Brief Announcement:
Introducing Recurrence in Self-Stabilization*

Oday Jubran and Oliver Theel

Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
{jubran,theel}@informatik.uni-oldenburg.de

Abstract. We introduce the notion of recurrence, which denotes how frequent a condition is satisfied in an execution of a system. We use this notion in self-stabilization to address the convergence of a system to a behavior that guarantees a minimum recurrence. We apply this notion to show how the design of distributed mutual exclusion algorithms can be altered, to achieve a high recurrence of granting unique privilege to processes, under various time and space requirements.

Keywords: Self-Stabilization, Recurrence, Performance, Mutual Exclusion.

1 Introduction

Self-stabilization ensures that a system’s desired behavior is eventually obtained and never voluntarily violated regardless of the system’s initial behavior. Self-stabilization requires that once the desired property is satisfied in a configuration, any following configuration has to satisfy the property. With this classical definition of self-stabilization, liveness properties, that do not necessarily need to be satisfied in each configuration, are not covered. Such properties may be the core of the system performance evaluation.

In this work, we introduce the notion of recurrence in self-stabilization, which describes (1) how frequent a global condition is satisfied in an execution, and (2) the convergence time to reach a configuration, from which a minimum recurrence is guaranteed. We apply this notion to show how the design of synchronous distributed mutual exclusion algorithms can be altered to achieve high recurrence of granting unique privilege, under various time and space requirements.

We consider the classical shared memory model: the topology is a connected graph, whose vertices are called processes, and each process has a unique id in \{0, ..., n − 1\}. A distributed algorithm is a set of sub-algorithms, executed by the processes. A configuration \(\gamma\) is a vector of the local states of all processes. An execution is a sequence of configurations \(\gamma_1, \gamma_2, \ldots\) such that for \(i \geq 1\), \(\gamma_{i+1}\) is reachable from \(\gamma_i\) by executing guarded commands. An execution is synchronous iff in each step, all processes with enabled commands execute their actions.

* This work was partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS, http://www.avacs.org/).
2 Recurrence in Self-Stabilization

Recurrence addresses the ratio of configurations, that satisfy some condition (i.e. predicate) over configurations, in an execution. This ratio is a means to cover finite and infinite executions.

**Definition 1 (Recurrence).** Let \( \text{con} \) be a condition over configurations, and let \( \Xi : \gamma_i, \gamma_{i+1}, \ldots \) be an execution. The recurrence of \( \text{con} \) in \( \Xi \), denoted by \( \text{Rec}_{\text{con}}, \Xi \), is the ratio of the configurations satisfying \( \text{con} \) in \( \Xi \).

By Definition 1, considering subsequent configurations satisfying \( \text{con} \): if a condition is not satisfied infinitely often, or if the time between two subsequent satisfactions of the condition increases without bounds, then recurrence approaches 0. If a condition holds in each configuration, then recurrence is 1.

Using recurrence, we define properties over executions as follows: the recurrence of some condition in an execution is equal to \( \Delta \), where \( \Delta \in [0, 1] \). Our concern is to measure the worst case convergence (i.e. stabilization) time to an execution suffix, that satisfies having \( \Delta \) recurrence of some condition \( \text{con} \). We denote such convergence time by \( \Delta_{\text{con}}\text{-Convergence time} \). Recall that in the classical self-stabilization, the desired condition to be satisfied has to hold in each configuration after the system stabilizes. This indicates that the recurrence of the condition is required to be 1.

**Definition 2 (\( \Delta_{\text{con}}\text{-Convergence Time} \)).** Let \( A \) be an algorithm, \( \text{con} \) be a condition, and \( \Delta \in [0, 1] \subset \mathbb{R}^+ \).

- An execution \( \Xi : \gamma_0, \gamma_1, \ldots \) is said to have a \( \Delta_{\text{con}}\text{-Convergence time of } k \text{ steps} \) iff \( k \) is the minimum number, such that \( \Delta = \text{Rec}_{\text{con}, \Xi'} \) for the execution suffix \( \Xi' : \gamma_k, \gamma_{k+1}, \ldots \) of \( \Xi \).
- Algorithm \( A \) is said to have \( \Delta_{\text{con}}\text{-Convergence time of } k \text{ steps} \) iff \( k \) is the maximum \( \Delta_{\text{con}}\text{-Convergence time} \) among all executions.

