プログラミング-言語・基礎・実践- 21－8
(1995. 3. 9)

均一な木の上での自己安定リーダ選出プログラムの形式的導出

網崎 孝志,* 辻野 嘉宏,** 都倉信樹**
* 島根大学理学部情報科学科
〒690 松江市西川津町 1060
E-mail: ami@cis.shimane-u.ac.jp
** 大阪大学基礎工学部情報工学科
〒560 豊中市待兼山 1-3
E-mail: {tsujino,tokura}@ics.es.osaka-u.ac.jp

あらまし 分散アルゴリズムは, それが対象とするシステムに対称性がある場合, 確率的な遷移を余儀なくされることがある. 一般に分散アルゴリズムの設計および検証は繁雑でその過程に誤りが混入しやすいため, 各種の形式的で厳密な手法が提案されてきた. 最近, Rao は確率的プログラムの推論を行なえるようUNITY を拡張したが, その本格的な適用例はいまのところ報告されていない. 我々は, この拡張されたUNITY の形式体系のもとで, 均一な木ネットワーク上でのリーダ選出問題を解く確率的な自己安定プログラムが導出できることを示す.

キーワード アルゴリズムの導出, 分散アルゴリズム, 確率的アルゴリズム, リーダ選出, UNITY

Formal derivation of a self-stabilizing program: Leader election on a uniform tree

Takashi AMISAKI,* Yoshihiro TSUJINO** and Nobuki TOKURA**
* Department of Computer and Information Science, Shimane University
1060 Nishikawatsu, Matsue 690, Japan.
E-mail: ami@cis.shimane-u.ac.jp
** Department of Information and Computer Sciences, Osaka university
1-3 Machikaneyama, Toyonaka 560, Japan.
E-mail: {tsujino,tokura}@ics.es.osaka-u.ac.jp

Abstract Probabilistic programs are effective for several kinds of distributed problems, especially for problems involving the resolution of symmetry, where no deterministic counterparts exist. This is also the case with the distributed program for leader election on a uniform tree network, because probabilistic transitions should be made to break the possible symmetry of the tree. In this paper, we report a formal derivation of a probabilistic distributed program for this problem, using the UNITY formalism which has been recently extended by Rao to treat probabilistic programs.

key words Derivation of distributed algorithms, probabilistic algorithms, leader election, UNITY

1 はじめに

並行プログラムの設計や検証にいくつかの形式的かつ厳密な手法が試みられており, その中の一つに表明型推論 (assertional reasoning) を用いた手法がある. この種の手法として分類されるものにはいくつかのものがあるが, そのいずれにおいても, プログラムに対する表明は安全性に関するものと前進性に関するものの二つに大別される. 直観的には, 安全性にかかわる表明はプログラムが違反してはならないことを表すために, 前進性にかかわる表明はプログラムが達成しなければならないことを表すために用いる.

Chandy と Misra により開発された UNITY も表明型推論を用いた手法の一つである.[1] UNITY の特徴としては, 実行系列に関する議論が不要であること, およびプログラムの事後検証だけでなく導出をも目的としていることなどが挙げられる. UNITY は分散システムに対する計算モデルおよび証明論理から構成されている. UNITY プログラムは条件付き多重代入文の集合であるが, UNITY の計算モデルとは, その各多重代入文を非決定的にかつ "条件なし公平性" のもとで選択し原子的に実行するというものである. また, UNITY の証明論理は一種の線形時相論理であるが, **ensures** なる演算子に "条件なし公平性" のもとでのプログラムの基本的な前進性を保証する役割を持たせているため, プログラムの実行系列とは無関係に前進性を議論することができる.

ところで, 並行プログラムは対象とするシステムに対称性がある場合, 確率的なふるまいを余儀なくされることがある. "条件なし公平性" なる概念が分散システムに対する UNITY の計算モデルで重要な役割を果たしていると述べたが, 確率的プログラムに対しては, Pnueli により "極度の公平性" なる概念が提唱されている.[2] この概念のもとでは, 確率的並行プログラムの実行をモデル化することができ, その正当性に関する表明型推論がおこなえるようになる.

最近, Rao はこの "極度の公平性" を取り込むことにより, UNITY の形式体系を拡張した.[3] その結果, 確率的並行プログラムの性質のうち, 決定性あるいは確率 1 で成立するような性質についての表明型推論が行なえるようになった. このように対象とする性質は限定されるが, その代わりに各々の確率的事象の具体的な生起確率とは無関係にプログラムの正しさに関する推論を行なえる. この拡張された UNITY は, 自己安定をはじめ, 相互排除, リーダ選出, 非同期合意などの分散問題に適用可能であると予想されている. しかし, 著者らの知るかぎり, これまでのところ, この拡張された UNITY を用いた本格的な適用例はなく, 実際の確率的並行プログラムの導出が現実的に行なえるのかは確かめられていない.

