p_m \leftarrow p_{m+1}$ より $p_m \in \mathcal{N}_i$ かつ経路情報 $\mathcal{P}_i$ 中に同じ識別子は 2 個以上存在しない．以上より，プロトコル S3 において関数 Consistent は真を返すので，$\mathcal{P}_i := \perp$ は実行されない．

よって，プロセス p_i の 1 原子動作前の経路情報 $\mathcal{P}_i$ と 1 原子動作後の経路情報 $\mathcal{P}_i$ との間には変更はない．よって，長さ $m+1$ の経路情報も持つプロセス p_i においても経路情報 $\mathcal{P}_i$ は変更されないので，帰納法において，補題は成り立つ． □

ここで，部分木のレベルを定義する．

定義 8 (部分木群のレベル). 部分木 $\mathcal{T}_i$ のレベル $L(\mathcal{T}_i)$ を，サブルート $R(\mathcal{T}_i)$ の経路情報のレベルとする．つまり，$L(\mathcal{T}_i) = L(R(\mathcal{T}_i))$ である． □

補題 7. 部分木 $\mathcal{T}_k$（$\mathcal{T}_0$ を除く）のサブルート $p_i = R(\mathcal{T}_k)$ は，1 原子動作により経路情報 $\mathcal{P}_i$ は変更されるかもしくは $\perp$ になる．

証明. 補題の条件より，部分木 $\mathcal{T}_k$（$\mathcal{T}_0$ を除く）のサブルート $p_i = R(\mathcal{T}_k) \neq p_r$ が 1 原子動作をする場合を考える．

まず，プロセス p_i はサブルートなので，プロセス p_i の親プロセス存在しない．そして補題 1 より，経路情報 $\mathcal{P}_i$ には同じ識別子が二個以上存在しない．よって，$\forall p_j \in \mathcal{N}_i, \mathcal{P}_i \neq \langle \mathcal{P}_j, p_i \rangle$ である．

そして，プロトコル S1 において隣接プロセスの中のひとつ（以下 p_k とする）が選ばれる．

プロトコル S2 において，$\mathcal{P}_i := \langle \mathcal{P}_k, p_i \rangle$ が実行される．ここで $\forall p_j \in \mathcal{N}_i, \mathcal{P}_i \neq \langle \mathcal{P}_j, p_i \rangle$ より，経路情報が変更されたのは，明らかである．

そして，プロトコル S3 において無矛盾経路情報であるかどうか判定すると，もし関数 Consistent が偽であるならば，$\mathcal{P}_i := \perp$ が実行しないので，経路情報は $\mathcal{P}_i := \langle \mathcal{P}_k, p_i \rangle$ であり，関数 Consistent が偽であるならば，$\mathcal{P}_i := \perp$ が実行するので，経路情報は $\mathcal{P}_i = \perp$ である．

以上より，補題は成り立つ.（証明終了）

□

補題 8. プロセス（p_r を除く）が所持する（空ではない）経路情報は，既存の（空ではない）経路情報によって，生成される

証明. プロセス p_r は，プロトコルより，常に経路情報は $\mathcal{P}_r = \langle p_r \rangle$ である．

$p_i(= p_r)$ は，プロトコルの S1 において，p_i の隣接プロセス $\mathcal{N}_i$ 内からひとつプロセス（以下 p_j とする）を選択する．そして，プロトコルの S2 において，プロセス p_i の経路情報 $\mathcal{P}_i$ は $\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle$ であるので，プロセス $p_i(\neq p_r)$ が所持する経路情報は，既存の経路情報によって，生成される．

そして，プロセス p_j の経路情報 $\mathcal{P}_j$ が $\mathcal{P}_j = \perp$ ならば，プロセス p_i は，プロトコルの S2 において $\mathcal{P}_i = \langle \mathcal{P}_j, p_i \rangle = \langle p_i \rangle$ であるので，プロトコルの S3 において無矛盾経路情報ではない $(p_1 \neq p_r)$ ので，$\mathcal{P}_i := \perp$ が実行される．

以上より，補題が成り立つ． □

補題 9. 単一部分木でない部分木 $\mathcal{T}_i$*(*$\mathcal{T}_0$ は除く*)* において，サブルート $R(\mathcal{T}_i)$ の経路情報が変化するとき，部分木 $\mathcal{T}_i$ は二つ以上の部分木 *(*$\mathcal{T}_{i1}, \mathcal{T}_{i2}, \cdots, \mathcal{T}_{ik}$*)* に分割され，それらのレベルは必ず $L(\mathcal{T}_i) + 1$ である．

証明. 補題の条件より，部分木 $\mathcal{T}_i$($\mathcal{T}_0$ は除く) のサブルート p_i の子プロセスを $p_{i1}, p_{i2}, \cdots, p_{ik}$（単一部分木ではないので $k \geq 1$）である．そして補題 7 より，サブルート p_i が 1 原子動作によって経路情報 $\mathcal{P}_i$ が変更されたとする．ここで各プロセス p_{ik} の経路情報 $\mathcal{P}_{ik} = \langle \mathcal{P}_{i(old)}, p_{ik} \rangle$ である（ただし $\mathcal{P}_{i(old)}$ はサブルートが経路情報を変更される前の経路情報）．ここで，サブルート p_i の経路情報 $\mathcal{P}_i$ が変更されたとき，$\mathcal{P}_i \neq \mathcal{P}_{i(old)}$ より，各プロセス p_{ik} において $\mathcal{P}_{ik} \neq \langle \mathcal{P}_i, p_{ik} \rangle$ であるので，$p_i \not\prec p_{ik}$ になる．よって，p_{ik} は，定義 4 より，親プロセスは存在しないので，プロセス p_{ik} はサブルー

トとなる．つまり， 部分木 $\mathcal{T}_i$ は，プロセス p_i と各プロセス p_{ik} をサブルートとした k 個の部分木 $(\mathcal{T}_{i1}, \mathcal{T}_{i2}, \cdots, \mathcal{T}_{ik})$ に分割される．

このとき，各プロセス p_{ik} の経路情報 $\mathcal{P}_{ik}$ は，サブルート p_i の子プロセスだったので，$\mathcal{P}_{ik} = \langle \mathcal{P}_i, p_{ik} \rangle$ である．よって，部分木 $\mathcal{T}_{i1}, \mathcal{T}_{i2}, \cdots, \mathcal{T}_{ik}$ のレベルは $L(\mathcal{P}_i) + 1 = L(\mathcal{T}_i) + 1$ であるので，補題は成り立つ.（証明終了） □

以上の補題を用いて，提案プロトコルが必ず正当な状況に到達することを証明する．

補題 10. プロトコルはやがて正当な状況になる．

証明. プロトコルの実行により，サブルートが経路情報を変更されていくたびに，部分木 $\mathcal{T}_i$（$\mathcal{T}_0$ を除く）のレベルは増加していく（補題 7，補題 9 より）．さらに増加するレベルは，補題 5 より，$n + 1$ 以上にはならない．

補題 6 と 8 より，部分木 T_0 がサブルート以外のプロセスによって分裂されることはないし，新たに経路情報が生成されることはない．よって，新たに他の部分木（$\mathcal{T}_0$ を除く）より小さいレベルの部分木は生成されない．

よって，部分木のレベルが増加していけば，やがて全てのプロセスは部分木 $\mathcal{T}_0$（レベル $1 = L(p_r)$）（補題 6 より）属するか単一部分木（レベル n もしくは $\mathcal{P} = \perp$）属することが言える．

ここで単一部分木に属するプロセス p_i が，プロトコルの S1 において，

- 単一部分木 $\mathcal{T}_k$($\mathcal{T}_0$ は除く) に属するプロセス (以下 p_j とする）に決定したとき

  プロセス p_j の経路情報がレベル n なら，補題 5 より，無矛盾経路情報ではない．そして，$\mathcal{P}_j = \perp$ であるなら，プロトコルの S2 において $\mathcal{P}_i = \langle p_i \rangle$ になるので，明らかに無矛盾経路情報ではない．よって，単一部分木に属するプロセスは親プロセスとして決定することができない．

- 部分木 $\mathcal{T}_0$ に属するプロセス（以下 p_k とする）に決定したとき

  プロセス p_k の経路情報は，補題 3 より，実経路情報である．プロトコルの S2 において $\mathcal{P}_i = \langle \mathcal{P}_k, p_i \rangle$ になる

  プロトコルの S3 において，無矛盾経路情報かどうかを判定する．ここで経路情報 $\mathcal{P}_i = \langle \mathcal{P}_k, p_i \rangle = \langle p_1, p_2, \cdots, p_m \rangle$ であるとする．まず，$\mathcal{P}_k$ が実経路情報より $p_1 = p_r, p_{m-1} = p_k$ であり，$\langle \mathcal{P}_k, p_i \rangle$ より $p_m = p_i$ である．そして，プロトコルの S1 より，プロセス p_k は必ず隣接プロセスなので，$p_{m-1} = p_k \in \mathcal{N}_i$ である．そして，補題 2 より，閉路が存在しないので，経路情報 $\mathcal{P}_i$ に同じ識別子が二個以上存在しない．以上より，無矛盾経路情報の条件を全て満たすので，単一部分木に属するプロセスは，部分木 $\mathcal{T}_0$ に属することができる．

以上より，やがて単一部分木は部分木 $\mathcal{T}_0$ と親子関係を成立させることができるので，ネットワーク中において部分木 $\mathcal{T}_0$ が唯一存在するのは，補題 3 より，全プロセスは実経路情報をもつことになるので，定義 6 の正当な状況と言える． □

補題 6 と補題 10 より，次の定理が言える．

定理 1. 「経路情報を用いた自己安定生成木構成プロトコル」は生成木構成問題を解く自己安定プロトコルである． □

第3章　生成木構成問題を解く強安定プロトコル

本章では，2 章のプロトコルを拡張した強安定プロトコルについて述べる．

文献 [4] で提案されている強安定生成木の Passage 述語は「正当な状況（生成木が構成されていること）」である．これは，生成木構成強安定プロトコルの Passage 述語としてはこれ以上望めない強い条件である．しかし，強安定であることを達成するために，トポロジ変化イベントとして，生成木の枝およびノードの消滅は強安定の対象としていない．そこで本稿では，このトポロジ変化イベントの条件を緩和し，任意のリンクの消滅を考慮した強安定プロトコルを提案する．

なお本章において，2 章で用いた定義 1,2,3,4 及びプロトコル中で使われた関数 (Parent,Consistent) を改めて再定義せずに用いる．

3.1　プロトコルのアイデア

提案プロトコルは，2 章で述べた「経路情報を用いた生成木構成自己安定プロトコル」を元に，任意の単一リンク故障時にもある安全条件を満たしながら，再安定するように拡張したものである．

プロトコルの動きとしては，直感的には以下の通りである．まず，任意の初期状況からプロトコルを開始すると，2 章のプロトコルに従って，生成木を構成し，安定する．

ここで，トポロジ変化イベントによって生成木を構成するリンクがひとつ消滅した場合を考える．

このとき，生成木は，図 3.1 のように，二つの部分木に分割される．ひとつは，p_r を根とした部分木であり，もう一方は消滅したリンクによって，親プロセスへのリンクを失ったプロセス p_i を根とした部分木である．この状況から，二つの部分木をつなぐリンク（以下代替リンクと呼ぶ）をひとつだけつなげれば，生成木が構成された状況に到達する．

以下，プロトコルのアイデアの実現について，より詳細に述べる．

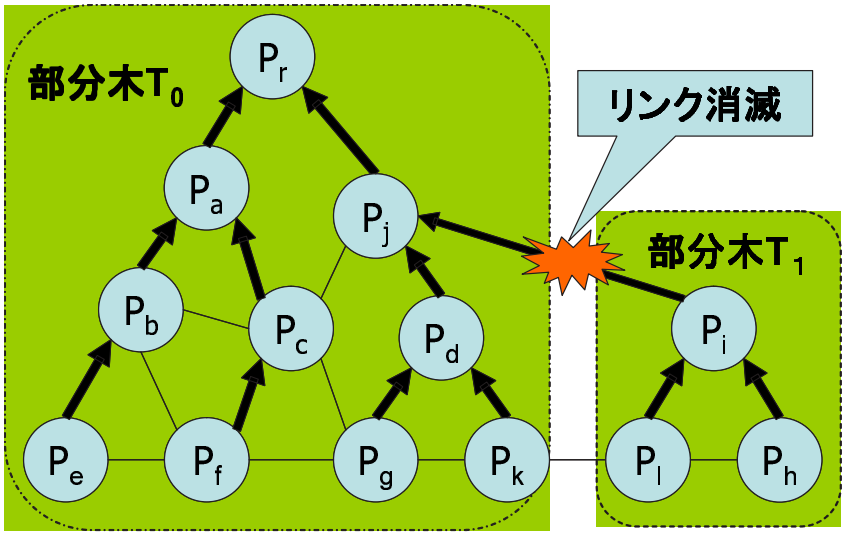

図 3.1: 生成木のリンクの消滅

3.1.1 部分木の区別

分散環境において，各プロセスが「どちらの部分木に属しているか」を区別するのは難しい．しかし，生成木中の各プロセスが，根から自分までの経路情報を知っている場合，比較的容易に自分の属する部分木について，情報を得ることができる．

