

個体群プロトコルにおける個体数の上界 N を用いた緩安定ランキング

山崎心[†], 江口僚太[†], 笹田大翔[†], 大下福仁^{††}, 井上美智子[†]

[†] 奈良先端科学技術大学院大学

^{††} 福井工業大学

概要

個体群プロトコルモデルはモバイルセンサネットワークの一般的な抽象モデルである。本研究では、エージェントにユニークかつ 0 から連続した整数のランクを割り当てる問題であるランキング問題に焦点を当てる。個体群プロトコルモデルにおいて、エージェントが全体のエージェント数 n を知らない場合、自己安定ランキングプロトコルを設計することは不可能であることが知られている。そこで、自己安定性における閉包性の要件を緩和した自己安定性と、実用上同様の利点を持つ緩安定性を考える。緩安定ランキングプロトコルでは、任意の状況から短時間で正しいランキングを持つ安全な状況に到達し、長時間そのランキングを保つことを保証する。本論文では、エージェントがノード数の上界 N を初期知識として利用する緩安定ランキングプロトコルを提案する。

1 はじめに

個体群プロトコル (Population Protocol, PP) モデル [3] は、受動的に移動し、交流によって状態を更新するエージェント群を扱う。エージェントは識別子を持たず、同じ状態の隣接エージェントを区別できない。本研究では、個体群プロトコルに関する多数の研究と同様に、エージェント間の交流可能性を表す交流グラフは完全グラフであり、スケジューラは各ステップで交流するエージェントのペアを一樣ランダムに選択すると仮定する。PP モデルにおけるランキング問題とは、個体群内のエージェントが 0 から連続するランクを持つことを要求する問題である。自己安定ランキング (Self Stabilizing Ranking, SS-RK) 問題は、任意の状況から始めて、個体群が正確に 0 から連続したランクを持つ安全な状況に到達し、その後は個体群がそのランクを永遠に保持することを要求する。自己安定ランキング問題を解くことで、 n 個のエージェントの中で 1 体のエージェントをリーダーとして選出するリーダー選挙問題 [1, 2, 4, 6, 7, 8, 9, 11, 10] や、 n 個のエージェントを k 個のグループに分割する一様問題 [12, 13, 14] などを解くことができる。具体的には、ランクが 0 のエージェントをリーダーとして出力することでリーダー選挙問題が可解であり、ランクを k の剰余で出力することで、一様 k 分割問題が可解である。その特性から、自己安定ランキング問題を解くことは、リーダー選挙問題や一様 k 分割問題などの様々な問題を解くことができるため、有用であり汎用性が高い。

個体群内のすべてのエージェントが個体群の正確なサイズ n (すなわち、エージェントの数) を知らない限り、自己安定リーダー選挙 (Self Stabilizing leader Election, SS-LE) 問題を解決するプロトコルは存在しないことが知られている [7]。これは、同様の条件で自己安定ランキング問題を解決する自己安定プロトコルが存在しないことを意味する。また、一様 k 分割問題においても同様の条件でプロトコルが存在しないことが知られている [14]。この問題を回避する一つのアプローチは、すべてのエージェントが n の正確な値を知っていると仮定することであり、自己安定問題を解決す

表 1: ランキング問題、リーダー選挙問題に関する自己安定個体群プロトコル（提案手法 P_{RK} は予想される結果）

	Type	Knowledge N	Convergence time	Holding time
[7]	SS-RK	$N = n$	$O(n^2)$	—
[6]	SS-RK	$N = n$	$O(\log n)$	—
[11]	LS-LE	$n \leq N$	$O(N \log N)$	$\Omega(e^N)$
[9]	LS-LE	$n \leq N$	$O(N)$	$\Omega(e^N)$
P_{RK}	LS-RK	$n \leq N$	$O(N \log N)$	$\Omega(e^N)$

るための時間・空間計算量に焦点を当てる [6, 7]. 別のアプローチでは、最終的にすべてのエージェントのうち、ある特定の状態を持つエージェントが存在するかどうかを知らせるオラクルを使用して問題を解決する [5].