本稿では, 拡張された UNITY を用いて, 均一な木ネットワーク上でリーダ選出問題を解く自己安定プログラムを導出することが可能であることを示す. このプログラムの基本的なアイデアは, 我々の研究室から過去に報告したアルゴリズム[4]と同一である. すなわち, 木の中心を求めるプログラムと, 二つのプロセスのリーダを選出するプログラムの合成により目的のプログラムを構築する. 前者は決定性の自己安定プログラム, 後者は確率的自己安定プログラムとなる. 本稿ではこれらのプログラムを UNITY の形式体系の下で導出する. また, 両者の合成には, UNITY のプログラム構造化法の一つである集合和合成を用いる. 導出されたプログラムの正当性はその構築法から保証される. なお, 導出の過程で我々が行なう検証は, そのすべてを述語計算の範囲内で行なうことができる. この意味で本稿でのプログラム導出は純粋に形式的である. 紙面の都合上, 検証のすべてを示すことはできないが詳細については別に報告する.[6]

2 表記法

本稿では述語やプログラムの記述に以下の量化表現を用いる:

$$\langle \oplus x : r.x : t.x \rangle.$$

ここで, $\oplus$ は結合的かつ可換な二項演算子である. また, 中置のドットは関数適用をあらわし, $x, r.x, t.x$ をそれぞれダミー, レンジ, 項という. 上記の量化表現はそのレンジを充足するようなすべての $x_1, x_2, \cdots, x_n$ について式 $t.x_1 \oplus t.x_2 \oplus \cdots \oplus t.x_n$ を評価した結果を表す. 例えば, $\langle \exists i : 0 \leq i \leq N : A[i] \rangle$ は $A[0..N]$ のどれかの要素が真の場合に真となり, $\langle \sum i : 0 \leq i \leq N \wedge A[i] < A[j] : 1 \rangle$ は $A[0..N]$ の要素のなかで $A[j]$ よりも小さいものの総数を表す. なお, レンジが空, つまり *false* の場合, 上記の量化表現が表す値は$\oplus$の単位元である. また, 文脈から明らかな場合はレンジを省略することがある. なお, 上述の例のように, 演算子 $\oplus$ が $+, \wedge, \vee, \uparrow$ の場合は, それぞれ $\sum, \forall, \exists, \max$ と表記する. ここで $\uparrow$ は二つの自然数のうち小さくない方を与える演算子であり, その単位元は 0 である.

3 UNITYの概要

拡張されたUNITYについて概説する．詳細については文献[1]と[3]を参照されたい．なお，以下に記述するUNITYは，本稿での目的に適するよう，元来のものを一部簡素化したものである．

UNITYプログラム

拡張されたUNITYプログラム F は確率的多重代入文 S_i の集合であり，以下のように表す:

$F \equiv \langle \mathbin{[]} i :: S_i \rangle$.

ここで $[]$ は代入文の集合和を与える2項演算子であると同時に二つの文の間のセパレータの役割をもつ．確率的多重代入文 S_i は以下のように表せる:

$x := e_0 \mid e_1 \mid \cdots \mid e_{k-1}$ **if** b,

ここで，x は変数のリスト，各 $e_j (0 \leq j < k)$ は式のリスト，b はブール式である．$k = 1$ の場合，S_i は通常の条件付き多重代入文である．

UNITYプログラムの実行とは，いずれかの S_i を一つ選択して実行することを永遠に繰り返すことである．この際，どの文も無限にしばしば実行され，ある特定の文が永遠に無視されることはないとする．この制約を“条件なし公平性”(unconditional fairness)という．また，各々の S_i の実行とは，b が真のときにかぎり，いずれか一つの j について $x := e_j$ を実行することであり，この k 個の多重代入文のそれぞれを S_i のモードという．但し，S_i のどのモードが実行されるかは，プログラムの状態に依存しないものとする．このことを“極度の公平性”(extreme fairness)，あるいは プログラムの実行が極めて公平であるという．なお，状態とはプログラムに出現する変数の対象領域の直積である．

どの S_i のどのモードを実行しても状態が変化しないとき，そのプログラムは不動点に到達したという．写像 **FP** を

$\mathbf{FP}.F \equiv \langle \forall i : S_i \in F : eq.S_i \rangle$,

で定義する．ただし，$eq.S_i$ は S_i の“:=”を“=”で，“**if**”を“$\Leftarrow$”で置換して得られる述語である．すると F の不動点とは $\mathbf{FP}.F$ を充足する状態と定義される．

UNITY演算子とその性質

UNITY演算子を定義するために二種類の述語変換子 (predicate transformers) [5, 3]を導入する．まず，述語変換子 $\mathbf{wp}.S_i$ を次のように定義する:

$\mathbf{wp}.S_i.X \equiv b \Rightarrow \langle \forall j : 0 \leq j < k : X[x := e_j] \rangle$,

ここで X はプログラムの状態の上での任意の述語である．また $X[x := e_j]$ は X に自由に出現する全ての x を e_j で置換した述語をあらわす．この述語 $\mathbf{wp}.S_i.X$ を X に関する S_i の最弱事前条件という．次に，述語変換子 $\mathbf{wpp}.S_i$ は次のように定義される:

$\mathbf{wpp}.S_i.X \equiv b \Rightarrow \langle \exists j : 0 \leq j < k : X[x := e_j] \rangle$.