たとえば図 3.2 のように，正当な状況からリンクの消滅によって，生成木が二つの部分木 $\mathcal{T}_0, \mathcal{T}_1$ に分割された場合を考える．ここで $R(\mathcal{T}_1) = p_i$ とする（もちろん $R(\mathcal{T}_0) = p_r$）と，次のことに気づく．

- 部分木 $\mathcal{T}_0$ に属する全プロセスの経路情報に p_i は含まれておらず，部分木 $\mathcal{T}_1$ に属する全プロセスの経路情報には p_i が含まれている．

部分木 $\mathcal{T}_0$ については，2 章の補題 3 より，部分木 $\mathcal{T}_0$ に属する全プロセスは実経路状態なので，部分木 $\mathcal{T}_1$ に属するプロセス p_i を経路情報に含まない．そして，部分木 $\mathcal{T}_1$ については，部分木 $\mathcal{T}_1$ のサブルートは p_i なので，$\mathcal{T}_1$ に属する全プロセスは経路情報に p_i を含んでいる（図 3.2 参照）．

以上より，リンク先のプロセスがいずれの部分木 $\mathcal{T}_0, \mathcal{T}_1$ に属しているかは，部分木 $\mathcal{T}_1$ のサブルート p_i の識別子が分かっていれば，各プロセスの持つ経路情報から簡単に判断することが可能であることが分かる．つまり，二つの部分木をつないでひとつの生成木を構成するための代替リンクの発見が可能である（図 3.3 参照）．

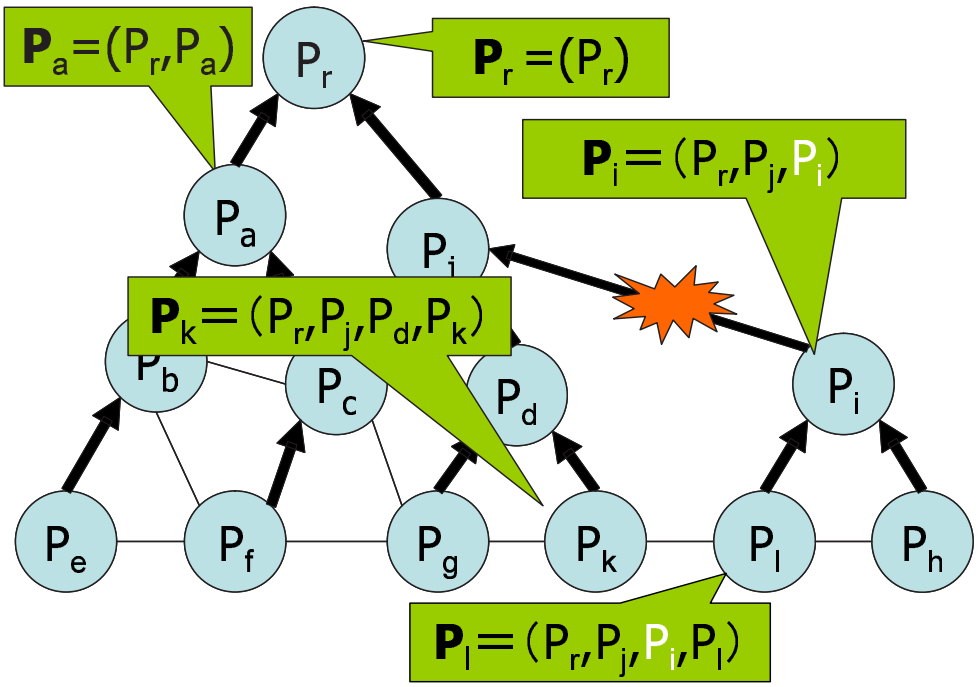

図 3.2: 部分木とその経路情報の関係

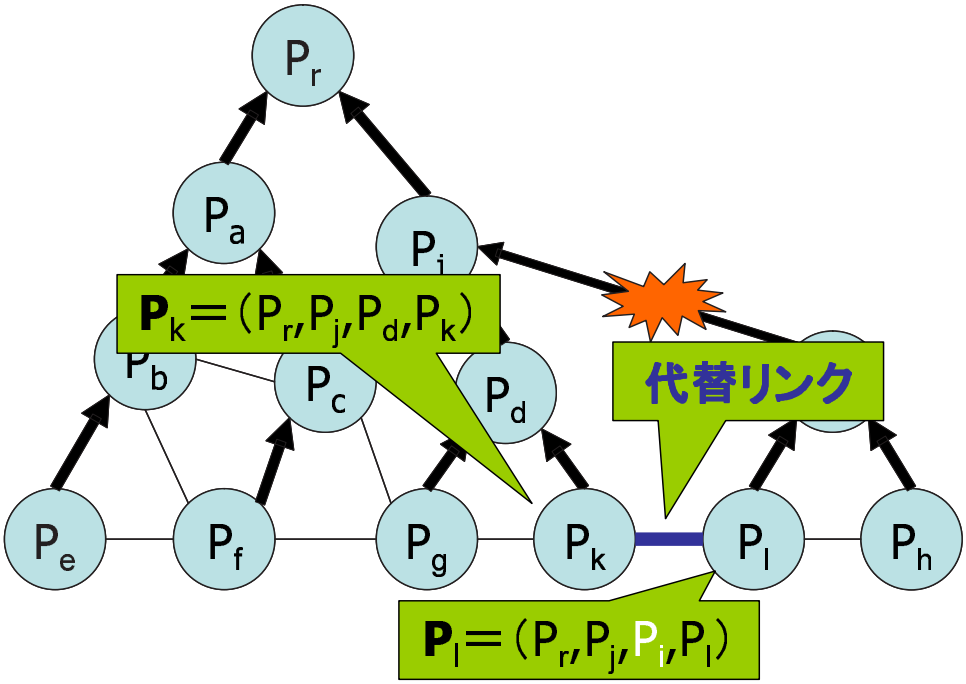

図 3.3: 代替リンクの発見

3.2 提案プロトコル

3.2.1 概略

生成木が構成された状況から，プロセス p_i の親へのリンクが消滅したと考える．

もし，部分木のサブルート p_i が代替リンクを持っている場合は，その代替リンク先のプロセスを新しい親にするだけであり，p_i をサブルートとする部分木 $\mathcal{T}_1$ に属するプロセスは親子関係を変更する必要はない（ただし，部分木 $\mathcal{T}_1$ に属していたプロセスは新しく自分の経路情報を更新する必要がある）．しかし，p_i に代替リンクがない場合は，部分木 $\mathcal{T}_1$ に属する全てのプロセスに代替リンクがあるかどうか尋ねる必要があるが，そのプロセスは部分木のサブルートが p_i だとは知らない．

そこで，部分木 $\mathcal{T}_1$ のサブルート p_i が代替リンク探索用のメッセージを部分木 $\mathcal{T}_i$ 内に放送することで，代替リンクを発見・決定し，生成木の再構成を行なう．

1. p_i が，自分を根とする部分木 $\mathcal{T}_1$ に対して，代替リンクを探索するために，代替リンク探索用変数 Detour を（メッセージに付与して）放送する（図 3.4 参照）．その際には，プロセス p_i が部分木 $\mathcal{T}_1$ のサブルートであることを意味する情報も含まれる．

2. メッセージに付与された代替リンク探索用変数 Detour を受け取り，かつ代替リンクを持つプロセスは，その事実を p_i に向けて代替リンク発見用変数 Discov をメッセージに付与して返信する（図 3.5 参照）．逆に代替リンクを持たないプロセスは，子プロセスに対して代替リンク検索用変数 Detour をメッセージに付与して伝播させる

3. p_i はメッセージに付与された代替リンク発見用変数 Discov を初めて受信すると，そのリンクを部分木の新たな親へのリンクとして選択する旨を，代替リンク決定用変数 Decide をメッセージに付与して部分木に属する特定の（決定した Discov を付与したメッセージを中継した）プロセスに対して放送する（図 3.6 参照）．

4. プロセス p_i が決定した代替リンクをもつプロセス（以下 p_j と呼ぶ）は，メッセージに付与された代替リンク決定用変数 Decide を受け取ったのをきっかけに，代替リンク先のプロセスを親プロセスにして，そのプロセスは親子関係を変更する．

5. p_i から p_j の間の経路中のプロセスが p_j 側から順次親子関係を変更していき，新たな生成木が構成され，再安定する（図 3.7 参照）．その間，p_i を根とする部分木の各プロセスは，プロセス p_i が決定した代替リンクをもつプロセスと p_i との間の経路中のプロセスは，親子関係を反転させ，それ以外の新しい部分木に属するプロセスは親を変更することはない．

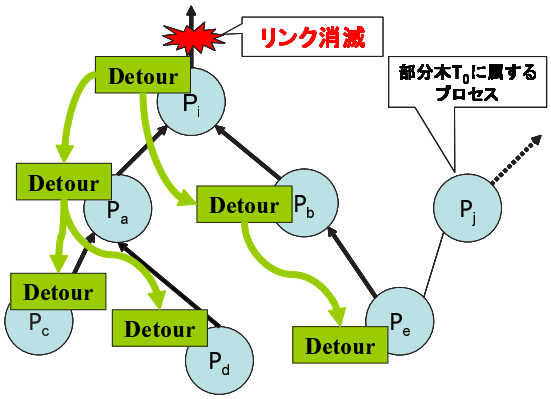

図 3.4: 代替リンク探索 Detour の伝播

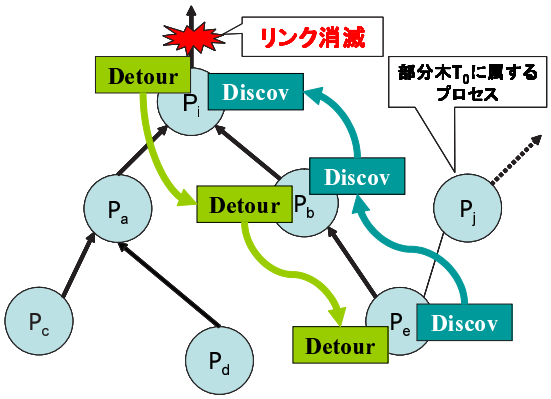

図 3.5: 代替リンク発見 Discov の伝播

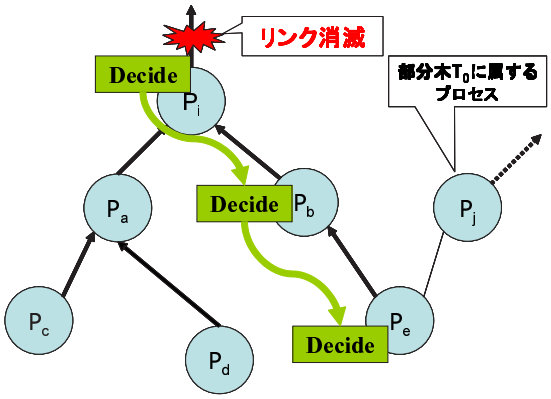

図 3.6: 代替リンク決定 Decide の伝播

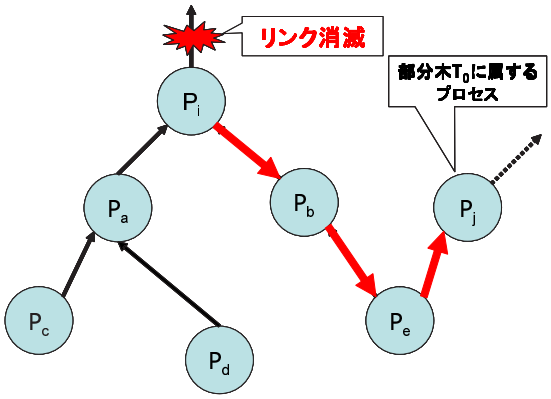

図 3.7: 生成木の再構成

なおこの際，部分木中の全プロセスの経路情報は書き換わることは先に述べたとおりである．

ここで，代替リンクを発見・決定する際にやり取りされる代替リンク経路情報を定義する．

定義 9. 代替リンク経路情報部分木 $\mathcal{T}_i$ が代替リンクを探索していると仮定して，代替リンクを発見したプロセスを $p_j(\in P(\mathcal{T}_i))$ とし，その代替リンク先プロセスを $p_k \notin P(\mathcal{T}_i)$ とすると，p_j が放送する代替リンク経路情報 $\mathcal{P}_D$ は次のようになる．
$\mathcal{P}_D = \langle \mathcal{P}_j, p_k \rangle$ □

3.2.2 プロトコル中で使用する変数・関数

プロトコルにおいて，p_i が所持する変数や実行する関数・手続きを説明する．

- 経路情報 $\mathcal{P}_i = \langle p_1, p_2, \cdots \rangle$:
  経路情報を格納するための変数．ただし，その大きさは有限であると仮定する．

- レジスタバッファ $\mathcal{R}_i = \{R_{ji}, R_{ki}, \cdots\}$:
  通信読み込み命令 read() が返したレジスタ構造体の集合を格納する変数

- 変数 $\mathcal{S}_i$：
  代替リンクの探索，発見や決定に際して，状態の通知に使うための変数であり，次のように表す．

  $\mathcal{S}_i = (\mathcal{SI}_i, \mathcal{SP}_i)|\perp$
  ただし $\mathcal{SI}_i = \text{Detour}|\text{Discov}|\text{Decide}|\perp, \mathcal{SP}_i = \mathcal{P}$