自己安定性の閉包性の要件を緩和しながらも、実用上の利点を保持する緩安定性の概念を導入するアプローチが存在する [11]. 具体的には、緩安定は、任意の初期状況から始めて、個体群が比較的短時間で安全な状況に到達することを保証する. その後、問題の仕様（例えば、一意のリーダーを持つこと）は、必ずしも永遠ではないが、十分に長い時間維持されなければならない. [11] では、すべてのエージェントが n の共通の上界 N を知っているとは仮定する緩安定リーダー選挙（Loosely Stabilizing Leader Election, LS-LE）プロトコルが提案されている. このプロトコルは、 $O(N \log N)$ の並行時間内に安全な状況に到達した後、 N の指数関数的な時間にわたって一意のリーダーを維持する. n の上界 N を使用できるという仮定は、 $N = 10n$ のように上界 N が実際のエージェント数 n に対して大幅に上回る場合においてもプロトコルが正しく動作するため、実用的である. PP モデルの多くの結果は、一様ランダムスケジューラを仮定している. すなわち、各ステップで交流するエージェントのペアが一様にランダムに選ばれる. 我々もこの一様性の仮定を採用する. PP モデルでは、収束時間や保持時間などの時間複雑性は、並列時間で評価されることが多い [6, 7, 8, 9, 11, 10]. 並列時間は、期待される交流数を n （エージェントの数）で割ったものとして定義される.

1.1 我々の貢献

Sudo ら [11] が提案した LS-LE プロトコルは、安全な状況に到達するために $O(N \log N)$ の並列時間を必要とし、その後、 $\Omega(e^N)$ の並行時間にわたって安全な状況を維持する. Izumi[9] は、任意の LS-LE プロトコルが安全な状況に到達した後、 $\Omega(e^N)$ の並列時間にわたって一意のリーダーを維持する場合、 $\Omega(N)$ の並列収束時間を必要とすることを証明した. 我々は、 n の上界 N を初期知識として持つ個体群において、リーダー選挙問題より上位の問題であるランキング問題を解く緩安定ランキングプロトコルを提案する. 提案プロトコルは、 $O(N \log N)$ (予想) で安全な状況に到達し、その後、 $\Omega(e^n)$ (予想) 時間、正しいランキングを維持する.

2 諸定義

本節では、計算モデルについて説明する. 個体群はエージェント群を頂点とした交流グラフ $G = (V, E)$ としてモデル化される. また、 $|V| = n$ とする. 交流グラフは完全グラフであると仮定し、任意のエージェントのペア (u, v) が交流できるとする. このとき、 u が交流の送信者であり、 v が受信者となる.

プロトコル $P(Q, Y, T, \pi_{out})$ は, 状態の有限集合 Q , 出力記号の有限集合 Y , 遷移関数 $T: Q \times Q \rightarrow Q \times Q$, および出力関数 $\pi_{out}: Q \rightarrow Y$ から構成される. 2つのエージェントが交流するとき, T はそれらの現在の状態に応じて次の状態を決定する. エージェントの出力は π_{out} によって決定される. 状態 q にあるエージェントの出力は $\pi_{out}(q)$ である.

状況は, すべてのエージェントの状態を出力する写像 $C: V \rightarrow Q$ である. プロトコル P のすべての状況の集合を $\mathcal{C}_{all}(P)$ と表す. 状況 C が交流 $e = (u, v)$ によって C' に変わることを, $C \xrightarrow{P, e} C'$ と表す. このとき, $(C'(u), C'(v)) = T(C(u), C(v))$ であり, $C'(w) = C(w)$ はすべての $w \in V \setminus \{u, v\}$ に対して成り立つ.