この $\mathbf{wpp}.S_i.X$ を X に関する S_i の確率的最弱事前条件という．

安全性に関する基本的な演算子には**unless**, **upto**, **stable** があり，上述の述語変換子を用いて以下のように定義される:

X **unless** $Y \equiv$
$\langle \forall i : S_i \in F : X \wedge \neg Y \Rightarrow \mathbf{wp}.S_i.(X \vee Y) \rangle$,
X **upto** $Y \equiv$
$\langle \forall i : S_i \in F : X \wedge \neg Y \Rightarrow \mathbf{wp}.S_i.X \vee \mathbf{wpp}.S_i.Y \rangle$,
stable $X \equiv X$ **unless** *false*.

X **unless** Y を例にその直観的な意味を述べると，プログラムが X を充足する状態に到達すると，それ以後，Y が成り立つまでは Xが成り立つことを表している．従って **stable** X は，一旦 Xが成り立つとそれ以後永遠に Xが成り立つことをあらわす．例えば，**stable** $x \geq k$ と $x = k$ **unless** $x > k$は共に x が減少しないことを表明する．

条件なし公平性のもとでは，前進性に寄与する文が一つでもあり，かつその他の文がそれを阻害しなければ，プログラムの前進性が保証される．このことを表明するための演算子が **ensures** および **entails** である:

X **ensures** $Y \equiv$
X **unless** $Y \wedge$
$\langle \exists i : S_i \in F : X \wedge \neg Y \Rightarrow \mathbf{wp}.S_i.Y \rangle$,
X **entails** $Y \equiv$
X **upto** $Y \wedge \langle \exists i : S_i \in F : X \wedge \neg Y \Rightarrow \mathbf{wpp}.S_i.Y \rangle$.

例えば，$x = k$ **ensures** $x > k$は x が単調に増大することを表明している．

ensures と **entails** はその定義よりプログラムテキストに直接対応するため，詳細化の後半の段階で登場することが多い．これに対し，詳細化の初期段階で前進性を表明するために用いる演算子として $\mapsto$, $\rightsquigarrow$, $\models\!\!\Rightarrow$ がある．$\mapsto$は以下のように定義される:

$\frac{X \textbf{ ensures } Y}{X \mapsto Y}$ (基底), $\frac{X \mapsto Y \quad Y \mapsto Z}{X \mapsto Z}$ (推移),
$\frac{\langle \forall X : X \in W : X \mapsto Y \rangle}{\langle \exists X : X \in W : X \rangle \mapsto Y}$ (離接).

$\models\!\!\Longrightarrow$の定義では, $\mapsto$の定義のうち基底だけを次のもので置き換える:

$\frac{X \textbf{ unless } Y \quad X \rightsquigarrow Y}{X \models\!\!\Longrightarrow Y}$ (基底).

直観的には, $X \mapsto Y$ ($X \models\!\!\Longrightarrow Y$) は X が充足される状態に到達すると, いずれは Y が充足される (確率 1 で充足される) ことを表している. なお, $\rightsquigarrow$ の定義は以下のとおりである:

$\frac{X \textbf{ entails } Y}{X \rightsquigarrow Y}$ (基底), $\frac{X \mapsto Y}{X \rightsquigarrow Y}$ (Leads-to),
$\frac{X \rightsquigarrow Y \quad Y \rightsquigarrow Z}{X \rightsquigarrow Z}$ (推移).

UNITY 演算子の性質[1, 3]の一部を列挙する.

- **unless** の一般合接:
$$\frac{\langle \forall i :: X.i \textbf{ unless } Y.i \rangle}{\langle \forall i :: X.i \rangle \textbf{ unless } \langle \forall i :: X.i \vee Y.i \rangle \wedge \langle \exists i :: Y.i \rangle}$$

- PSP(前進-安全-前進):
$$\frac{X \mapsto Y \quad U \textbf{ unless } V}{(X \wedge U) \mapsto (Y \wedge U) \vee V}$$

- 完備化 (Completion):
$$\frac{\langle \forall i :: X.i \mapsto Y.i \rangle \quad \langle \forall i :: Y.i \textbf{ unless } Z \rangle}{\langle \forall i :: X.i \rangle \mapsto \langle \forall i :: Y.i \rangle \vee Z}$$

- 帰納原理: $(W, \prec)$ を整礎な順序集合とする. M を プログラムの状態から W への写像とすると,
$$\frac{\langle \forall m : m \in W : (X \wedge M = m) \mapsto M \prec m \rangle}{true \mapsto \neg X}$$

プログラム合成

集合和合成とは, 二つのプログラム F と G の集合和をとることであり, $F \,[]\, G$ と表す. 集合和合成に関してはいくつかの定理があるが, その直観的な概略を述べると, X **unless** Y (X **upto** Y) がいずれのプログラムでも成り立つなら合成プログラムでも成り立つ, さらに一方で X **ensures** Y (X **entails** Y) が成り立てば合成プログラムでもそれが成り立つ.