  1. $\mathcal{S}_i = (\text{Detour}, \mathcal{P})$：代替リンク探索状態．$\mathcal{P}$ は代替リンクを探しているプロセスの経路情報．
  2. $\mathcal{S}_i = (\text{Discov}, \mathcal{P})$：代替リンク発見状態．
  3. $\mathcal{S}_i = (\text{Decide}, \mathcal{P})$：代替リンク決定状態．
  4. $\mathcal{S}_i = (\perp, \perp)$：上記以外の状態．ただし，これを $\mathcal{S} = \perp$ と表記することもある．

- レジスタ構造体 R:レジスタ通信で用いられる構造体
  次のように定義する．

  $$R = (ID, \mathcal{P}, \mathcal{S})$$

  ただし，ID にはプロセスの識別子 $p_1, \cdots, p_n$ のうちどれか一つ，もしくは空であることを意味する記号 $\perp$ が代入され，$\mathcal{P}$ には経路情報が格納され，$\mathcal{S}$ には変数 $\mathcal{S}$ が代入される．レジスタ構造体に含まれる $ID, \mathcal{P}, \mathcal{S}$ は，それぞれ $R.ID, R.\mathcal{P}, R.\mathcal{S}$ で参照可能である．

- トポロジ変化検出フラグ t_i:故障（トポロジ変化）検知機能が故障の検知を通知するためのフラグ．故障を検知すると，t_i が真になる．それ以外は偽．

- $P_p, \mathcal{P}_p, \mathcal{S}_p$：$P_p$ にはプロセス p_i の親プロセス $(P_p \leftarrow p_i)$ の識別子が格納される．$\mathcal{P}_p$ には，その親プロセスからのレジスタ構造体 $R(R.ID = P_p)$ 内の $R.\mathcal{P}$ が格納される．$\mathcal{S}_p$ には，その親プロセスからのレジスタ構造体 $R(R.ID = P_p)$ 内の $R.\mathcal{S}$ が格納される．ただし，親プロセスが存在しなければ，すべて $\perp$ が代入される．

- P_c：p_i の子プロセスのプロセス集合が格納される（つまり，$P_c = \{\forall p_x \in \mathcal{N}_i, p_i \leftarrow p_x\}$）．ひとつも存在しない場合は $\perp$ が代入される．

- Parent($\mathcal{P}_x, \mathcal{R}_x$): レジスタバッファ$\mathcal{R}_x$内のレジスタ構造体集合において，経路情報$\mathcal{P}_x = \langle p_1, p_2, \cdots, p_{m-1}, p_m \rangle$における$p_{m-1}$と一致する$ID$を持つレジスタ構造体$R$（つまり$R.ID = p_{m-1}$）を返す関数．もし存在しないのなら$\perp$を返す．また，$\mathcal{P}_i$の長さ$m$が$m \leq 1$の場合も$\perp$を返す．

- Consistent($\mathcal{P}_i$): 経路情報$\mathcal{P}_i$が無矛盾経路かどうかを判定する関数．具体的には，$\mathcal{P}_i$が長さmの経路情報$\mathcal{P}_i = \langle p_1, p_2, \cdots, p_m \rangle$において，プロセス$p_i$とその隣接プロセス集合$\mathcal{N}_i$に関して次の条件を全て満たすとき，真を返し，それ以外は偽を返す．

  1. $p_1 = p_r \wedge p_m = p_i$
  2. $p_{m-1} \in \mathcal{N}_i$
  3. $\mathcal{P}_i$中に同じ識別子が2個以上存在しない

- Random2($\mathcal{R}_i$)：レジスタバッファ$\mathcal{R}_i$内の各レジスタ構造体Rに対して，$R.\mathcal{S} = \perp$であるレジスタ構造体を選別し，選別されたレジスタ構造体の集合から，無作為にひとつのレジスタ構造体を返す関数．もし返せるレジスタ構造体がない場合は$\perp$を返す．

- Last($\mathcal{P}_x$)：長さkの経路情報$\mathcal{P}_x = \langle p_1, p_2, \cdots, p_k \rangle$の一番最後の識別子$p_k$を返す関数．もし，$\mathcal{P}_x = \perp$なら$\perp$を返す．

- Prefix($\mathcal{P}_x, \mathcal{P}_y$)：$\mathcal{P}_y = \langle \mathcal{P}_x, \mathcal{P} \rangle$($\mathcal{P}$は任意の経路情報)であるとき，真を返す関数．それ以外は偽を返す．

- SearchSubLink($\mathcal{P}_x, \mathcal{R}_i$)：レジスタ構造体の集合$\mathcal{R}_i$から，一番最初にPrefix($\mathcal{P}_x, R.\mathcal{P}$)が偽である，レジスタ構造体$R$をひとつだけ返す関数．もし，ひとつも見つからない場合は$\perp$を返す．

- NextPathNode($\mathcal{P}_x, \mathcal{P}_y, \mathcal{R}_i$)：$\mathcal{P}_y = \langle \mathcal{P}_x, p_{y1}, p_{y2}, \cdots, p_{ym} \rangle$であり（Prefix($\mathcal{P}_x, \mathcal{P}_y$)が真），かつ$p_{y1} \in \mathcal{N}_i$であるとき，レジスタ構造体の集合$\mathcal{R}_i$から，レジスタ構造体$R(R.ID = p_{y1})$を返す関数．それ以外の場合は$\perp$を返す．

- $\mathcal{SE}lect(P, \mathcal{R}_i)$：与えられた任意のプロセス集合$P$とレジスタ構造体の集合$\mathcal{R}_i$から，$R.ID \in P$であるレジスタ構造体$R$を選別し，その選別されたレジスタ構造体の集合から，一番優先順位の高いレジスタ構造体Rを返す関数．

  レジスタ構造体Rの変数$R.\mathcal{S}$の四種類の$R.\mathcal{SI}$には，それぞれ以下のような優先順位がつけられる．

  $$\perp < \text{Detour} < \text{Discov} < \text{Decide}$$

  そして，レジスタ構造体Rに($R.\mathcal{SI}$の優先順位，$R.\mathcal{SP}$の長さ（降順），プロセスの識別子$R.ID$の値)の3項組みによる辞書式順序で全順序を与える．

以下に，プロトコル中で使用する述語を定義する．

- $M_1() = (P_p = \perp) \wedge (Last(\mathcal{P}_i) = p_i)$
  $\wedge(\mathcal{SI}_i = \text{Detour}|\text{Dicide})$
- $M_2() = (P_p \neq \perp) \wedge (Last(\mathcal{P}_i) = p_i) \wedge$
  $(\mathcal{SI}_p = \text{Detour}|\text{Dicide}))$
- $M_3() = (\forall R \in \mathcal{R}_i, R.\mathcal{P} \neq \perp)$
- $G_1() = M_3() \wedge (R_x.\mathcal{SI} = \perp) \wedge (\mathcal{SI}_i = \text{Detour})$
- $G_2() = M_3() \wedge (R_x.\mathcal{S} = \text{Discov}) \wedge (\mathcal{SI}_i = \text{Detour})$
- $G_3() = M_3() \wedge (R_x.\mathcal{S} = \text{Discov}|\perp) \wedge (\mathcal{SI}_i = \text{Decide})$
- $G_4() = (R_x.\mathcal{SI} = \perp) \wedge (\mathcal{SI}_p = \text{Detour})$
- $G_5() = (R_x.\mathcal{SI} = \text{Discov}) \wedge (\mathcal{SI}_p = \text{Detour})$
- $G_6() = M_3() \wedge (R_x.\mathcal{SI} = \text{Discov}|\perp) \wedge (\mathcal{SI}_p = \text{Decide})$

以下では，プロトコル中で使用する手続きを定義する．

- 通信書き込みマクロ WRITE(M)
  引数 $M = \{(P_1, \mathcal{S}_1), (P_2, \mathcal{S}_2), \cdots, (P_k, \mathcal{S}_k)\}$ において，全隣接レジスタに対して，次のとおりに実行する．

  $\text{write}(P_1, (p_i, \mathcal{P}_i, \mathcal{S}_1))$;$\text{write}(P_2, (p_i, \mathcal{P}_i, \mathcal{S}_2))$;$\cdots$;
  $\cdots$; $\text{write}(P_k, (p_i, \mathcal{P}_i, \mathcal{S}_k))$;
- Error()
  エラー手続き．$\mathcal{P}_i = \perp, \mathcal{S}_i = \perp, M := \{(\mathcal{N}_i, (p_i, \mathcal{P}_i, \mathcal{S}_i))\}$ が実行される．

Root Process(p_r):ルート p_r の動作
do
$\mathcal{P}_r := \langle p_r \rangle; \mathcal{S}_r := \perp;$
$M := \{(\mathcal{N}_r, \perp)\};$
write($\mathcal{N}_r, (p_r, \mathcal{P}_r, \perp)$);
od
Non-Root Process($p_i \neq p_r$):ルート p_r 以外の動作
do
if t_i then{
if $\mathcal{SI}_i = \perp$ then $\mathcal{S}_i := (\text{Detour}, \mathcal{P}_i)$;
$t_i := false$;
}
$\mathcal{R}_i$:=read();
if $M_1()$ then SuperRootMode
else if $M_2()$ then SuperNodeMode
else NormalMode
WRITE(M);
do

Procedure **NormalMode**
$R_{tmp} :=$ Parent($\mathcal{P}_i, \mathcal{R}_i$);
if $R_{tmp} = \perp$ then $\{R_{tmp} :=$ Random2($\mathcal{R}_i$)$\}$;
$\mathcal{P}_i := \langle R_{tmp}.\mathcal{P}, p_i \rangle$;
if ¬Consitent($\mathcal{P}_i$) then $\{\mathcal{P}_i := \perp;\}$
$\mathcal{S}_i := \perp;$
$M := \{(\mathcal{N}_i, \perp)\};$

Procedure **SuperRootMode**
$R_x := \mathcal{S}Elect(P_c, \mathcal{R}_i);$
if $G_1()$ then {**(G1)**
R_y:=SearchSubLink($\mathcal{P}_i, \mathcal{R}_i$);
if $R_y \neq \perp$ then {
$\mathcal{P}_i := \langle R_y.\mathcal{P}, p_i \rangle; \mathcal{S}_i := \perp; M := \{(\mathcal{N}_i, \perp)\}$;**(A1-1)**
} else $M := \{(P_c, \mathcal{S}_i), ((\mathcal{N}_i - P_c), \perp)\}$;**(A1-2)**
}else if $G_2()$ then {**(G2)**
$\mathcal{S}_i :=$ (Decide, $R_x.\mathcal{SP}$);
$R_y :=$NextPathNode($\mathcal{P}_i, \mathcal{SP}_i, \mathcal{R}_i$);
$M := \{(R_y.ID, \mathcal{S}_i), ((\mathcal{N}_i - R_y.ID), \perp)\}$;**(A2)**
} else if $G_3()$ then {**(G3)**
$R_y :=$NextPathNode($\mathcal{P}_i, \mathcal{SP}_i, \mathcal{R}_i$);
if $R_y.ID = \perp | P_p$ then Error();**(A3-1)**
else if $(R_y.ID \notin P_c)$ then {
$\mathcal{P}_i := \langle R_y.\mathcal{P}, p_i \rangle; \mathcal{S}_i := \perp; M := \{(\mathcal{N}_i, \perp)\}$;**(A3-2)**
}else $M := \{(R_y.ID, \mathcal{S}_i), ((\mathcal{N}_i - R_y.ID), \perp)\}$;**(A3-3)**
} else Error();**(A0)**

Procedure **SuperNodeMode**
$\mathcal{S}_i := \perp; R_x = \mathcal{S}Elect(P_c, \mathcal{R}_i);$
if $G_4()$ then {**(G4)**
R_y:=SearchSubLink($\mathcal{SP}_p, \mathcal{R}_i$);
if $R_y \neq \perp$ then {
$M := \{(P_p, (\text{Discov}, \langle \mathcal{P}_i, R_y.ID \rangle)), ((\mathcal{N}_i - P_p), \perp)\}$;**(A4-1)**
}else $M := \{(P_c, \mathcal{S}_p), ((\mathcal{N}_i - P_c), \perp)\}$;**(A4-2)**
}else if $G_5()$ then {**(G5)**
$M := \{(P_p, R_x.\mathcal{S}), (P_c, \mathcal{S}_p)), ((\mathcal{N}_i - P_p - P_c), \perp)\}$;**(A5)**
}else if $G_6()$ then {**(G6)**
R_y:=NextPathNode($\mathcal{P}_i, \mathcal{SP}_p, \mathcal{R}_i$);
if $(R_y.ID = \perp | P_p) | (R_y.\mathcal{SI} =$Decide) then Error();**(A6-1)**
else if $R_y \notin P_c$ then {
$\mathcal{P}_i := \langle R_y.\mathcal{P}, p_i \rangle; M := (\mathcal{N}_i, \perp)$;**(A6-2)**
}else $M := \{(R_y.ID, \mathcal{S}_x), ((\mathcal{N}_i - R_y.ID), \perp)\}$;**(A6-3)**
}else Error();**(A0)**