スケジュール $\gamma = \gamma_0, \gamma_1, \dots = (u_0, v_0), (u_1, v_1), \dots$ は交流の列である. スケジュールは各時点での交流が発生するかを決定する. すなわち, スケジュール γ の下で時刻 t に交流 γ_t が発生する. 特に, この論文では一様ランダムスケジューラ $\Gamma = \Gamma_0, \Gamma_1, \dots$ を考える. 各 Γ_t は確率変数であり, 任意の $t \geq 0$ および任意の異なる $u, v \in V$ に対して $\Pr(\Gamma_t = (u, v)) = \frac{1}{n(n-1)}$ である. なお, この一様ランダムスケジューラには大文字の Γ を使用し, 決定論的スケジュールには小文字の γ を使用する. 初期状況 C_0 とスケジュール γ が与えられたとき, プロトコル P の実行は $\Xi_P(C_0, \gamma) = C_0, C_1, \dots$ と定義される. このとき, すべての $t \geq 0$ に対して $C_t \xrightarrow{P, \gamma_t} C_{t+1}$ である. 一様ランダムスケジューラの下での実行 $\Xi_P(C_0, \Gamma) = C_0, C_1, \dots$ は, 各 C_i が確率変数である状況の列である. スケジュール $\gamma = \gamma_0, \gamma_1, \dots$ と任意の $t \geq 0$ に対して, エージェント $v \in V$ が γ_t に参加するとは, v が γ_t の送信者または受信者であることを意味する.

個体群プロトコルを解く緩安定プロトコルについて, 期待保持時間と期待収束時間は次のように定義される. 問題 \mathcal{P} の specification を $SC_{\mathcal{P}}$ とする. 任意の $C \in \mathcal{C}_{all}(\mathcal{P})$ について, 実行 $\Xi_{\mathcal{P}}(C)$ が $SC_{\mathcal{P}}$ を満たし続ける交流回数の期待値を期待保持時間といい, $\text{EHT}_{\mathcal{P}}(C, SC_{\mathcal{P}})$ と定義する. つまり, 期待保持時間は, 実行 $\Xi_{\mathcal{P}}(C, \Gamma) = C_0, C_1, \dots, C_x, \dots$ において, C_0 から C_x まで $SC_{\mathcal{P}}$ を満たし続けるような最大値 x の期待値である. 任意の状況集合 $\mathcal{S} \subseteq \mathcal{C}_{all}(\mathcal{P})$, 任意の状況 $C \in \mathcal{C}_{all}(\mathcal{P})$ について, 実行 $\Xi_{\mathcal{P}}(C, \Gamma)$ が開始してから \mathcal{S} に属する状況に遷移するまでの交流回数の期待値を期待収束時間といい, $\text{ECT}_{\mathcal{P}}(C, \mathcal{S})$ と定義する. ある計算が高確率で終了するとは, 確率 $1 - O(n^{-c})$ ($c \geq 1$) で終了することをいう.

Definition 1. 緩安定プロトコルであるとは, $\max_{C \in \mathcal{C}_{all}(\mathcal{P})} \text{ECT}_{\mathcal{P}}(C, \mathcal{S}) \leq \alpha$ かつ $\min_{C \in \mathcal{S}} \text{EHT}_{\mathcal{P}}(C, SC_{\mathcal{P}}) \geq \beta$ を満たすような状況の集合 $\mathcal{S} \subseteq \mathcal{C}_{all}(\mathcal{P})$ が存在することをいう.

3 アルゴリズム

本節では, 提案する緩安定ランキングプロトコル P_{RK} についての概要を述べる.

3.1 緩安定ランキングプロトコル P_{RK}

P_{RK} は, 各エージェントが以下の変数を持つ. 変数 Rank は, エージェントのランクを表し, 0 から N までの値を取る. ランキングを満たす状況では, エージェントのランクは 0 から $n-1$ までの連続した値を取る. maxRank 変数は, エージェントが知っている最も高いランクを表す. protocolPhase 変数は, プロトコル全体のフェーズを表す. seqRank 変数は, エージェントが知っているランクの中で最も高い連続したランクを表す. seqPhase 変数は, SEQUENCE-CHECK 関数のフェーズを表す. 最も高い連続したランクの情報は, 後述する SEQUENCE-CHECK 関数によって, エージェントが全体として連続したランクを持つかを判断するために使用される.