仕様と詳細化

UNITY の形式体系では, 仕様とは UNITY 演算子で記述した表明の集まりをいう. また, 仕様 P が仕様 Q を含意するとき, Q から P を導出することを仕様の詳細化という. プログラム導出は, 仕様の詳細化をそのプログラムが自明となるまで繰り返すことにより行なう. 詳細化の各段階の正しさの検証には, UNITY 演算子の性質に関する定理を用いる.

4 問題と初期仕様

有限個のプロセスの空でない集合 V および Vの相異なる二つの要素の間のリンクの集合 E, すなわち, $E = \langle u, v : u \in V \wedge v \in V \wedge u \neq v : \{u, v\} \rangle$ を考える. Vを頂点集合, Eを辺の集合とみたてたときのグラフの形状が木であるとき, 二項組 $N = (V, E)$ を木ネットワークという. この木ネットワーク上で唯一のリーダを選出することがここでの問題である. ただし, この問題を解くプログラム L に以下の性質を要求する: すべてのプロセスは固有の識別子をもたず, 同一のプログラムを実行する (均一性). なお, L は各プロセス u の実行するプログラム L_u の合成 $\langle \,[]\, u : u \in V : L_u \rangle$ である; L の実行を任意の状態から開始しても解状態に到達し, 以後その状態は変化しない (自己安定性); そして, プロセス間の情報の交換にはレジスタ通信モデルを仮定する (通信モデルの制約). この通信モデルでは, 各リンク $\{u, v\} \in E$ の各通信方向に一つのレジスタが存在し, 全ての通信はレジスタを介して行なう.

プログラム L に対する基本的なアイデアは木の中心が一個あるいは隣接する二個の頂点であることを利用することである.[4] すなわち, 木の中心を求めるプログラムを G, リンク $e \in E$ に接続する二つのプロセスのリーダを求めるプログラムを F_e とすると, L は合成 $\langle \,[]\, e : e \in E : F_e \rangle \,[]\, G$ とみなすことができる. すると, プログラム G により木の唯一の中心と判定されるプロセス, あるいは G により二つの中心の一つと判定されかつ F_e によりリーダと判定されるプロセスが全体のリーダである. ここで, 各 F_e および G が変数を全く共有しなければ, それぞれのプログラムの性質は合成プログラム L で保存される.[6] そこでこれらのプログラムの間で共有変数の必要性が生じないように詳細化を行なうことにする.

まず, プログラム F_eに対する仕様を与える. なお, ネットワークは均一なので, このプログラムが対象とする二つのプロセスに固有の識別子はないが, 便宜上, x および $\bar{x}$ として区別する. そして, $\bar{x}$に対する x の内部変数 $x.s[\bar{x}]$ と x に対する$\bar{x}$の内部変数$\bar{x}.s[x]$ を導入する. ここで変数 s を配列としているのは, 各 s のこれらの要素がこの F_e に局所的であることを保証するためである. 二つのプロセスのリーダとは当該要素の値が大きい方であると定め, 最初の段階の

仕様として以下の F1 を提案する.

[仕様 F1]
(F1.1) $true \models\!\Longrightarrow \mathbf{FP}.F$
(F1.2) **stable** $\mathbf{FP}.F$
(F1.3) $\mathbf{FP}.F \Rightarrow x.s[\bar{x}] \neq \bar{x}.s[x]$

なお, 表明 F1.1 の演算子が $\mapsto$ ではなく $\models\!\Longrightarrow$ となっているのは, ネットワークが対称のため決定性のプログラムでは解状態に到達し得ないからである.

次に, 木ネットワーク Nの中心のプロセスを求めるプログラム G について考える. 木の中心とは離心率が最小の頂点であるから, G の解状態においては, どのプロセスも自身の離心率を知っており, かつその離心率が N において最小であるかを判定できなければならない. そこで, 最初の段階の仕様として以下の G1 を提案する.

[仕様 G1]
(G1.1) $true \mapsto \mathbf{FP}.G$
(G1.2) **stable** $\mathbf{FP}.G$
(G1.3) $\mathbf{FP}.G \Rightarrow \langle \forall x :: x.e = e_x \rangle$

ここで, $x.e$ はプロセス x に局所的な内部変数, e_xは x の離心率である.

F1 および G1 を初期段階の仕様とすることに問題はないと思われる. ただしこれらは自己安定性および解状態の性質について表明しているのみで, 均一性および通信モデルの制約は陽には述べていない. これらの制約については詳細化の段階で導入する.

今後の詳細化のために離心率の性質について整理しておく.