図 3.8: 経路情報を用いた生成木構成問題を解く強安定プロトコル

3.2.3 経路情報を用いた生成木構成強安定プロトコル

経路情報を用いた生成木構成問題を解く強安定プロトコルを図3.8に示す．ただし，プロセスが所持する定数や変数以外でのプロトコル中で利用されている変数（R_{tmp}）のライフサイクルは１原子動作中のみである．つまり，１原子動作が終わるたび，$\perp$を代入することで，初期化する．

3.3 強安定プロトコルの証明

以下で，強安定生成木構成プロトコルの正当性を証明するため，まず正当な状況を定義する．

定義 10 (正当な状況). すべてのプロセス p_i が実経路情報 $\mathcal{P}_i$ を持ち，かつ $\mathcal{S}_i = \perp$ であり，そして全レジスタ中の $\mathcal{S} = \perp$ である． □

次に，本稿で対象とするトポロジ変化イベントのクラス Λ を定義する．

定義 11 (トポロジ変化イベントクラス Λ). 単一のリンク消滅をトポロジ変化イベント Λ と呼ぶ. □

正当な状況からクラス Λ に属するトポロジ変化イベント ε が生じたとする．また，リンクの消滅によって生じた二つの部分木を $\mathcal{T}_0(R(\mathcal{T}_0) = p_r), \mathcal{T}_1(R(\mathcal{T}_1) = p_i)$ とする．

このとき，ε が生じてから再安定するまでの実行において，満たされるべき条件（Passage 述語）を以下のように定義する．

定義 12 (Passage 述語). 再安定実行中，親子関係を変更するのは，p_i*(*部分木 $\mathcal{T}_1$ のサブルート*)* と代替リンクをもっているプロセス p_j を結ぶ $\mathcal{T}_1$ 中の経路上にあるプロセスのみであり，それ以外のプロセスは親子関係を変更しない. □

本プロトコルの証明のために，部分木の状態を定義する．

定義 13 (部分木の状態). 部分木 $\mathcal{T}_i$ のサブルートを p_i であるとするとき，

1. $\mathcal{SI}_i = \perp$ かつ $\mathcal{P}_i \neq \perp$ なら，部分木 $\mathcal{T}_i$ は通常状態と呼ぶ.
2. $\mathcal{SI}_i = \mathrm{Detour}$ かつ $\mathcal{P}_i \neq \perp$ なら，部分木 $\mathcal{T}_i$ は探索状態と呼ぶ.
3. $\mathcal{SI}_i = \mathrm{Decide}$ かつ $\mathcal{P}_i \neq \perp$ なら，部分木 $\mathcal{T}_i$ は決定状態と呼ぶ.
4. それ以外の部分木は異常状態と呼ぶ.

□

提案プロトコルが自己安定プロトコルであることを証明するため，次の前提を用意する．

前提

全てのプロセスは少なくとも一回プロトコルを実行している（第 2 ラウンド以降である）.

まず提案プロトコルが自己安定プロトコルであることを証明する．

補題 11. 少なくとも一回プロトコルを実行したプロセス p_i は $Last(\mathcal{P}_i) = p_i$ か $\mathcal{P}_i = \perp$ である．

証明. プロセス p_i がプロトコルを実行すると，

1. プロセス p_i は $Last(\mathcal{P}_i) \neq p_i \wedge \mathcal{P}_i \neq \perp$ であると仮定する．

   ここで，プロセス p_i が一回プロトコルを実行すると，$Last(\mathcal{P}_i) \neq p_i$ より，条件 $M_1(), M_2()$ は偽であり，手続き NormalMode を実行する．

   手続き NormalMode の実行を行なうと，プロセス p_i は，関数 Parent もしくは関数 Random2 より，親プロセスを一つ決めるが，関数 Random2 は $\perp$ を返すときがあるので，

   - プロセス p_i が親プロセスと決めたプロセスが存在する

     プロセス p_i が親プロセスと決めたプロセス (p_j とする）の経路情報 $\mathcal{P}_j$ の末尾に識別子 p_i を追加した経路情報 $\langle \mathcal{P}_j, p_i \rangle$ がプロセス p_i の経路情報となる．その新しい経路情報がもし無矛盾経路情報の条件を満たすなら $\mathrm{Last}(\mathcal{P}_i) = p_i$，満たさない場合は $\mathcal{P}_i = \perp$ である．
   - プロセス p_i が親プロセスと決めたプロセスが存在しない（$\perp$ のとき）

     プロトコルの手続き NormalMode 内の $R.tmp$ は，プロセス p_i が親プロセスと決めたプロセスが存在しないなら，$R_{tmp}.ID = \perp$ である．よって，$R_{tmp}.\mathcal{P} = \perp$ であるので，これに識別子 p_i を追加した経路情報は $\langle R_{tmp}.\mathcal{P}, p_i \rangle = \langle p_i \rangle$ なので，$p_1 \neq p_r$ より無矛盾経路情報の条件を満たさないので $\mathcal{P}_i = \perp$ である．

   親プロセスと決めたプロセスの経路情報の末尾に自分の経路情報を追加するので，$Last(\mathcal{P}_i) = p_i$ か，$\mathcal{P}_i = \perp$（矛盾経路情報と判定されれば）である．

2. プロセス p_i は $\mathcal{P}_i = \perp$ であると仮定する．

   ここで，プロセス p_i が一回プロトコルを実行すると，$\mathcal{P}_i = \perp$ なので，$Last(\mathcal{P}_i) \neq p_i$ より，条件 $M_1(), M_2()$ は偽であり，手続き NormalMode を実行する．そして，手続き NormalMode の実行を行なうと，前と同様に $Last(\mathcal{P}_i) = p_i$ か $\mathcal{P}_i = \perp$ である．

3. プロセス p_i は $Last(\mathcal{P}_i) = p_i$ であると仮定する．

   ここで，プロセス p_i が一回プロトコルを実行すると，$Last(\mathcal{P}_i) = p_i$ より，条件 $M_1(), M_2()$ は真であり，手続き SuperRootMode，SuperNodeMode か NormalMode のいずれかを実行する．

   - 手続き NormalMode の実行ならば，前と同様に $Last(\mathcal{P}_i) = p_i$ か $\mathcal{P}_i = \perp$ である．

- 手続き SuperRootMode の実行ならば,
  (a) プロトコルの A1-1 か A3-2 の実行なら，$Last(\mathcal{P}_i) = p_i$
  (b) プロトコルの A3-1 か A0 の実行なら，$\mathcal{P}_i = \perp$
  (c) それ以外なら，経路情報を変更しないので，$Last(\mathcal{P}_i) = p_i$
- 手続き SuperRootMode の実行ならば,
  (a) プロトコルの A6-2 の実行なら，$Last(\mathcal{P}_i) = p_i$
  (b) プロトコルの A6-1 か A0 の実行なら，$\mathcal{P}_i = \perp$
  (c) それ以外なら，経路情報が変更しないので，$Last(\mathcal{P}_i) = p_i$

以上より，補題は成立する．□

補題 12. 関数 Parent が返すレジスタ構造体 R が $\perp$ であるとき，手続き NormalMode において，$\mathcal{S} \neq \perp$ をレジスタに書き込んでいるプロセスを親プロセスとして親子関係は成立しない．

証明. プロセス p_i が手続き NormalMode を実行して，手続き NormalMode において，関数 Parent が返すレジスタ構造体 R が $\perp$ であるとすると，$R_{tmp} = \perp$ であるので，関数 Random2 が実行される．この関数の機能により，$\mathcal{S} \neq \perp$ をレジスタに書き込んでいるプロセスからのレジスタ構造体は返されない．

また，補題 11 より，各プロセス p_j は $Last(\mathcal{P}_j) = p_j$ か $\mathcal{P}_j = \perp$ である．よって，$\perp$ 以外において，経路情報 $\mathcal{P}_j$ と同じ経路情報を持つプロセスは存在しない．

また，関数 Random2 が返すレジスタ構造体 R の $R.\mathcal{P}$ が $R.\mathcal{P} = \perp$ であるなら，これに識別子 p_i を追加した経路情報は $\langle R.\mathcal{P}, p_i \rangle = \langle p_i \rangle$ なので，$p_1 \neq p_r$ より，無矛盾経路情報の条件を満たさないので $\mathcal{P}_i := \perp$ が実行され，親子関係は成立しない．

以上より，補題は成立する．□

補題 13. 親プロセスからのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を読み込むなら，そのプロセスの 1 原子動作は A4–1, A4–2, A5 のいずれしか実行できない．

証明. ここで，プロセス p_i が，親プロセス p_j からのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を読み込むとする．

そうなると，述語 $M_1()$ の $P_p = \perp$ を満たさないので，手続き SuperRootMode を実行しないが，$p_j \leftarrow p_i$ かつ $\mathcal{SI}_p = \text{Detour}$ より，述語 $M_2()$ を満たすので，手続き SuperNodeMode を実行する．そして，手続き SuperNodeMode において，$\mathcal{SI}_p = \text{Detour}$ より，$G_4()$ と $G_5()$ が真になる可能性を持つので，A4-1,A4-2 もしくは A5 を実行することができる．そして，$G_6()$ が偽であるのは，$\mathcal{SI}_p = \text{Detour}$ より，明らかである．

それ以外の場合を考える．そうなると，それ以外の場合は $\neg G_4() \wedge \neg G_5() \wedge (\mathcal{SI}_p = \text{Detour})$ であるので，$(R_x.\mathcal{SI} = \text{Detour}|\text{Decide}) \wedge (\mathcal{SI}_p = \text{Detour})$．つまり，子プ

ロセス（p_k とする）からのレジスタに対して $R.\mathcal{SI} = \text{Detour|Decide}$ であるレジスタ構造体 R を読み込む場合である．しかし，プロセス p_i の子プロセスからのレジスタに対して $R.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R を読み込むことはありえない．なぜなら，p_k が p_i の子プロセスになるには，p_k が 1 原子動作により，$\mathcal{P}_k = \langle \mathcal{P}_i, p_k \rangle$ に経路情報を変更しないといけない．そうなると，手続き NormalMode やプロトコルの A1-1，A3-2 や A6-2 で経路情報を変更するときは，必ず全隣接プロセスに対して，$\mathcal{SI} = \perp$ を送信しないといけない．よって，p_k は親プロセス p_i に対して $\mathcal{SI} = \text{Detour|Decide}$ を送信することは不可能である．

以上より，プロトコルの A0 が実行されることはないので，補題は成り立つ． □

補題 14. 全プロセス p_i について $\mathcal{S}_i = \perp$ であり，かつトポロジ変化イベントが十分長い時間起きなければ，全レジスタ $\forall R_{xy}(\forall p_x, p_y \in P, (p_x, p_y) \in L)$ に対して，$R_{xy}.\mathcal{S} = \perp$ になる．

証明. 任意の部分木 $\mathcal{T}_k$ を考える．

部分木 $\mathcal{T}_k$ のサブルート $p_i = R(\mathcal{T}_k)$(ただし $p_i \neq p_r$) は，$\mathcal{S}_i = \perp$ かつ p_i は部分木 $\mathcal{T}_k$ の根であるので親プロセスがいない．よって，述語 $M_1()$ の $\mathcal{SI}_i = \text{Detour|Decide}$ を満たさないので，手続き SuperRootMode を実行しない．そして，述語 $M_2()$ の $P_p \neq \perp$ を満たさないので，手続き SuperNodeMode を実行しない．よって，プロセス p_i は手続き NormalMode しか実行できないので，$R.\mathcal{S} = \perp$ であるレジスタ構造体 R を全隣接プロセス（間のレジスタ）に送る．そして，$p_i = p_r (= R(\mathcal{T}_0))$ なら，プロトコルの「ルート p_r の動作」より，$R.\mathcal{S} = \perp$ であるレジスタ構造体 R を全隣接プロセス（間のレジスタ）に送る．

そうすると，プロセス p_i の子プロセスは，親プロセスのレジスタから $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を読み込むので，述語 $M_1()$ の $\mathcal{SI}_i = \text{Detour|Decide}$ を満たさないので，手続き SuperRootMode を実行せず，述語 $M_2()$ の $\mathcal{SI}_p = \text{Detour|Decide}$ を満たさないので，手続き SuperNodeMode を実行しない．よって，手続き NormalMode しか実行できないので，そのプロセスは手続き NormalMode において，$R.\mathcal{S} = \perp$ であるレジスタ構造体を全隣接プロセス（間のレジスタ）に送る．

これを繰り返していくと，部分木 $\mathcal{T}_k$ に属する全プロセスは，手続き NormalMode を実行するしかなくなるので，補題は成り立つ． □

補題 15. 全プロセス p_i に対して $\mathcal{S}_i = \perp$，全レジスタ $\forall R_{xy}(\forall p_x, p_y \in P, (p_x, p_y) \in L)$ に対して $R_{xy}.\mathcal{S} = \perp$ であり，かつトポロジ変化イベントが十分長い時間起きなければ，正当な状況に到達する．