P_{RK} の構成図を図 1 に示す.

表 2: エージェントの各変数と範囲

変数	説明	範囲
Rank	エージェントのランク	$\{0, 1, \dots, N\}$
maxRank	最も高いランク	$\{0, 1, \dots, N\}$
protocolPhase	プロトコル全体のフェーズ変数	$\{\text{Ranking}, \text{pReset}\}$
seqRank	0 から連続したランクの最大値	$\{0, 1, \dots, N\}$
seqPhase	SEQUENCE-CHECK 関数のフェーズ変数	$\{\text{Compute}, \text{Inspect}, \text{sReset}\}$
resetTimer	PROPAGATE-RESET 関数のリセットタイマー	$\{0, 1, \dots, rt_{ep}\}$
delayTimer	PROPAGATE-RESET 関数の遅延タイマー	$\{0, 1, \dots, rt_{de}, st_{de}\}$
zeroTimer	ZERO-CHECK 関数のタイマー	$\{0, 1, \dots, t_{zero}\}$
seqTimer	SEQUENCE-CHECK 関数のタイマー	$\{0, 1, \dots, \max(st_{compute}, st_{inspect}, st_{reset})\}$

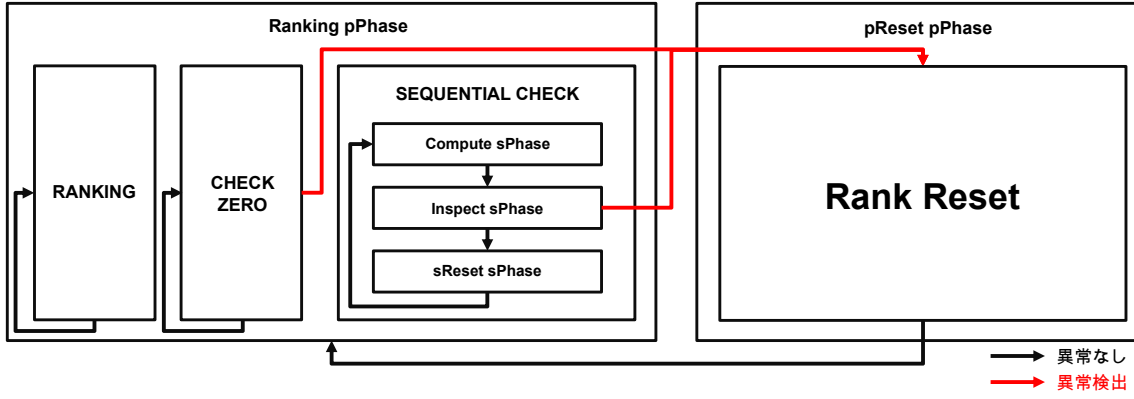


図 1: P_{RK} のフェーズにおける状態遷移図

本プロトコルは、大きく Ranking フェーズと pReset フェーズの 2 つのフェーズに分かれる。Ranking フェーズでは、RANKING 関数により、エージェント全体でランキングを行う。同時に CHECK-ZERO 関数により、ランク 0 を持つエージェントが存在するかを確認する。また、SEQUENCE-CHECK 関数により、エージェントが連続したランクを持つかを確認する。これらのチェック関数により、エージェントが正しいランキングを持つか、あるいはランキングを満たす状況に到達可能であるかを判断し、到達不可能であれば pReset フェーズに移行する。pReset フェーズでは、PROPAGATE-RESET 関数 [6] を用いて全体をリセットする。各関数の詳細については、当日発表にて説明する。

3.2 解析の方針

最後に、提案プロトコル P_{RK} の解析について、今後の方針について述べる。

3.2.1 状況集合の定義

ランキングを満たす状況集合を以下のように定義する。全てのエージェントが異なるランクを持ち、それらが連続している状況の集合を RK とする。また、全てのエージェントの $protocolPhase$