定義 1 (部分木の高さ) 木からリンク $\{x, y\} \in E$ を除去して得られる木のうち, プロセス x を含む木の x を根としたときの高さを h_x^y とすると, この高さは次のように帰納的に定義される:

$$\langle \forall x, y : \{x, y\} \in E : h_x^y = \langle \max v : \{v, x\} \in E \wedge v \neq y : h_v^x + 1 \rangle \rangle.$$

定義 2 (離心率) プロセス x の離心率 e_x は部分木の高さを用いて次のように定義される:

$$\langle \forall x :: e_x = \langle \max v : \{v, x\} \in E : h_v^x + 1 \rangle \rangle.$$

定理 1 $\langle \forall x :: \langle \forall y : \{x, y\} \in E : e_x = h_x^y \rangle \equiv \langle \forall y : x \neq y : e_x < e_y \rangle \rangle.$

定理 2 $\langle \forall x, y : \{x, y\} \in E : e_x = h_x^y + 1 \equiv e_x = e_y \wedge \langle \forall z : z \neq x \wedge z \neq y : e_x < e_z \rangle \rangle.$

定理 3 $\langle \exists x :: \langle \forall y : y \neq x : e_x < e_y \rangle \rangle \not\equiv \langle \exists x, y : \{x, y\} \in E : e_x = e_y \wedge \langle \forall z : z \neq x \wedge z \neq y : e_x < e_z \rangle \rangle.$

5 2プロセス間リーダ選出プログラム

5.1 レジスタ変数の導入

二種類の詳細化を行なう. 一つは, 目的アーキテクチャとしてレジスタ通信モデルを想定しているので, レジスタ変数 $x.r[\bar{x}]$ および $\bar{x}.r[x]$ を導入することである. 両プロセスはこれらのレジスタ変数を介してそれぞれの内部変数 $x.s[\bar{x}]$ および $\bar{x}.s[x]$ の値を伝達しあうものとする. このように考え, 表明 F1.2 を 以下の F2.3 に詳細化する. もう一種の詳細化は前進性についてであり, **unless** の恒等式 $true$ **unless** p および $\models\!\Longrightarrow$ の定義の "形" が示唆する詳細化の方向に素直に従うことにする.

なお, 現時点で対象としているプログラムはただ一つの F_eであり, 各プロセスの通信相手は特定できる. そこで, 以後, 簡単のために, 配列変数の添字を省略する. また, これ以後用いる自由変数 i, j, k は相異なる自然数とする.

[仕様 F2]
(F2.1) $x.s = \bar{x}.s \rightsquigarrow x.s \neq \bar{x}.s$
(F2.2) $x.s \neq \bar{x}.s \rightsquigarrow \mathbf{FP}.F$
(F2.3) $x.s = i \wedge x.r = i \wedge \bar{x}.r \neq i$ **unless** $x.s = i \wedge x.r = i \wedge \bar{x}.r = i$
(F2.4) $\mathbf{FP}.F \equiv x.s \neq \bar{x}.s \wedge x.s = x.r \wedge \bar{x}.s = \bar{x}.r$

[検証] F1.1 について:
1 $x.s = \bar{x}.s \rightsquigarrow \mathbf{FP}.F$
, F2.1,F2.2,$\rightsquigarrow$ の推移性
2 $true \rightsquigarrow \mathbf{FP}.F$
, 1,F2.2, 有限離接
3 $true$ **unless** $\mathbf{FP}.F$
, 自明
4. $true \models\!\Longrightarrow \mathbf{FP}.F$ — (F1.1)
, 2,3, $\models\!\Longrightarrow$ の定義

F1.2 は F2.3 の合接から得られる. また F1.3 は F2.4 から直ちに従う. □

なお, 以後, $\mathbf{FP}.F$ を $x.s \neq \bar{x}.s \wedge x.s = x.r \wedge \bar{x}.s = \bar{x}.r$ で置き換え, 表明 F2.4 は省略する.

5.2 前進性の詳細化

仕様 F2 に現れる $\leadsto$ は, 抽象的な前進性を表す演算子なので, これをプログラムテキストに直接対応する **ensures** や **entails** で置き換えることにする.

[仕様 F3]
(F3.1) $x.s = i \wedge x.r \neq i$ **ensures** $x.i = i \wedge x.r = i$
(F3.2) $x.s = i \wedge x.r = i \wedge \bar{x}.r \neq i$ **unless**
$x.s = i \wedge x.r = i \wedge \bar{x}.r = i$
(F3.3) $x.s = i \wedge x.r = i \wedge \bar{x}.s = i \wedge \bar{x}.r = i$
ensures $x.s \neq \bar{x}.s$
(F3.4) $x.s = i \wedge x.r = i \wedge \bar{x}.s = j \wedge \bar{x}.r = i$
entails $(x.s = i \wedge x.r = i \wedge \bar{x}.s = j \wedge \bar{x}.r = j) \vee$
$(x.s = k \wedge x.r = i \wedge \bar{x}.s = j \wedge \bar{x}.r = i)$

[検証] 省略. □

5.3 分散化

ここまではリンク e に接続する二つのプロセス x と $\bar{x}$ が共同して実行するプログラム F_e に関する仕様の詳細化を行なってきた. この段階で $F_e = F_x \,[]\, F_{\bar{x}}$ と分解した二つのプログラムに対応する仕様への詳細化, つまり分散化を行なう. アーキテクチャ上の制約を考慮して分散化すると, プロセス x が実行するプログラム F_xに対して以下の仕様 F4 が得られる.