証明. 全プロセス p_i について $\mathcal{S}_i = \perp$ かつ全レジスタ $\forall R_{xy}(\forall p_x, p_y \in P, (p_x, p_y) \in L)$ に対して $R_{xy}.\mathcal{S} = \perp$ であるならば，述語 $M_1()$ の $\mathcal{SI}_i = \text{Detour|Decide}$ を満たさないので，手続き SuperRootMode を実行しない．そして，述語 $M_2()$ の $\mathcal{SI}_p = \text{Detour|Decide}$ を満たさないので，手続き SuperNodeMode を実行しない．よって，全プロセス p_i は $M_1(), M_2()$ の条件より，手続き NormalMode しか実行できない．

さらに補題の仮定により，トポロジ変化イベントが起きないので，t_i は常に偽であるので，プロトコル中の $\mathcal{S}_i := (\mathrm{Detour}, \mathcal{P}_i)$ は実行されない．以上より，常に全プロセス p_i は常に手続き NormalMode を実行し続ける．

ここで，全レジスタ $\forall R_{xy}(\forall p_x, p_y \in P, (p_x, p_y) \in L)$ に対して $R_{xy}.\mathcal{S} = \perp$ であるならば，関数 Random2 の機能は関数 Random1 の機能と変わらなくなる．よって，手続き NormalMode は，2 章で紹介した経路情報を用いた自己安定プロトコルと同一になる．よって 2 章の定理 1 より，いずれ全プロセスは実経路情報を所持する（部分木 $\mathcal{T}_0$ が唯一存在する）状況に到達する．

以上より，これは定義 6 で示した正当な状況である．よって，補題が成り立つ． □

補題 14,15 より，変数 $\mathcal{S}_i \neq \perp$ であるプロセス p_i がやがてなくなることを証明すれば，提案プロトコルが自己安定プロトコルであるといえる．

補題 16. 部分木 $\mathcal{T}_0$ に隣接する探索状態である部分木 $\mathcal{T}_i$ は，やがて探索状態以外に遷移する

証明. 探索状態の部分木 $\mathcal{T}_i$ を考える．

1. 部分木 $\mathcal{T}_0$ に隣接する探索状態である部分木 $\mathcal{T}_i$ は，部分木 $\mathcal{T}_i$ 以外の部分木間に代替リンクを発見することを証明する．

   探索状態の部分木 $\mathcal{T}_i$ のサブルートを p_j とする．

   提案プロトコルは，2 章のプロトコルを基本としているので，経路情報についてを 2 章の定義と同一である．よって，補題 3 より，部分木 $\mathcal{T}_0$ に属する全プロセスの経路情報は実経路であることは変わらない．

   そして，部分木 $\mathcal{T}_0$ に属するプロセスの経路情報には，部分木 $\mathcal{T}_i$ に属するプロセス p_j の識別子は含まれていないことは明らかであり，補題 11 より探索状態のサブルートの経路情報は $\mathrm{Last}(\mathcal{P}_j) = p_j$ であるので，部分木 $\mathcal{T}_i$ に属するプロセスの経路情報には必ず p_j が含まれているのは明らかである．よって，部分木 $\mathcal{T}_i$ に属する全プロセスは，サブルートの識別子 p_j を知ることができれば，あるプロセスが部分木 $\mathcal{T}_i$ 以外に属していかどうかを判別することができる．

   そして部分木 $\mathcal{T}_i, \mathcal{T}_0$ が隣接している場合，部分木 $\mathcal{T}_i$ は部分木 $\mathcal{T}_0$ 間に代替リンクを発見できる．また部分木 $\mathcal{T}_0$ 以外の部分木間に代替リンクを発見する場合もある．

2. 部分木 $\mathcal{T}_0$ に隣接する探索状態である部分木 $\mathcal{T}_i$ は，やがて探索状態以外に遷移することを証明する．背理法で証明するため,「部分木 $\mathcal{T}_0$ に隣接する探索状態である部分木 $\mathcal{T}_i$ は，いつまでも探索状態である」と仮定する

   部分木 $\mathcal{T}_i$ は探索状態なので，サブルート p_j は $\mathcal{SI}_j = \mathrm{Detour}$ かつ $\mathrm{Last}(\mathcal{P}_j) = p_j$ であるので，手続き SuperRootMode を実行する（述語 $M_1()$ が真）．そこで

(a) $G_1()$ が真である場合

i. 代替リンクを発見できる
プロトコルの A1-1 より，$\mathcal{S}_j := \perp$ を実行するので，探索状態ではなくなる．

ii. 代替リンクを発見できない
プロトコルの A1-2 を実行するので，子プロセスへのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を書き込む．

(b) $G_1()$ が偽かつ $G_2()$ が真の場合
プロトコルの A2 を実行するので，$\mathcal{S}_j := (\text{Decide}, \mathcal{P}_D)$ より，決定状態になる．

(c) それ以外の場合
$\mathcal{SI}_j = \text{Detour}$ より，$G_3()$ の $\mathcal{SI}_j = \text{Decide}$ が満たされないので，$G_3()$ が真になることはありえない．よって，それ以外の場合は，$M_3()$ が偽である（隣接プロセスの経路情報が $\perp$ である）とき，もしくは子プロセスからのレジスタに対して $R.\mathcal{SI} = \text{Detour}|\text{Decide}$ であるレジスタ構造体 R を読み込むときである．そのときはプロトコルの A0 を実行するので，異常状態になる．

よって，背理法の仮定に従うなら，サブルート p_j はプロトコル A1-2 が実行可能な状況でなければならない．そしてサブルート p_j が，プロトコル A1-2 を実行することにより，子プロセスへのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を書き込む．

ここで，部分木 $\mathcal{T}_i$ に属するプロセス p_k が，親プロセスからのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を読み込むなら，述語 $M_1()$ の $P_p = \perp$ を満たさないので，手続き SuperRootMode を実行しないが，述語 $M_2()$ を満たすので，手続き SuperNodeMode を実行する．

(a) $G_4()$ が真である場合

i. 代替リンクを発見できる
プロセス p_k は，プロトコルの A4-1 を実行するので，親プロセスへのレジスタに対して $R.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R を書き込む．

ii. 代替リンクを発見できない
プロセス p_k は，プロトコルの A4-2 を実行するので，子プロセスへのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を書き込む

(b) $G_4()$ が偽かつ $G_5()$ が真の場合

プロセス p_k は，プロトコルの A5 を実行するので，親プロセスへのレジスタに対して $R_p.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_p を書き込み，子プロセスへのレジスタに対して $R_c.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R_c を書き込む．

(c) それ以外の場合

補題 13 より，親プロセスからのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を読み込んでいるので，プロセス p_k はプロトコルの A4-1,A4-2,A5 のいずれしか実行できない．

上記の 1 において，部分木 $\mathcal{T}_0$ に隣接する探索状態である部分木 $\mathcal{T}_i$ は，部分木 $\mathcal{T}_i$ 以外の部分木間に代替リンクを発見することを証明したので，サブルート p_j がプロトコル A1-2 を実行し続けるなら（探索状態から遷移しないなら），必ず部分木 $\mathcal{T}_i$ に属するプロセスにプロトコルの A4-1 を実行するプロセスが必ず存在し，$R.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R はサブルート p_j に対して必ず中継されていく．よって，サブルート p_j に $R.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R が中継されると，サブルート p_j はプロトコルの A2 を実行しなければらない．そうなると，部分木 $\mathcal{T}_i$ の探索状態を遷移しなければならない．よって，背理法の仮定は矛盾するので，部分木 $\mathcal{T}_0$ に隣接する探索状態である部分木 $\mathcal{T}_i$ は，やがて探索状態以外に遷移することを証明できる．

上記 2 より，補題は成り立つ． □

補題 17. トポロジ変化イベントが起きなければ，$\mathcal{SI}_i = \text{Decide}$ であるプロセス p_i は，やがて $\mathcal{S}_i = \perp$ になり，以後そのままである．

証明. プロセス $p_i, p_{i+1}, \cdots, p_{i+l-1}, p_{i+l}$ が p_i をサブルートとした部分木に属していて，かつ $p_i \leftarrow p_{i+1} \leftarrow \cdots \leftarrow p_{i+l-1} \leftarrow p_{i+l}$ であるシステム状況を考える．ここで，部分木は決定状態であり，$\mathcal{S}_i = (\text{Decide}, \mathcal{P}_D)$ であるとする．ただし，$\mathcal{P}_D$ は有限長の任意の経路情報である．

背理法で証明するので，「トポロジ変化イベントが起きなければ，決定状態の部分木はいつまでも決定状態である」を仮定する．

部分木 $\mathcal{T}_k$ が決定状態なので，プロセス p_i（条件 $M_1()$ が真）が 1 原子動作を行なうと，まず $\mathcal{SI}_i = \text{Decide}$ なので述語 $G_1(), G_2()$ は偽であり，

- $G_3()$ が偽のとき，手続き Error を実行するので，必ずプロセス p_i の状態を遷移する（$\mathcal{S}_i := \perp$）．
- $G_3()$ が真，かつ関数 NextPathNode が $\perp$ を返すとき，手続き Error（A3-1）を実行するので，必ずプロセス p_i の状態は遷移する（$\mathcal{S}_i := \perp$）．
- $G_3()$ が真，かつ関数 NextPathNode が子プロセスでない識別子を返すとき，A3-2 を実行するので，必ずプロセス p_i の状態は遷移する（$\mathcal{S}_i := \perp$）．

- $G_3()$ が真，かつ関数 NextPathNode が子プロセスである識別子を返すとき，A3-3 を実行するので，プロセス p_i の状態は遷移しない．

である．よって，背理法の仮定に従うためには，プロセス p_i は $\mathcal{S}_i$ =Decide でなければならないので，プロセス p_i はプロトコルの A3-3 しか実行できない．そしてプロセス p_i は，$p_i \leftarrow p_{i+1}$ であるから，p_i が A3-3 を実行することで，子プロセス p_{i+1} へのレジスタに $R.\mathcal{SI}$ = Decide であるレジスタ構造体 R が書き込む．

よって，p_{i+1} は親プロセス p_i からのレジスタに対して，$R.\mathcal{SI}$ = Decide であるレジスタ構造体 R を書き込まれている（条件 $M_2()$ が真）ので，ここで p_{i+1} が 1 原子動作を行なうと，まず $\mathcal{SI}_p$ = Decide なので述語 $G_4(), G_5()$ は偽であり，

- $G_6()$ が偽のとき，手続き Error を実行するので，必ず状態を遷移する ($\mathcal{P}_{i+1} := \perp$ より $p_i \not\leftarrow p_{i+1}$)．

- $G_6()$ が真，かつ NextPathNode($\mathcal{P}_D$)=$\perp | P_p$ のとき，手続き Error(A6-1) を実行するので，必ず状態を遷移する ($p_i \not\leftarrow p_{i+1}$)．

- $G_6()$ が真，かつ NextPathNode($\mathcal{P}_D$)$\notin P_c$ のとき，A6-2 を実行するので，必ず状態が遷移する ($p_i \not\leftarrow p_{i+1}$)．

- $G_6()$ が真，かつ NextPathNode($\mathcal{P}_D$)$\in P_c$ のとき，A6-3 を実行するので，状態は遷移しない．

である．

背理法の仮定に従う（サブルート p_i が A3-3 を実行できる）ためにも，プロセス p_{i+1} は $p_i \leftarrow p_{i+1}$ でなければならないので，プロセス p_{i+1} は A6-3 しか実行できない．

そして $p_{i+1} \leftarrow p_{i+2}$ であるから，p_{i+1} が A6-3 を実行することで，レジスタ $R_{(i+1)(i+2)}$ に対して $R.\mathcal{S}$ =Decide が書き込まれる．

よって，p_{i+2} は親プロセス p_{i+1} からのレジスタ $R_{(i+1)(i+2)}$ に対して Decide が書き込まれている (条件 $M_2()$ が真) ので，p_{i+2} も p_{i+1} 同様に A6-3 しか実行できない．

これを繰り返していけば，プロセス p_{i+k} において，

- プロセス p_{i+k} が持つ経路情報 $\mathcal{P}_{i+k}$ の長さと代替リンク経路情報 $\mathcal{P}_D$ の長さが一致するときがあり，そのときの関数 NextPathNode は必ず $\perp$ を返すときがある．

- トポロジ変化イベントが起きなければ，ネットワーク中のプロセスの数が n 個で固定される．そうなれば，補題 2 より親子関係によって閉路ができないので，子プロセスになれるプロセスは最大 $(n-1)$ 個であるので，必ず NextPathNode($\mathcal{P}_D$)$\notin P_c$ であるときが存在する．

以上より，必ず状態を遷移しなければならないプロセス p_{i+k} が存在する．

そして，プロセス p_{i+k} が A6-3 以外の実行をすれば，プロセス p_{i+k} の状態が遷移するので，親プロセス p_{i+k-1} との親子関係は成立しなくなる．よって，仮定よりプロセス p_{i+k-1} は A6-3 を実行していたので，補題 12 より，プロセス p_{i+k-1} が A6-3 以外を実行しない限り，p_{i+k} が再びプロセス p_{i+k-1} の子プロセスになることはない．よって，p_{i+k-1} は p_{i+k} が子プロセスではないから，A6-3 以外を実行しなければならないので，プロセス p_{i+k-2} との親子関係が成立しなくなる．