Considering distributed algorithms that are defined by guarded commands, recurrence can be used to measure the frequency of running particular commands, by analyzing the recurrence of their guards. In the following, we apply recurrence to alter the design of distributed mutual exclusion algorithms.

3 Mutual Exclusion Algorithms

We use [1, Algorithm 3], which is designed for synchronous environments, to design three synchronous mutual exclusion algorithms. Mutual Exclusion, denoted by \( ME \), comprises two properties: (1) Safety: at most one process is privileged in each configuration, and (2) Liveness: each process is privileged infinitely often. We also aim to satisfy that the recurrence of granting unique privilege is 1. We follow a similar approach to [2] to design our algorithms.

[1, Algorithm 3] results in an incrementing system with a finite domain. Each process \( p \) has a variable \( r_p \), that is incremented modulo \( n \), such that in a...
legitimate behavior, (1) the value of \( r_p \) is equal to the value of \( r_q \) for each process \( q \), and (2) the value of \( r_p \) is incremented in each step. We extend this system as follows: (1) In the legitimate behavior, the value of \( r_p \) has to be in \( \{0, ..., n - 1\} \). (2) A process is privileged if and only if the id of \( p \) is equal to \( r_p \). This ensures that the algorithm is self-stabilizing wrt. \( ME \). In addition, since these processes’ ids are \( \{0, ..., n - 1\} \), in each step, exactly one process is privileged. In other words, the recurrence of granting a unique privilege is 1. We write “\( 1_{\text{privileged}} \)” to denote this. Due to limited space, we only show the time and space complexity results of our three algorithms. The algorithms are presented in detail in [3].

Table 1 presents the time and space complexities of the three algorithms. Considering the safety property of \( ME \) (granting unique privilege), Algorithm 2 has the shortest convergence time. However, the convergence time to \( 1_{\text{privileged}} \) is the largest. On the other hand, Algorithm 3 has the largest \( ME \)-Convergence time, but the \( 1_{\text{privileged}} \)-Convergence time is shorter than the one of Algorithm 2. In particular, besides space requirements, the overall convergence time to achieve both \( ME \) and \( 1_{\text{privileged}} \) in Algorithm 3 is less than the time required by Algorithm 2. In other words, Algorithm 3 guarantees to start granting unique privilege, safely, before Algorithm 2 does, and with less space requirements.

### Table 1. Time and Space Complexities of Algorithms 1-3

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( ME )-Convergence Time</th>
<th>( 1_{\text{privileged}} )-Convergence Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \text{diam}(G) - 1 )</td>
<td>( 2\times\text{diam}(G) )</td>
<td>( \text{diam}(G) + n )</td>
</tr>
<tr>
<td>2</td>
<td>( \lfloor \text{diam}(G)/2 \rfloor - 1 )</td>
<td>( &gt; 2\times\text{diam}(G) )</td>
<td>( 2\times\text{diam}(G) + n )</td>
</tr>
<tr>
<td>3</td>
<td>( 2\times\text{diam}(G) - 1 )</td>
<td>( 2\times\text{diam}(G) )</td>
<td>( n + 1 )</td>
</tr>
</tbody>
</table>

4 Optimality of \( \left\lceil \frac{\text{diam}(G)}{2} \right\rceil - 1 \) \( ME \)-Convergence Time

As a particular interest, the convergence time wrt. \( ME \), observed in Algorithm 2, slightly improves the state-of-the-art \( \lfloor \text{diam}(G)/2 \rfloor \), introduced in [2]. In addition, this result rectifies [2, Theorem 4] that \( \lfloor \text{diam}(G)/2 \rfloor \) is a lower bound; in the proof of [2, Theorem 4], the issue, that the privilege condition of a process \( p \) may also cover the states of the neighbors of \( p \), is missing. Considering this point, while following the steps of the proof of [2, Theorem 4], we conclude that \( \left\lceil \frac{\text{diam}(G)}{2} \right\rceil - 1 \) is optimal for synchronous executions. Details are found in [3].

References