[仕様 F4]
(F4.1) **stable** $\bar{x}.s = i$ **in** F_x
(F4.2) **stable** $\bar{x}.r = i$ **in** F_x
(F4.3) **stable** $x.s = i \wedge x.r = i \wedge \bar{x}.r \neq i$ **in** F_x
(F4.4) $x.s = i \wedge x.r \neq i$ **ensures**
$x.s = i \wedge x.r = i$ **in** F_x
(F4.5) $x.s = i \wedge x.r = i \wedge \bar{x}.r = i$ **ensures**
$x.s \neq i \wedge x.r = i \wedge \bar{x}.r = i$ **in** F_x
(F4.6) $x.s = i \wedge x.r = i$ **upto**
$x.s \neq i \wedge x.r = i \wedge x.s \neq \bar{x}.s$ **in** Fx

プログラム $F_{\bar{x}}$ に対する仕様は上記表明で $x, \bar{x} := \bar{x}, x$ とすれば得られる.
[検証] 省略. □

5.4 プログラム F

この段階で UNITY プログラム F への変換はほぼ明らかになっている. **ensures** を用いた表明 F4.4 と F4.5 をそれぞれ仕様 F4 が表明する安定性に違反しない代入文で実現すればよい. また, これまで行なった詳細化から判断して, $x.s$ などは有界な三値変数でよさそうである. そこで, F4.4 と F4.5 をそれぞれ

$x.r := x.s$ **if** $x.r \neq x.s$, および
$x.s := x.s + 1 \mid x.s + 2$ **if** $x.s = x.r \wedge x.s = \bar{x}.r$

に対応させる. ただし, 加算および比較は 3 を法にして行なうものとする.
[検証] これらの代入文が表明 F4.1-5 に従うことは自明である. また, 第一番目の代入文が表明 F4.6 に従うことも明らかである. ここでは第二番目の代入文 (s とおく) が表明 F4.6 に従うことのみを証明する. 簡単のために, 表明 F4.6 の **upto** の左辺を X, 右辺を Y とおく. s が F4.6 に違反しないことを証明するには,

$X \wedge \neg Y \Rightarrow \mathbf{wp}.s.X \vee \mathbf{wpp}.s.Y$

を示せばよいが, この述語は恒真である. なぜなら, $X \wedge \neg Y$ の仮定のもとで, $\mathbf{wp}.s.X \equiv x.s \neq \bar{x}.r$ および $\mathbf{wpp}.s.Y \equiv x.s = \bar{x}.r$ を述語計算で示すことができる. よって, $\mathbf{wp}.s.X \vee \mathbf{wpp}.s.Y \equiv true$. □

なお前述の二つの代入文は, 二つのプロセス x と $\bar{x}$ からなるネットワークに対してのものである. 木ネットワークのすべてのリンクに対するプログラム F は以下のようになる.

プログラム F
$\langle \,[]\, x : x \in V :$
$\quad \langle \,[]\, y : \{y, x\} \in E :$
$\qquad x.r[y] := x.s[y]$ **if** $x.r[y] \neq x.s[y]$
$\qquad [] \; x.s[y] := x.s[y] + 1 \mid x.s[y] + 2$
$\qquad\qquad$ **if** $x.s[y] = x.r[y] \wedge x.s[y] = y.r[x]$
$\quad \rangle$
$\rangle$

プログラム F の正しさは前述の代入文が仕様 F4 に従うこと, および F に出現するプログラム変数の局所性により保証される.

6 木の中心を求めるプログラム

6.1 レジスタ変数の導入

既に述べたように, 全プロセス x が自分の離心率を内部変数 $x.e$ に保持している状態に到達すれば, その状態で最小の $x.e$ をもつプロセスは一つかあるいは隣接する二つのプロセスである. また, 自分の離心率が最小であるかは部分木の高さと離心率から判定できる (定理 1, 2). 部分木の高さは各リンクに対して一対だけ定義されているので, 各リンク $\{x, y\} \in E$

に対して部分木の高さ h_x^y と h_y^xに対応する一対のレジスタ変数, $x.d[y]$ と $y.d[x]$, を導入する.

また 前進性の表明 G1.1 はその形から判断して, $\mapsto$ の帰納原理を適用し詳細化するのが妥当と思われる.

[仕様 G2]
(G2.1) $\neg P1 \wedge M = k \mapsto M \succ k$
(G2.2) $P1 \mapsto P1 \wedge P2$
(G2.3) **stable** $P1$
(G2.4) **stable** $P1 \wedge P2$
(G2.5) $\mathbf{FP}.G \equiv P1 \wedge P2$
但し, $P1 \equiv$
$\langle \forall x, y :: x.d[y] = \langle \max v : v \neq y : v.d[x] + 1 \rangle \rangle$,
$P2 \equiv \langle \forall x :: x.e = \langle \max v :: v.d[x] + 1 \rangle \rangle$.