これを繰り返していけば，プロセス p_i も A3-3 が実行できない状況がやがて生じる．

以上より，プロセス p_i も A3-3 が実行できない状況がやがて生じるのは，背理法の仮定に矛盾しているので，補題は正しい． □

以上より，次の補題がいえる．

補題 18. 任意の初期状況から，トポロジ変化イベントが十分長い時間起きなければ，やがて全プロセス p_i について $\mathcal{S}_i = \perp$ になる．

証明. 任意の初期状況において，変数 $\mathcal{S}_i \neq \perp$ であるプロセス p_i が，1 原子動作を行なうと，手続き SuperNodeMode や手続き NormalMode においては，その手続き内で $\mathcal{S}_i := \perp$ を実行しているので，プロセス p_i は手続き SuperRootMode を実行していないといけないので，述語 $M_1()$ より，探索状態もしくは決定状態の部分木のサブルート p_i だけが $\mathcal{S}_i \neq \perp$ である．

そしてトポロジ変化イベントが起きなければ，各プロセス p_k のトポロジ変化検出フラグ t_k はいつまでも偽であるので，$\mathcal{S}_k := (\mathrm{Detour}, \mathcal{P}_k)$ は実行されないので，$\mathcal{S}_k = \perp$ から $\mathcal{S}_k = (\mathrm{Detour}, \mathcal{P}_k)$ に遷移するプロセスは存在しない．

また，補題 17 より，初期状況で決定状態である部分木はやがて $\mathcal{SI}_i = (\mathrm{Decide}, \mathcal{P}_i)$ から $\mathcal{S} = \perp$ に遷移する．

そして，部分木 $\mathcal{T}_0$ に隣接している探索状態の部分木は，補題 16 よりやがて探索状態以外の状態に遷移する．もし，探索状態から決定状態に遷移すれば，補題 17 より，その決定状態である部分木はやがてなくなる．

ネットワーク中に部分木 $\mathcal{T}_0$ だけではないなら，部分木 $\mathcal{T}_0$ に隣接する部分木（$\mathcal{T}_i$ と呼ぶ）が存在するのは，ネットワークが二連結以上であることから明らかである．その部分木 $\mathcal{T}_i$ が探索状態もしくは決定状態なら，補題 16 と補題 17 より，通常状態か異常状態のいずれかの状態に遷移する．そうなれば，手続き NormalMode しか実行できなくなった部分木 $\mathcal{T}_i$ のサブルートが，$\mathcal{T}_i$ 以外の部分木 $\mathcal{T}_k$(部分木 $\mathcal{T}_0$ も含む) に属するプロセスを親プロセスとして選ぶなら，部分木 $\mathcal{T}_i$ に属する一部（もしくは全部）のプロセスがその部分木 $\mathcal{T}_k$ に属することになり，部分木 $\mathcal{T}_i$ はなくなる．そうなれば，またネットワーク中に部分木 $\mathcal{T}_0$ だけではないなら，また部分木 $\mathcal{T}_0$ に隣接する部分木が存在するのは，ネットワークが二連結以上であることから明らかである．よって，部分木 T_0 に属するプロセスは実経路情報であり，経路情報を変更しないので，部分木 $\mathcal{T}_0$ に属するプロセスの数は単調増加である．

よって，部分木 $\mathcal{T}_0$ に隣接しておらず，最後まで探索状態から遷移できない部分木 ($\mathcal{T}_l$ と呼ぶ）が存在するとしても，トポロジ変化イベントが十分長い時間起きなければ，ネットワーク中のプロセスの数は一定であるので，単調増加で拡大する部分木 $\mathcal{T}_0$ と部分木 $\mathcal{T}_l$ はやがて隣接する．そうなれば，部分木 $\mathcal{T}_l$ は補題 16 と補題 17 より，通常状態か異常状態のいずれかの状態に遷移する．

以上より，やがて全プロセス p_i について $\mathcal{S}_i = \perp$ になる． □

定理 2. 本プロトコルは，トポロジ変化イベントが十分長い時間起きなければ，生成木構成問題を解く自己安定プロトコルである

証明. トポロジ変化イベントが十分長い時間起きなければ，補題 18, 補題 14 により，補題 15 が証明可能になる．よって，定理 1 より提案プロトコルは自己安定プロトコルである □

トポロジ変化イベントが十分長い時間起きなければ，自己安定性を証明することができたので，続いて強安定性について説明する．

補題 19. 正当な状況 LS からクラス Λ に属するトポロジ変化イベントが生じたあとの状況を Σ_1 とする．$\forall c_i \in \Sigma_1$ において，手続き SuperRootMode を実行可能なプロセスは高々一つである．

証明. トポロジ変化イベントクラス Λ（単一リンクの消滅）に属するトポロジ変化イベントによって，トポロジ変化検出フラグ t_i が真になるプロセス p_i は，高々二つである．つまり，消滅したリンクに接続していたプロセスがトポロジ変化イベントを検出する．

ここで，クラス Λ に属するトポロジ変化イベントがプロセス p_i で生じた，つまり p_i と隣接プロセスの間のリンクが消滅したとする．このとき，消滅したリンクによって，以下の場合に分けられる．

- 生成木を構成するリンク以外が消滅したとき

  プロセス p_i のトポロジ変化検出フラグ t_i が真になり，プロトコルの実行によって $\mathcal{S}_i = (\mathrm{Detour}, \mathcal{P}_i)$ になるが，プロセス p_i の親プロセスとのリンクは切れていない（$P_p \neq \perp$）ので，述語 $M_1()$ は偽である．また，トポロジ変化イベント直前は正当な状況であったので，親プロセスからのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R が書き込まれているので，述語 $M_2()$ は偽である．

  以上より，プロセス p_i は手続き NormalMode を実行する．

- 生成木を構成するリンクが消滅したとき

  ここで生成木を構成するリンクが消滅したとするとき，生成木を構成するリンクは両端のプロセスが親子関係を成立させている．よって，トポロジ変化イベントがプロセス p_i で生じたとき，プロセス p_i はもう片方のプロセスとどのような親子関係を成立させているかで，動作が異なる．

– 子プロセスとのリンクが切れたとき

プロセス p_i のトポロジ変化検出フラグ t_i が真になり，プロトコルの実行によって $\mathcal{S}_i = (\text{Detour}, \mathcal{P}_i)$ になるが，プロセス p_i の親プロセスとのリンクは切れていない（$P_p \neq \perp$）ので，述語 $M_1()$ は偽である．また，トポロジ変化イベント直前は正当な状況であったので，親プロセスからのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R が書き込まれているので，述語 $M_2()$ は偽である．

以上より，プロセス p_i は手続き NormalMode を実行する．

– 親プロセスとのリンクが消滅したとき

プロセス p_i のトポロジ変化検出フラグ t_i が真になり，プロトコルの実行によって $\mathcal{S}_i = (\text{Detour}, \mathcal{P}_i)$ になる．また，正当な状況からトポロジ変化イベントが生じたので，実経路情報の定義より $Last(\mathcal{P}_i) = p_i$．そして，親プロセスとのリンクは切れたので，述語 $M_1()$ は真である．

よって，プロセス p_i は手続き SuperRootMode を実行する．

以上より，クラス Λ に属するトポロジ変化イベントが生じた場合，手続き SuperRootMode を実行するプロセスは，イベントにより消滅したリンクを親プロセスへのリンクとしていたプロセスのみであり，これは高々ひとつである． □

補題 20. $\forall c_i \in \Sigma_1$ において，SuperRootMode を実行可能なプロセスを p_i とする．このとき，$c_{i+1} = c_i(S), p_i \in S$ とすると，c_{i+1} は次のいずれかである．

1. p_i が，子プロセスへのレジスタに対して $R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を書き込んだ状況 Σ_2
2. p_i が，代替リンクを見つけて，経路情報を変更した状況 Σ_5

証明. 補題 19 においてプロセス p_i が 1 原子動作を実行すると，

まず正当な状況からトポロジ変化イベントだけが起きた状況 Σ_1 なので，全隣接プロセスの経路情報が $\perp$ ではないので，述語 $M_3()$ は真である。

そして，トポロジ変化イベントが生じたので，トポロジ変化検出フラグ t_i が真であるので，プロトコルより，$\mathcal{S}_i := (\text{Detour}, \mathcal{P}_i)$ である．以上より，プロトコルの $G_1()$ が真になる（もちろん $M_1()$ も真）ので，プロセス p_i は，SearchSubLink 関数より，代替リンクの発見の可否でプロトコルの A1-1 か A1-2 のどちらかを実行する．

- 全隣接リンクの中から代替リンクを発見しないとき

  プロトコルの A1-2 を実行するので，子プロセスへのレジスタに対して，$R.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R を書き込んだ状況 Σ_2 である．

- 全隣接リンクの中から代替リンクを発見するとき

  プロトコルの A1-1 を実行するので，経路情報を変更した状況 Σ_6 である．

□

補題 21. $\forall c_i^0 \in \Sigma_2$ において，手続き SuperRootMode を実行可能なプロセスを p_i とする．c_i^0 を初期状況とした任意の実行 $E = c_i^0, c_i^1, \cdots, c_i^k, \cdots$ において，プロセス p_i が，初めて子プロセスからのレジスタに対して $R.\mathcal{SI}$ = Deicov であるレジスタ構造体 R を読み込み，かつ Discov をひとつだけ選択した状況 $c_i^k (k > 0)$ が存在する（このような状況の集合を Σ_3 とする）．

証明. まず，補題 20 より，プロセス p_i はプロトコルの A1-2 を実行した．よって，プロセス p_i は，子プロセスへのレジスタに対して $R_c.\mathcal{SI}$ = Detour であるレジスタ構造体 R_c を書き込み，それ以外のプロセスへのレジスタに対して $R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を書き込む．

正当な状況 LS からクラス Λ に属するトポロジ変化イベントが起きたとき，プロセス p_i が手続き SuperRootMode を実行して，その他のプロセスは手続き NormalMode を実行している．プロセス p_i は，補題 20 において，プロトコルの A1-2 を実行している．つまり，正当な状況 LS から状況 Σ_2 の間に，経路情報を変更したプロセスは存在しないので，述語 $M_3()$ は真である．

そして，p_i をサブルートとした部分木に属するプロセス p_j（p_i を除く）が 1 原子動作したとき，

動作 1 プロセス p_j の親プロセスからのレジスタに対して $R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を読み込む場合

プロセス p_j には親プロセスが存在していて（述語 $M_1()$ が偽），かつその親プロセスからのレジスタに対して $R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を読み込む（述語 $M_2()$ が偽）ので，プロセス p_j は手続き NormalMode を必ず実行する。よって、プロセス p_j の全隣接プロセスへのレジスタに対して，$R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を書き込む．

動作 2 プロセス p_j の親プロセスからのレジスタに対して $R.\mathcal{SI}$ = Detour であるレジスタ構造体 R を読み込む場合

プロセス p_j には親プロセスが存在していて（述語 $M_1()$ が偽），かつその親プロセスからのレジスタに対して $R.\mathcal{SI}$ = Detour であるレジスタ構造体 R が読み込む（述語 $M_2()$ が真）．そして，$M_3()$ が真なので，$G_4()$ が真である．

よって，ここで代替リンクの有無によって，プロトコルの A4-1 か A4-2 のどちらかを実行する．

1. SearchSubLink 関数により代替リンクを持っていないことが分かった場合

プロセス p_j はプロトコルの A4-2 を実行するので，子プロセスへのレジスタに対して $R_c.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R_c を書き込み，それ以外のプロセスへのレジスタに対して $R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を書き込む．

2. SearchSubLink 関数により代替リンクを持っていることが分かった場合
プロセス p_j はプロトコルの A4-1 を実行するので，親プロセスへのレジスタに対して $R_p.\mathcal{S} = (\text{Discov}, \mathcal{P}_D)$ であるレジスタ構造体 R_p を書き込み，それ以外のプロセスへのレジスタに対して $R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を書き込む．ただし，$\mathcal{P}_D$ は，代替リンク経路情報であり，定義 9 に従う．

動作 3 親プロセスからのレジスタに対して $R_p.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R_p を読み込んで，かつ子プロセスからのレジスタに対して $R_c.\mathcal{S} = \text{Discov}, \mathcal{P}_D)$ であるレジスタ構造体 R_c を読み込む場合

プロセス p_j はプロトコル A5 を実行するので，子プロセスへのレジスタに対してレジスタ構造体 R_p を書き込み，親プロセスへのレジスタに対してレジスタ構造体 R_c を書き込み，それ以外のプロセスへのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を書き込む．

動作 4 それ以外の動作

親プロセスからのレジスタに対して $R_p.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R_p を読み込んでいない場合は，手続き NormalMode しか実行できない．そして，親プロセスからのレジスタに対して $R_p.\mathcal{SI} = \text{Detour}$ であるレジスタ構造体 R_p を読み込んだ場合，プロセスは補題 13 より，A4-1,A4-2,A5 のいずれかしか実行できない．

よって，動作 4 が実行されることはありえない．

以上より，Detour は動作 2 より，代替リンクを持つ（動作 2 の 2 を実行可能な）プロセスに到達するまで，プロセス p_i をサブルートとした部分木全体へと伝播していく．