は *Ranking* でなければならない。

$$RK = \left\{ C \in \mathcal{C}_{all} \mid \begin{array}{l} \forall_i \{0, 1 \dots n-1\} \\ \exists j : a_j \cdot rank = i \wedge a_j \cdot protocolPhase = Ranking \end{array} \right\} \quad (1)$$

3.2.2 期待保持時間

本プロトコルでは、*protocolPhase* が *Ranking* である場合に、2つの検証用のサブルーチンである ZERO-CHECK と SEQUENCE-CHECK を実行する。ZERO-CHECK は、いずれかのエージェントがランク 0 を持つかどうかを確認する。SEQUENCE-CHECK は、全てのエージェントが連続したランクを持つかどうかを確認する。従って、プロトコル全体の期待保持時間は、2つのサブプロトコルの期待保持時間が最も短い方に依存する。ZERO-CHECK は、状況 C_0 から始めて、ある時間後にタイムアウトが1度でも発生している確率を失敗確率として扱う。SEQUENCE-CHECK は、状況 C_0 から始めて、各 *seqPhase* における状況遷移の失敗確率を考慮する。

3.2.3 期待収束時間

本プロトコルでは、*RK* に到達不可能な状況において、エラー検出とリセットを行う。プロトコル全体の期待収束時間は、*RK* に到達するまでの時間と、*RK* に到達不可能な状況においてリセットを行う時間に依存する。従って、以下の実行時間を考慮する必要がある。

- ZERO-CHECK 関数のエラー検出時間
- SEQUENCE-CHECK 関数のエラー検出時間
- PROPAGATE-RESET 関数のリセット時間
- RANKING 関数の実行時間

参考文献

- [1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2560–2579, 2017.
- [2] Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proc. of the 42nd International Colloquium on Automata, Languages, and Programming*, pp. 479–491, 2015.
- [3] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distrib. Comput.*, Vol. 18, No. 4, pp. 235–253, 2006.
- [4] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. Vol. 3, No. 4, 2008.
- [5] J. Beauquier, P. Blanchard, J. Burman, and O. Denysyuk. Oracles for self-stabilizing leader election in population protocols. Technical report, INRIA, 2013.
- [6] Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In *PODC 2021: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021.
- [7] S. Cai, T. Izumi, and K. Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory of Computing Systems*, Vol. 50, No. 3, pp. 433–445, 2012.
- [8] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, Vol. 31, No. 4, pp. 257–271, 2018.
- [9] T. Izumi. On space and time complexity of loosely-stabilizing leader election. In *International Colloquium on Structural Information and Communication Complexity*, pp. 299–312, 2015.
- [10] Y. Sudo, T. Masuzawa, A. K. Datta, and L. L. Larmore. The same speed timer in population protocols. In *Proc. 36th IEEE Int. Conf. Distrib. Comput. Syst.*, pp. 252–261, 2016.
- [11] Y. Sudo, J. Nakamura, Y. Yamauchi, F. Ooshita, H. Kakugawa, and T. Masuzawa. Loosely-stabilizing leader election in a population protocol model. *Theoretical Comput. Sci.*, Vol. 444, pp. 100–112, 2012.
- [12] Hiroto Yasumi. Space-optimal population protocols for uniform bipartition under global fairness. *IEICE TRANSACTIONS on Information and Systems*, Vol. 102, No. 3, pp. 454–463, 2019.
- [13] Hiroto Yasumi. A study on uniform k-partition and graph class identification in the population protocol model. *Doctoral Thesis*, 2022.
- [14] Hiroto Yasumi, Fukuhito Ooshita, and Michiko Inoue. Uniform partition in population protocol model under weak fairness. *23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*, 2020.