また M はプログラムの状態の上での写像で, その値域を W とすると順序$\succ$ のもとで W は整礎であるとする. なお, M の選択は詳細化の最後の段階で行なう.
[検証] G2.4 と G2.5 が G1.2 を含意することは明らか. G1.3 については G2.5 と x を根とする部分木の高さの上での帰納法で証明できる. G1.1 については:

$\neg P1 \wedge M = k \mapsto M \succ k$ — (G2.1)
⇒ { 帰納原理 }
$true \mapsto P1$
⇒ {G2.2 と推移性 }
$true \mapsto P1 \wedge P2$
= {G2.5}
$true \mapsto \mathbf{FP}.G$ — (G1.1) □

これ以後, $\mathbf{FP}.G$ を $P1 \wedge P2$ で置き換え, 表明 G2.5 は省略する.

6.2 前進性についての詳細化

各プロセスへの分散化を意識して詳細化を行なう. これはおおまかにいえば, 仕様 G2 の表明の全称記号を UNITY 演算子の外側に括り出す操作に相当する. また, $\mapsto$ を含む抽象的な前進性表明を **ensures** を用いた表明へ詳細化する.

[仕様 G3]
(G3.1) $\langle \forall x, y ::$
$x.d[y] = i \wedge \beta_x^y = j \wedge M = k$ **ensures** $M \succ k \rangle$
(G3.2) $\langle \forall x :: x.e \neq \alpha_x$ **ensures** $x.e = \alpha_x \rangle$
(G3.3) $\langle \forall x, y : x.d[y] = K[x,y] \wedge \beta_x^y = K[x,y] \wedge \pi_x^y$
unless $x.d[y] = K[x,y] \wedge \neg\pi_x^y \rangle$
(G3.4) $\langle \forall x ::$
$x.e = k \wedge \alpha_x = k$ **unless** $x.e = k \wedge \alpha_x \neq k \rangle$
但し, $\alpha_x = \langle \max v :: v.d[x] + 1 \rangle$,
$\beta_x^y = \langle \max v : v \neq y : v.d[x] + 1 \rangle$,
$\pi_x^y \equiv \langle \forall v : v \neq y : v.d[x] = K[v,x] \rangle$,
また K は自由変数の配列である.

[検証] 仕様 G3 が表明 G2.3 と G2.4 を含意することの検証[6] については割愛する.

G2.1 について:
G3.1
⇒ {↦の定義 }
$\langle \forall x, y :: x.d[y] = i \wedge \beta_x^y = j \wedge M = k \mapsto M \succ k \rangle$
⇒ { 離接 }
$\langle \exists x, y :: x.d[y] = i \wedge \beta_x^y = j \rangle \wedge M = k \mapsto M \succ k$
⇒ { 自由変数除去 }
$\langle \exists x, y :: x.d[y] \neq \beta_x^y \rangle \wedge M = k \mapsto M \succ k$
= {de Morgan}
$\neg \langle \forall x, y :: x.d[y] = \beta_x^y \rangle \wedge M = k \mapsto M \succ k$ — (G2.1)

G2.2 について:
1 $P1$ **unless** *false*
, G2.3 の証明より得られる[6]
2 $\langle \forall x :: x.e \neq \alpha_x \mapsto x.e = \alpha_x \rangle$
, G3.2 より
3 $\langle \forall x :: P1 \wedge x.e \neq \alpha_x \mapsto P1 \wedge x.e = \alpha_x \rangle$
, 1 と 2 の PSP
4 $\langle \forall x :: P1 \wedge x.e = \alpha_x \mapsto P1 \wedge x.e = \alpha_x \rangle$
, 自明
5 $\langle \forall x :: P1 \mapsto P1 \wedge x.e = \alpha_x \rangle$
, 3, 4, 離接
6 $\langle \forall x :: P1 \wedge x.e = \alpha_x$ **unless** *false*$\rangle$
, G2.4 の証明より得られる[6]
7 $P1 \mapsto P1 \wedge \langle \forall x :: x.e = \alpha_x \rangle$ — (G2.2)
, 5, 6, 完備化 □

6.3 分散化

通信モデルの制約の下でプログラムおよびデータを各プロセスに分散化するための最終的な詳細化を行なう. 木ネットワーク上の任意のプロセス x の動作を記述するプログラム G_x として以下の仕様 G4 が得られる.