そして，ネットワークが二連結以上であることより，部分木に属するプロセスに必ず動作 2 の 2 を実行可能なプロセスは存在する．

よって，動作 3 によって，親プロセスへと $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_c が中継されていくので，必ずサブルートであるプロセス p_i はレジスタ構造体 R_c を読み込む．そして，プロセス p_i が，R_c を読み込むとき，$R_c.\mathcal{SI} = \text{Discov}$ より，プロトコル A2 を実行する．そして，たとえ p_i の複数の子プロセスからのレジスタに対して $R.\mathcal{SI} = \text{Discov}$ のレジスタ構造体 R が書き込まれても，手続き SuperRootMode 内部で最初に実行される $\mathcal{S}Elect()$ 関数より，$R.\mathcal{S}$ の一番優先順位の高いレジスタ構造体 R がひとつだけ選択される．

以上より，補題は成立する． □

補題 22. $\forall c_i^0 \in \Sigma_3$ において，SuperRootMode 実行可能なプロセスを p_i とする．
$\forall c_i^0 \in \Sigma_3$ を初期状況とする任意の実行 $E = c_i^0, c_i^1, \cdots$ において，p_i から Decide を受信・中継するプロセスは選択された $R.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R を中継したプロセスであり，かつそれらのみである．

証明. プロセス $p_i, p_{i+1}, \cdots, p_{i+l-1}, p_{i+l}$ が p_i をサブルートとした部分木に属していて，かつ $p_i \leftarrow p_{i+1} \leftarrow \cdots \leftarrow p_{i+l-1} \leftarrow p_{i+l}$ であるネットワーク状況を考える．ここで，代替リンクを発見可能なプロセスが p_{i+l} であり，その代替リンク先のプロセスを p_d であるとする．そうすれば，代替リンク経路情報 $\mathcal{P}_D$ は $\mathcal{P}_D = \langle \mathcal{P}_{i+l}, p_d \rangle$ である．

正当な状況 LS からクラス Λ に属するトポロジ変化イベントが起きたとき，プロセス p_i が手続き SuperRootMode を実行し，プロセス p_i をサブルートとする部分木に属する（サブルートを除いた）プロセスは手続き SuperNodeMode か手続き NormalMode を実行し，その他のプロセスは手続き NormalMode を実行している．プロセス p_i は，補題 20 においてプロトコルの A1-2 を実行し，補題 21 においてプロトコルの A1-2 と A2 を実行している．また，プロセス p_i をサブルートとする部分木に属する（サブルートを除いた）プロセスは，補題 21 において，p_i 手続き NormalMode かプロトコルの A4-1,A4-2,A5 を実行している．つまり，正当な状況 LS から状況 Σ_3 の間に，経路情報を変更したプロセスは存在しないので，述語 $M_3()$ は真である．

- l に関する帰納法により「p_i から $R_p.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R_p を受信・中継するプロセスは，選択された $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_c を中継したプロセスである」が成り立つことを証明する．

  $l = 0$ のときは，代替リンクをもつプロセスとプロセス p_i が一致するなら，補題 20 において，A1-1 を実行しているはずなので，ありえない．

  $l = 1$ のとき，代替リンク経路情報 $\mathcal{P}_D$ は $\mathcal{P}_D = \langle \mathcal{P}_{i+1}, p_d \rangle$ である．そして $p_i \leftarrow p_{i+1}$ より，$\mathcal{P}_{i+1} = \langle \mathcal{P}_i, p_{i+1} \rangle$ である．ここで，プロセス p_i は，$\mathcal{S}_i = (\text{Decide}, \mathcal{P}_D)$ であるので，$G_1(), G_2()$ は偽であり，$G_3()$ が真である．そこでプロトコルより，プロセス p_i が実行する関数 NextPathNode$(\mathcal{P}_i, \mathcal{P}_D, \mathcal{R}_i)$ が返す値は必ず p_{i+1} であり，かつ $p_i \leftarrow p_{i+1}$ より，プロトコル A3-3 を実行する．プロセス p_i がプロトコルの A3-3 を実行すると，プロセス p_{i+1} へのレジスタに対してレジスタ構造体 R_p を書き込み，それ以外のプロセスへのレジスタに対して $R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を書き込む．よって，$l = 1$ において「p_i から $R_p.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R_p を受信・中継するプロセスは，選択された $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_c を中継したプロセスである」は成り立つ．

  ここで，$l = m$ のとき，「p_i から $R_p.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R_p を受信・中継するプロセスは，選択された $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構

造体 R_c を中継したプロセスである」が成り立つと仮定し，$l = m+1$ のときを考える．よって，代替リンク経路情報 $\mathcal{P}_D$ は $\mathcal{P}_D = \langle \mathcal{P}_{i+m+1}, p_d \rangle$ である．そして，$p_i \leftarrow \cdots \leftarrow p_{i+m+1}$ より，$\mathcal{P}_{i+m+1} = \langle \mathcal{P}_i, p_{i+1} \cdots, p_{i+m}, p_{i+m+1} \rangle$ である．ここで $l = m$ のとき補題が成り立つので，プロセス p_m までレジスタ構造体 R_p が中継されてきたとする．つまり，プロセス p_{i+m} は，親プロセス p_{i+m-1} からのレジスタに対してレジスタ構造体 R_p を読み込むので，$G_4(), G_5()$ は偽であり，$G_6()$ が真である．そこでプロトコルより，プロセス p_{i+m} が実行する関数 NextPathNode$(\mathcal{P}_{i+m}, \mathcal{P}_D, \mathcal{R}_{i+m})$ が返す値は必ず p_{i+m+1} であり，かつ $p_{i+m} \leftarrow p_{i+m+1}$ より，プロトコル A6-3 を実行する．よって，プロセス p_{i+m} がプロトコルの A6-3 を実行すると，プロセス p_{i+m+1} へのレジスタに対してレジスタ構造体 R_p を書き込み，それ以外のプロセスへのレジスタに対して $R.\mathcal{SI} = \perp$ であるレジスタ構造体 R を書き込む．よって，帰納法により「p_i から $R_p.\mathcal{SI} =$ Decide であるレジスタ構造体 R_p を受信・中継するプロセスは，選択された $R_c.\mathcal{SI} =$ Discov であるレジスタ構造体 R_c を中継したプロセスである」が成り立つ．

- 「選択された $R_c.\mathcal{SI} =$ Discov であるレジスタ構造体 R_c を中継したプロセス以外は $R_p.\mathcal{SI} =$ Decide であるレジスタ構造体 R_p を受信・中継しない」ことを証明する．

  サブルート p_i が $R_c.\mathcal{SI} =$ Discov であるレジスタ構造体 R_c を選択したとき，p_i の（レジスタ構造体 R_c を中継していない）子プロセス p_k が，p_i へのレジスタに対して $R.\mathcal{SI} =$ Discov であるレジスタ構造体 R を書き込んだとする．

  ここでサブルート p_i が 1 原子動作すると，サブルート p_i が $R_c.\mathcal{SI} =$ Discov であるレジスタ構造体 R_c を選択したので，すでに $\mathcal{S}_i = (Decide, \mathcal{P}_D)$ であるので，$G_1(), G_2()$ は偽であり，$G_3()$ が真である．よってサブルート p_i はプロトコルの A3-1,A3-2 か A3-3 のいずれかを実行するので，

  1. サブルート p_i がプロトコルの A3-1 を実行する
     手続き Error を実行するので，$\mathcal{S}_i = \perp$ であり，全隣接プロセス（プロセス p_k）へのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を書き込む．
  2. サブルート p_i がプロトコルの A3-2 を実行する
     親子関係を変更して，かつ全隣接プロセス（プロセス p_k）へのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を書き込む．
  3. サブルート p_i がプロトコルの A3-3 を実行する
     サブルート p_i が実行した関数 NextPathNode が返す値は，レジスタ構造体 R_c を中継した，サブルート p_i の子プロセスである，ひとつだけの識別子である．よって，サブルート p_i はそのひとつだけの識別子をも

つプロセスへのレジスタに対して R_p を書き込み，それ以外のプロセス（プロセス p_k）へのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を書き込む.

以上より，サブルート p_i がどのような動作を行なっても，レジスタ構造体 R_c を中継していないプロセス p_k へのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R が書き込まれているだけであるので,「選択された $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_c を中継したプロセス以外は $R_p.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R_p を受信・中継しない」は成り立つ.

「p_i から $R_p.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R_p を受信・中継するプロセスは，選択された $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_c を中継したプロセスである」が成り立ち,「選択された $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_c を中継したプロセス以外は $R_p.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R_p を受信・中継しない」が成り立つので，補題は成り立つ. □

補題 23. $\forall c_i^0 \in \Sigma_3$ において，SuperRootMode を実行可能なプロセスを p_i とし，p_i がひとつだけ選択した $R_c.\mathcal{SI} = \text{Discov}$ であるレジスタ構造体 R_c を初めて送信したプロセスを p_j とする.

c_i^0 を初期状況とした任意の実行 $\dot{E} = c_i^0, c_i^1, \cdots, c_i^k, \cdots$ において，初めて経路情報を変更する状況 $c_i^k (k > 0)$ が存在し，かつ p_j は代替リンクを持つ（このような状況の集合を Σ_4 とする）.

証明. プロセス $p_i, p_{i+1}, \cdots, p_{i+l-1}, p_{i+l}, p_j$ が p_i をサブルートとした部分木に属していて，かつ $p_i \leftarrow p_{i+1} \leftarrow \cdots \leftarrow p_{i+l-1} \leftarrow p_{i+l} \leftarrow p_j$ であるネットワーク状況を考える．ここで，代替リンクを発見可能なプロセスが p_j であり，その代替リンク先のプロセスを p_d であるとする．このとき，代替リンク経路情報 $\mathcal{P}_D$ は $\mathcal{P}_D = \langle \mathcal{P}_j, p_d \rangle$ である.

正当な状況 LS からクラス Λ に属するトポロジ変化イベントが起きたとき，プロセス p_i が手続き SuperRootMode を実行し，プロセス p_i をサブルートとする部分木に属する（サブルートを除いた）プロセスは手続き SuperNodeMode か手続き NormalMode を実行し，その他のプロセスは手続き NormalMode を実行している．プロセス p_i は，補題 20 においてプロトコルの A1-2 を実行し，補題 21 においてプロトコルの A1-2 と A2 を実行し，補題 22 においてプロトコルの A3-3 を実行している．また，プロセス p_i をサブルートとする部分木に属する（サブルートを除いた）プロセスは，補題 21 において手続き NormalMode かプロトコルの A4-1,A4-2,A5 を実行し，補題 22 においてさらにプロトコルの A6-3 を実行している．つまり，正当な状況 LS から状況 Σ_3 の間に，経路情報を変更したプロセスは存在しなくて，かつ述語 $M_3()$ は真である.

補題 22 より，必ずプロセス p_j に $R_p.\mathcal{SI} = \text{Decide}$ であるレジスタ構造体 R_p が中継される．プロセス p_j は，親プロセス p_{i+l} からのレジスタに対してレジスタ構

造体 R_p を読み込むので，$G_4(), G_5()$ は偽であり，$G_6()$ が真である．そしてプロトコルより，関数 NextPathNode$(\mathcal{P}_j, \mathcal{P}_D, \mathcal{R}_j)$ が返す値は必ずプロセス p_d であり，かつそのプロセスとは親子関係は成立していない $(p_j \not\leftarrow p_d)$ ので，必ずプロトコル A6-2 を実行する．そして，プロトコル A6-2 が実行されれば，プロセス p_j の経路情報は変更される．

よって，補題は成り立つ． □

補題 24. $\forall c_i^0 \in \Sigma_4$ において，SuperRootMode を実行可能なプロセスを p_i とし，p_i がひとつだけ選択した $R_c.\mathcal{SI} =$ Discov であるレジスタ構造体 R_c を初めて送信したプロセスを p_j とする．

c_i^0 を初期状況とした任意の実行 $E = c_i^0, c_i^1, \cdots, c_i^k, \cdots$ において，プロセス p_i が経路情報を変更する状況 $c_i^k (k > 0)$ が存在する（このような状況の集合を Σ_5 とする）．

証明. プロセス $p_i, p_{i+1}, \cdots, p_{i+l-1}, p_{i+l}$ が p_i をサブルートとした部分木に属していて，かつ $p_i \leftarrow p_{i+1} \leftarrow \cdots \leftarrow p_{i+l-1} \leftarrow p_{i+l}$ であるネットワーク状況を考える．ここで，p_i がひとつだけ選択した $R_c.\mathcal{SI} =$ Discov であるレジスタ構造体 R_c を初めて送信したプロセスが $p_{i+l} = p_j$ であり，その代替リンク先のプロセスを p_d であるとする．そうすれば，代替リンク経路情報 $\mathcal{P}_D$ は $\mathcal{P}_D = \langle \mathcal{P}_j, p_d \rangle$ である．