A 付録 A: R_{RK} の疑似コード

Algorithm 1 pReset the protocol state

```

1: function RESET( $a$ )
2:    $a.\text{Rank} \leftarrow 0$ 
3:    $a.\text{maxRank} \leftarrow 0$ 
4:    $a.\text{seqRank} \leftarrow 0$ 
5:    $a.\text{seqPhase} \leftarrow \text{Compute}$  State  $a.\text{protocolPhase} \leftarrow \text{Ranking}$ 
6:    $a.\text{zeroTimer} \leftarrow t_{\text{zero}}$ 
7:    $a.\text{seqTimer} \leftarrow st_{\text{compute}}$ 

8: function PROPAGATE-RESET( $a_0, a_1$ )
9:   if  $a_0.\text{resetTimer} > 0 \wedge a_1.\text{protocolPhase} \neq \text{pReset}$  then
10:     $a_1.\text{protocolPhase} \leftarrow \text{pReset}$ 
11:     $a_1.\text{resetTimer} \leftarrow 0$ 
12:     $a_1.\text{delaytimer} \leftarrow rt_{\text{de}}$ 
13:   if  $a_1.\text{protocolPhase} = \text{pReset}$  then
14:     $a_0.\text{resetTimer}, a_1.\text{resetTimer} \leftarrow \max(a_0.\text{resetTimer} - 1, a_1.\text{resetTimer} - 1, 0)$ 
15:   for all  $i \in \{a_0, a_1\}$  with  $i.\text{protocolPhase} = \text{pReset} \wedge i.\text{resetTimer} = 0$  do
16:     if  $i.\text{resetTimer}$  just became 0 then
17:        $i.\text{delayTimer} \leftarrow rt_{\text{de}}$ 
18:     else
19:        $i.\text{delayTimer} \leftarrow i.\text{delayTimer} - 1$ 
20:     if  $i.\text{delayTimer} = 0 \vee a_1.\text{protocolPhase} \neq \text{pReset}$  then
21:       executeRESET( $i$ )

```

Algorithm 2 seqCheck

```
1: function seqRANK-COMPUTE( $a_0, a_1$ )
2:   if  $a_0$ .seqPhase = sReset then
3:      $a_0$ .seqPhase  $\leftarrow$  Compute
4:   if  $a_0$ .Rank =  $a_1$ .seqRank + 1 then
5:      $a_0$ .seqRank  $\leftarrow$   $a_0$ .Rank
6:      $a_1$ .seqRank  $\leftarrow$   $a_0$ .Rank
7:   else
8:      $a_0$ .seqRank  $\leftarrow$   $a_1$ .seqRank  $\leftarrow$   $\max(a_0$ .seqRank,  $a_1$ .seqRank)
9:   for all  $i \in \{0, 1\}$  do
10:    if  $a_i$ .Rank = 0 then
11:      if  $a_i$ .seqTimer > 0 then
12:         $a_i$ .seqTimer  $\leftarrow$   $a_i$ .seqTimer - 1
13:      else
14:         $a_i$ .seqPhase  $\leftarrow$  Inspect
15:         $a_i$ .seqTimer  $\leftarrow$   $st_{inspect}$ 

16: function seqRANK-INSPECT( $a_0, a_1$ )
17:   for all  $i \in \{0, 1\}$  do
18:     if  $a_i$ .seqPhase = Compute then
19:        $a_i$ .seqPhase  $\leftarrow$  Inspect
20:     if  $a_i$ .Rank >  $a_i$ .seqRank then
21:        $a_i$ .protocolPhase  $\leftarrow$  pReset
22:        $a_i$ .resetTimer  $\leftarrow$   $rt_{ep}$ 
23:     return
24:   if  $a_i$ .Rank = 0 then
25:     if  $a_i$ .seqTimer > 0 then
26:        $a_i$ .seqTimer  $\leftarrow$   $a_i$ .seqTimer - 1
27:     else
28:        $a_i$ .seqPhase  $\leftarrow$  sReset
29:        $a_i$ .seqTimer  $\leftarrow$   $st_{reset}$ 
30:        $a_i$ .delayTimer  $\leftarrow$   $t_{sde}$ 

31: function seqRANK-RESET( $a_0, a_1$ )
32:   if  $a_0$ .seqTimer > 0  $\wedge$   $a_1$ .seqPhase = Inspect then
33:      $a_1$ .seqPhase  $\leftarrow$  sReset
34:      $a_1$ .seqTimer  $\leftarrow$  0
35:      $a_1$ .delaytimer  $\leftarrow$   $st_{de}$ 
36:   if  $a_1$ .seqPhase = sReset then
37:      $a_0$ .seqTimer,  $a_1$ .seqTimer  $\leftarrow$   $\max(a_0$ .seqTimer - 1,  $a_1$ .seqTimer - 1, 0)
38:   for all  $i \in \{a_0, a_1\}$  with  $i$ .seqPhase = sReset  $\wedge$   $i$ .seqTimer = 0 do
39:     if  $i$ .seqTimer just became 0 then
40:        $i$ .delayTimer  $\leftarrow$   $st_{de}$ 
41:     else
42:        $i$ .delayTimer  $\leftarrow$   $i$ .delayTimer - 1
43:     if  $i$ .delayTimer = 0  $\vee$   $a_1$ .seqPhase  $\neq$  sReset then
44:        $i$ .seqPhase  $\leftarrow$  Compute
45:        $i$ .seqTimer  $\leftarrow$   $st_{compute}$ 
46:        $i$ .seqRank  $\leftarrow$  0
```