[仕様 G4]
(G4.1) $\langle \forall y : y \neq x :$ **stable** $y.e = k$ **in** $G_x \rangle$
(G4.2) $\langle \forall y, z : y \neq x :$ **stable** $y.d[z] = k$ **in** $G_x \rangle$

(G4.3) **stable** $x.e = i \wedge \alpha_x = i$ **in** G_x

(G4.4) $\langle \forall y :: \textbf{stable}\ x.d[y] = k \wedge \beta_x^y = k\ \textbf{in}\ G_x \rangle$

(G4.5) $\langle \forall y ::$
$x.d[y] = i \wedge M = k\ \textbf{unless}\ M \succ k\ \textbf{in}\ G_x \rangle$

(G4.6) $M = k\ \textbf{unless}\ M \succ k\ \textbf{in}\ G_x$

(G4.7) $\langle \forall y :: x.d[y] = i \wedge \beta_x^y = j\ \textbf{ensures}$
$x.d[y] = j \wedge \beta_x^y = j\ \textbf{in}\ G_x \rangle$

(G4.8) $x.e \neq \alpha_x\ \textbf{ensures}\ x.e = \alpha_x\ \textbf{in}\ G_x$

[検証] 省略. □

6.4 プログラム G

この段階でプログラム G への変換がほぼ明らかになっている. 仕様 G4 に現れている前進性の表明は G4.7 と G4.8 のみで, それぞれ, $x.d[y] := \beta_x^y$ および $x.e := \alpha_x$ の使用を示唆している. そこで, プログラム G の自然な実現として以下のものが考えられる.

プログラム G
$\langle \Box x : x \in V :$
$\quad \langle \Box y : \{y, x\} \in E :$
$\quad x.d[y] := \langle \max v : \{v, x\} \in E \wedge v \neq y : v.d[x] + 1 \rangle$
$\quad \rangle$
$\quad \Box\ x.e := \langle \max v : \{v, x\} \in E : v.d[x] + 1 \rangle$
$\rangle$

[検証] メトリック M として, 以下のような (h_m+1)-項組を採用する $(h_m = \langle \max u, v : \{u, v\} \in E : h_u^v \rangle)$:

$(\langle \sum u, v : h_u^v = 0 \wedge u.d[v] = \beta_u^v : 1 \rangle,$
$\langle \sum u, v : h_u^v = 1 \wedge u.d[v] = \beta_u^v : 1 \rangle,$
$\vdots$
$\langle \sum u, v : h_u^v = h_m \wedge u.d[v] = \beta_u^v : 1 \rangle)$

プログラム G の状態の上で この M がとりうる値全体の集合 W は (h_m+1)-項組での辞書式順序 $\succ$ のもとに整礎である. このようにメトリック M を定めると, 例えば, 代入文 $x.d[y] := \beta_x^y$ は表明 G4.5 に違反しない. なぜなら, この代入文に対する表明 G4.5 は $x.d[y] = l \wedge M = k \wedge \beta_x^y \neq l \Rightarrow M[x.d[y] := \beta_x^y] \succ k$ と同値であることを述語計算により示すことができ, M の定め方よりこの述語は恒真である. 他のものについても同様に述語計算で検証が行なえる. □

なお, 定理 1, 2 より, プログラム L $(= F \Box G)$ の不動点では, あるプロセス x において, それに隣接する全てのプロセス y に対して $x.e = x.d[y]$ であるか, あるいは隣接するある yについて $x.e = x.d[y] + 1$ かつ $x.s[y] > y.r[x]$ ならば, そのプロセス x がリーダである.

7 おわりに

均一な木ネットワーク上でのリーダ選出問題を解く確率的な自己安定プログラムを, 拡張された UNITY の形式体系で導出することができることを示した. プログラム導出は仕様の詳細化を繰り返すことにより行なった. 各詳細化の正しさの検証は, 仕様やプログラムコードの意味するところを考慮することなく, 述語計算の範囲内で機械的に行なえた. この点で本稿のプログラム導出は形式的であるといえる. さらに, 具体的なプログラムの実行系列や, 各々の確率的遷移の生起する具体的な確率も考慮する必要がなかった. 今回は自己安定プログラムの形式的導出の一例を示したのみであるが, 導出の出発点とした初期仕様は, それ自体を自己安定性の形式的な定義とみなすことができる. これは他の自己安定プログラムの形式的導出の可能性を示唆している.

参考文献

[1] Chandy, K. M., and Misra, J.: *A Foundation of Parallel Program Design*, Addison-Wesley, Reading, Mass. (1988).

[2] Pnueli, A.: On the extremely fair treatment of probabilistic algorithms, In *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing*(Boston, Mass.), pp. 278-290, ACM, New York (1983).

[3] Rao, J. R.: Reasoning about Probabilistic Parallel Programs, *ACM Trans. Program. Lang. Sys.*, Vol. 16, No. 3, pp. 798-842 (1994).

[4] 西川直樹, 増澤利光, 都倉信樹: 相互排除問題を解く均一な自己安定分散アルゴリズム, 電子情報通信学会論文誌 D-I, Vol. J75-D-I, No. 4, pp. 201-209 (1992).

[5] Dijkstra, E. W. and Scholten, C. S.: *Predicate calculus and program semantics*, Springer-Verlag, New York (1990).

[6] 網崎孝志, 辻野 嘉宏, 都倉信樹: 均一な木ネットワーク上での自己安定リーダ選出プログラムの形式的導出, 大阪大学基礎工学部 (1995).