正当な状況 LS からクラス Λ に属するトポロジ変化イベントが起きたとき，プロセス p_i が手続き SuperRootMode を実行し，プロセス p_i をサブルートとする部分木に属する（サブルートを除いた）プロセスは手続き SuperNodeMode か手続き NormalMode を実行し，その他のプロセスは手続き NormalMode を実行している．プロセス p_i は，補題 20 においてプロトコルの A1-2 を実行し，補題 21 においてプロトコルの A1-2 と A2 を実行し，補題 22 においてプロトコルの A3-3 を実行している．また，プロセス p_i をサブルートとする部分木に属する（サブルートを除いた）プロセスは，補題 21 において p_i 手続き NormalMode かプロトコルの A4-1,A4-2,A5 を実行し，補題 22 ではさらにプロトコルの A6-3 を実行している．そして，プロセス p_j は補題 23 においてプロトコルの A6-2 を実行して，経路情報は $\mathcal{P}_j = \langle \mathcal{P}_d, p_j \rangle$ に変更するが，代替リンク先のプロセスの経路情報は実経路情報なので，$\mathcal{P}_j$ も実経路情報になる（プロセス p_r をサブルートとする部分木に属する）ので，経路情報 $\mathcal{P}_j$ は常に $\mathcal{P}_j \neq \perp$ である．つまり，正当な状況 LS から状況 Σ_3 の間に，経路情報が $\perp$ であるプロセスは存在しないので，述語 $M_3()$ は真である．

$l = 0$ のときは，代替リンクをもつプロセスとプロセス p_i が一致するなら，補題 20 において，A1-1 を実行しているはずなので，ありえない．

$l = 1$ のときを考える．そうすれば，$p_i \leftarrow p_j$ であり，代替リンク経路情報 $\mathcal{P}_D$ は $\mathcal{P}_D = \langle \mathcal{P}_i, p_j, p_d \rangle$ である．そして，補題 23 より，プロセス p_j は，代替リンク先のプロセスを新しい親プロセスとして親子関係を成立させるので，$p_i \not\leftarrow p_j$ になる．そして p_d は，特別なプロセス p_r をサブルートとする部分木に属しているので，p_d は p_j に対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を書き込んでいる．よって，プロセス p_j は，E において，手続き NormalMode を実行するのでプロセス p_j の経路情報は変更

されない．よって，プロセス p_i は，補題 23 より，$\mathcal{S}_i = (\mathrm{Decide}, \mathcal{P}_D)$ であるので，述語 $G_1(), G_2()$ は偽であり，$G_3()$ が真である．よって，関数 $\mathrm{NextPathNode}(\mathcal{P}_D, \mathcal{R}_i)$ が返す値は p_j であるが，$p_i \not\leftarrow p_j$ より，プロセス p_i はプロトコル A3-2 を実行する．そうすれば，プロセス p_i は p_j を親とした親子関係が成立する（経路情報を変更する）．

ここで，$l = m(m > 0)$ のとき，補題が正しいと仮定して，$l = m + 1$ のときを考える．ここでも、補題 23 より、プロセス $p_j = p_{i+m+1}$ は代替リンク先のプロセスを新しい親プロセスとして親子関係を成立させるので，$p_{i+m} \not\leftarrow p_j$ になる．そして p_d は，特別なプロセス p_r をサブルートとする部分木に属しているので，p_d は p_j に対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を書き込んでいる．よって，プロセス p_j は，E において，手続き NormalMode を実行するのでプロセス p_j の経路情報は変更されない．そして、プロセス p_{i+m} が 1 原子動作すると，補題 22 よりプロセス p_{i+m} は親プロセス p_{i+m-1} からのレジスタに対して $R_p.\mathcal{S} = (\mathrm{Decide}, \mathcal{P}_D)$ であるレジスタ構造体 R_p を読み込む．よって，述語 $G_4(), G_5()$ は偽であり，述語 $G_6()$ が真である．そうすれば，関数 $\mathrm{NextPathNode}(\mathcal{P}_D, \mathcal{R}_{i+m})$ が返す値は p_j であるが，$p_{i+m} \not\leftarrow p_j$ より，プロセス p_{i+m} はプロトコル A6-2 を実行し，プロセス p_{i+m} は p_j を親とした親子関係は成立させる（経路情報を変更する）．そして，プロセス p_{i+m} は，経路情報を変更するので，プロセス p_{i+m-1} と親子関係は成立しなくなる（$p_{i+m-1} \not\leftarrow p_{i+m}$）．

以上より，$l = m(m > 0)$ が正しいなら，$l = m + 1$ も成り立つので，帰納法により補題は成り立つ． □

補題 25. $\forall c_i \in \Sigma_5$ において，親子関係を変更するプロセスは，$p_i \leftarrow \cdots \leftarrow p_j$ のプロセスであり，かつそれらのみである．

証明. 補題 22，補題 23 と補題 24 より，明らかである． □

補題 26. $\forall c_i^0 \in \Sigma_5$ を初期状況とした任意の実行 $E = c_i^0, c_i^1, \cdots, c_i^k, \cdots$ において，次が満たされている．

1. $\forall c_i^k \in E, k \geq 0$ において，親子関係が変更されない
2. $\exists m, c_i^l \in LS, (l \geq m)$

証明. これより，二つの補題を証明する．

1. 背理法により,「親子関係は変更される」と仮定する．

   補題 24 より，サブルートであったプロセス（以下 p_i）は，プロトコル A3-2 を実行したので，$\mathcal{S}_i = \perp$ に状態が遷移し，全隣接プロセスへのレジスタに対して $R.\mathcal{S} = \perp$ であるレジスタ構造体 R を書き込む．

   よって，プロセス p_i が親子関係を変更する前まで，プロセス p_i の子プロセスだったプロセス (以下 p_j) はプロセス p_i が親子関係が成立しなくなる．

しかし，プロセス p_j の経路情報 $\mathcal{P}_j$ は $\mathcal{P}_j = \langle \mathcal{P}_{i(old)}, p_j \rangle$ （ただし，$\mathcal{P}_{i(old)}$ はプロセス p_i が親子関係を変更する前の経路情報）であり，補題 11 より，$\mathrm{Last}(\mathcal{P}_{i(old)}) = p_i$ である．よって，プロセス p_j が 1 原子動作すると，$\mathcal{S}_j = \perp$ であり，親プロセスがなくなったので，手続き NormalMode を実行する（$\neg M_1() \wedge \neg M_2()$ である）．プロセス p_j の手続き NormalMode では，経路情報 $\mathcal{P}_j = \langle \mathcal{P}_{i(old)}, p_j \rangle, \mathrm{Last}(\mathcal{P}_{i(old)}) = p_i$ より $R_{tmp} := \mathrm{Parent}(\mathcal{P}_j, \mathcal{R}_j)$ は必ず $R_{tmp}.ID = p_i$ である．

よって，プロセス p_j は新しく経路情報を更新したプロセス p_i と再び親子関係を成立させるので,「親子関係が変更する」という仮定に矛盾している．よって,「親子関係が変更されない」は成り立つ．

2. 上記の 1 を繰り返していくと，元々プロセス p_i をサブルートとした部分木に属する全プロセスは，手続き NormalMode を実行すると，全レジスタ $\forall R_{xy}(\forall p_x, p_y \in P, (p_x, p_y) \in L)$ に対して $R_{xy}.\mathcal{S} = \perp$ となり，これは補題 15 の条件に当てはまる．

   以上，補題 15 より，正当な状況に到達し，定理 1 より閉包性が保障されるので，$\exists m, c_i^l \in LS, (l \geq m)$ である m は存在する．

以上より，補題は成り立つ． □

補題 27. 正当な状況 LS からクラス Λ に属するトポロジ変化イベント ε が生じたときの状況から，正当な状況 LS になるまでにトポロジ変化イベントが起きなければ，*Passage* 述語は満たされている．

証明. 補題 19，補題 20，補題 21，補題 23，補題 24，補題 25 と補題 26 より，正当な状況 LS からクラス Λ に属するトポロジ変化イベント ε が生じたときの状況から Passage 述語は満たされているので，補題は成り立つ． □

よって，定理 2 と補題 27 より，次の定理が言える．

定理 3. 提案プロトコルは，クラス Λ に属するトポロジ変化イベントに対して，生成木構成問題を解く強安定プロトコルである． □

第4章　まとめ

本稿では，生成木構成問題を解く強安定プロトコルを提案した．Dolev ら [4] の強安定プロトコルは生成木の枝リンクの消滅を考慮していない．一方，本プロトコルは，正当な状況から，生成木を構成しているリンクの中での単一リンクの消滅に対し,「消滅した生成木を構成するリンクと（部分木 $\mathcal{T}_0, \mathcal{T}_1$ 間の）代替リンクの間に存在するプロセス（$\mathcal{T}_1$ に属する）のみが親子関係を変更させるだけであり，残りのプロセスの親子関係は変化させない」という条件を満たしながら再安定する．これは対応できるトポロジ変化がより一般的になった点で優れている（図 4.1 参照）．

特に，生成木を構成しているリンクの中での単一リンクの消滅に対して Passage 述語を満たすということは，再安定実行中の生成木の形は消滅したリンクと代替リンク以外は変化しないことになる．つまり，生成木を利用するプロトコル（上位プロトコル：たとえば生成木上をトークンが巡回するプロトコルなど）に対して，提案プロトコルが動的ネットワークのトポロジ変化イベントによる故障を限定・隠蔽していることになる．よって，上位プロトコルがトポロジ変化イベントを考慮する必要なく，動的ネットワークに実装できることになる．

そして，アドホックネットワークにおいても，トポロジ変化が Λ に属するなら，提案プロトコルにより，アドホックネットワークに生成木を構築できることがいえる．そうなれば，先ほどと同様にいくつかの分散プロトコルをアドホックネットワークに実装可能になる利点がある．

4.1　今後の課題

提案プロトコルは，既存のプロトコルに対して，トポロジ変化がより一般的になった点で有効であるが，まだアドホックネットワークに用いるには，いくつかの問題点が存在する．

1. 特別なプロセスが存在しないといけない

   生成木を構築するので，特別なプロセスが存在しなければならない．そうなると，アドホックネットワークがいくつかのネットワークに分裂した場合は，特別なプロセスは唯一なので，生成木を構築できないネットワークができてしまう．また，アドホックネットワークは P2P ネットワークの一種なので，

プロトコル	故障の種類	安全条件
既知の研究 Dolevら	生成木の枝以外の消滅	安全条件①を満たす
	生成木の枝の消滅	**×（規定されていない）**
提案プロトコル	生成木の枝以外の消滅	安全条件①を満たす
	生成木の枝の消滅	**安全条件②を満たす**

安全条件①＝　「生成木の構成が変更されない」

安全条件②＝　「消滅した生成木を構成するリンクと代替リンクの間に存在するプロセスのみが親子関係を変更させるだけであり、残りのプロセスの親子関係は変化させない」

図 4.1: 生成木構成問題を解く既存プロトコルとの比較

特別なプロセスそのものが存在することは，P2P ネットワークの意義に反するのが問題である．

2. 再安定するまで，トポロジ変化は起きてはいけない

   リンクの消滅によって，分裂した２つの部分木を区別するのに，経路情報を用いているので，必ず区別するには経路情報は常に正しくなければならない．よって，頻繁にトポロジ変化イベントが起きるネットワークでは，再安定する前に次のトポロジ変化イベントが起きてしまうと，もう Passage 述語を満たすことができなくなる．

3. 経路情報がプロセス数 n に比例する

   ネットワーク中のプロセス数が増大すると，それだけ多くのメモリ空間が必要になる．経路情報を用いずに，代替リンクの発見，親子関係の変更ができるプロトコルの開発が望まれる．

謝辞

本研究の一部は，平成16年度日本学術振興会科学研究補助金（基盤研究(C)16500028，若手(B)16700010）の研究助成によるものである．

関連図書

[1] E.W.Dijkstra, "Self-Stabilizing systems in Spite of Distributed Control,"CACM 17 pp.643-644,1974.

[2] S.Ikeda, I.Kubo, N.Okumoto and M.Yamashita, "Fiar Circulation of a Token", IEEE Trans. on Parallel and Distributed Systems, Vol.13, No.4, pp.367-372,2002.

[3] N.Chen, H.Yu, and S.Huang, "A self-stabilizing algorithm for constructing spanning trees," Information Processing Letters vol.39 pp.147-151,1991.

[4] S.Dolev and T.Herman, "Superstabilizing Protocols for Dynamic Distributed Systems," Proc. 2nd Workshop on Self-Stabilizing Systems, pp.3.1-3.15,1995.

[5] T.Herman, "Superstaibilizing mutual exclusion," Proc.International Conf. on Parrallel and Distributed Systems, pp.31-40,1995.

[6] Y.Katayama, E.Ueda, H.Fujiwara and T.Masuzawa, "A Latency Optimal Superstabilizing Mutual Exclusion Protocol in Unidirectional Rings", JPDC, No.62, pp.865-884, 2002.

[7] S.Gosh, A.Gupta, and S.V.Pemmaraju, "Fault-containing Self-Stabilizing Algorithm for Spanning Trees," Proc. 40th PODC, pp.45-54, 1996.

[8] 片山喜章, 増澤利光,"重み最小生成木を構成する故障封じ込め自己安定プロトコル," D-1 Vol.J84-D-1 No.9 pp.1307-1317, 2001.

[9] Mobile Ad Hoc Networks(MANET) "RFC2501" Network WG 1999

[10] S.Kutten and B.Patt-Shamir, "Time-adaptive self stabilization," Proc. 16th PODC, pp.140-158, 1997