Algorithm 3 P_{RK}

Input initiator: a_0 , responder: a_1 are agents

```
1: function ZERO-CHECK( $a_0, a_1$ )
2:   if  $a_0.\text{Rank} = 0 \vee a_1.\text{Rank} = 0$  then
3:      $a_0.\text{zeroTimer} \leftarrow a_1.\text{zeroTimer} \leftarrow t_{zero}$ 
4:   else
5:      $a_0.\text{zeroTimer} \leftarrow a_1.\text{zeroTimer} \leftarrow \max(a_0.\text{zeroTimer} - 1, a_1.\text{zeroTimer} - 1, 0)$ 
6:   for all  $i \in \{a_0, a_1\}$  with  $i.\text{zeroTimer} = 0$  do
7:      $i.\text{protocolPhase} \leftarrow pReset$ 
8:      $i.\text{seqTimer} \leftarrow rt_{ep}$ 

9: function SEQUENCE-CHECK( $a_0, a_1$ )
10:  if  $a_0.\text{seqPhase} = \text{Compute} \vee a_1.\text{seqPhase} = \text{Compute}$  then
11:    execute seqRANK-COMPUTE( $a_0, a_1$ )
12:  else if  $a_0.\text{seqPhase} = \text{Inspect} \vee a_1.\text{seqPhase} = \text{Inspect}$  then
13:    execute seqRANK-INSPECT( $a_0, a_1$ )
14:  else
15:    execute seqRANK-RESET( $a_0, a_1$ )

16: function RANKING( $a_0, a_1$ )
17:  if  $a_0.\text{Rank} = a_1.\text{Rank}$  then
18:     $a_1.\text{Rank} \leftarrow a_0.\text{maxRank} \leftarrow a_1.\text{maxRank} \leftarrow \min(\max(a_0.\text{maxRank}, a_1.\text{maxRank}) +$ 
19:       $1, N)$ 
20:  else
21:     $a_0.\text{maxRank} \leftarrow a_1.\text{maxRank} \leftarrow \max(a_0.\text{maxRank}, a_1.\text{maxRank})$ 

21: if  $a_0.\text{protocolPhase} = pReset \vee a_1.\text{protocolPhase} = pReset$  then
22:   execute PROPAGATE-RESET( $a_0, a_1$ )
23: else
24:   execute RANKING( $a_0, a_1$ )
25:   execute ZERO-CHECK( $a_0, a_1$ )
26:   execute SEQUENCE-CHECK( $a_0, a_1$ )
```
