# Schlegel Diagram and Optimizable Immediate Snapshot Protocol*

Susumu Nishimura
Dept. of Mathematics, Graduate School of Science, Kyoto University
susumu@math.kyoto-u.ac.jp

**Abstract**

In the topological study of distributed systems, the immediate snapshot is the fundamental computation block for the topological characterization of wait-free solvable tasks. However, in reality, the immediate snapshot is not available as a native built-in operation on shared memory distributed systems. Borowsky and Gafni have proposed a wait-free multi-round protocol that implements the immediate snapshot using more primitive operations, namely the atomic reads and writes.

In this paper, up to an appropriate reformulation on the original protocol by Borowsky and Gafni, we establish a tight link between each round of the protocol and a topological operation of subdivision using Schlegel diagram. Due to the fact shown by Kozlov that the standard chromatic subdivision is obtained by iterated subdivision using Schlegel diagram, the reformulated version is proven to compute the immediate snapshot in a topologically smoother way. We also show that the reformulated protocol is amenable to optimization: Since each round restricts the possible candidates of output to an iteratively smaller region of finer subdivision, each process executing the protocol can decide at an earlier round, beyond which the same final output is reached no matter how the remaining rounds are executed. This reduces the number of read and write operations involved in the overall execution of the protocol, relieving the bottleneck of access to shared memory.

## 1 Introduction

The snapshot models [1, 4, 6, 11] for the shared memory distributed system have been intensively studied for the analysis of solvability of distributed tasks in the wait-free (or more generalized failure) models. In particular, the immediate snapshot model [1, 4] and the iterated immediate snapshot mode [6] are central to the study. The (iterated) immediate snapshot protocol is modeled by a topological operation on simplicial complexes, namely, the (iterated) standard chromatic subdivision. This topological interpretation has boosted theoretical investigations on distributed systems using simplicial complexes. Most notably, Herlihy and Shavit have established the asynchronous computability theorem [13], which states that a distributed task is wait-free solvable in the asynchronous read-write shared memory model if and only if the task is expressed by a suitable pair of an iterated standard chromatic subdivision and a simplicial map.

Although immediate snapshots are not natively supported in real distributed systems, the protocol proposed by Borowsky and Gafni [5] provides a wait-free implementation of it on asynchronous shared memory systems with atomic reads and writes. Their protocol is a multi-round protocol, where each individual process in a distributed system consisting of $n+1$ processes decides its snapshot only if it witnesses, for $k$-th round, $n+2-k$ different processes (including the process itself) that have written to shared memory.

In the present paper, we give yet another multi-round protocol for the immediate snapshot, reformulating the one by Borowsky and Gafni. Though both protocols work equally, the reformulated version has notable advantages over the original one.

*This work is to appear in *OPODIS 2017 — The 21st International Conference on Principles of Distributed Systems.*

1. The reformulated multi-round protocol induces a neat correspondence of each individual round with a topological construction, namely a subdivision using Schlegel diagram.

   Benavides and Rajsbaum [2] showed that each round of the protocol by Borowsky and Gafni does not simply subdivides the input complex but it produces an intermediate protocol complex that is not a (pseudo)manifold. They make use of 'collapsing' to describe how the protocol complex is transformed topologically at each protocol round, concluding that the standard chromatic subdivision is obtained as the final result. In contrast, the present paper shows that, up to reformulation, each protocol round exactly corresponds to a subdivision using Schlegel diagram. This series of corresponding subdivisions gives rise to the standard chromatic subdivision, due to a straightforward topological argument by Kozlov [17].

2. Due to the simpler topological structure, the reformulated version is amenable to mechanical optimization.

   In shared memory systems, shared memory access is a major bottleneck. Processes share a single shared memory module, which serializes simultaneous requests to handle them one at a time. The above mentioned multi-round protocols for the immediate snapshot involve read and write requests multiplied by the number of rounds to be performed (and also by the number of processes). The multiplied requests to the shared memory thus can cause performance degradation due to memory contention.

   For each concrete implementation of a protocol that makes use of the immediate snapshot, the reformulated version of the immediate snapshot protocol can be optimized to issue a lesser number of memory requests so that each process decides its output value at an earlier round, beyond which any execution path converges to a single unique output. This refinement is possible because the reformulated protocol iteratively subdivides the protocol complex at each round, narrowing down the candidates of output to those in a smaller region of finer subdivision. Thus the output can be decided as soon as the possible outputs have been narrowed down to a singleton set. This optimizes the protocol to perform a lesser number of shared memory access, where the optimized code can be mechanically derived for each concrete instance.

   With this optimization technique, the author believes that the immediate snapshot model, which has been studied mostly of theoretical concern, can also serve as a fundamental construct for wait-free distributed programming in a more practical context.

**Related Work.** In [17], Kozlov has proven that the standard chromatic subdivision is indeed a subdivision by showing the standard chromatic subdivision is obtained by the iterated subdivision using Schlegel diagram. He also argued that the transient complexes that appear in the intermediate steps of iterated subdivision can be given computational interpretation, but his interpretation combines atomic writes with immediate scan operations. In contrast, the present paper gives the computational interpretation solely with atomic reads and writes, establishing the exact correspondence of each round of the reformulated immediate snapshot protocol with subdivision using Schlegel diagram.

Benavides and Rajsbaum [2] studied the topological structure induced by the multi-round immediate snapshot protocol of Borowsky and Gafni. They observed that the series of shared memory reads and writes involved in each single round of the protocol generates a protocol complex that augments the standard chromatic subdivision with extra simplexes. Due to those extras, the protocol complexes are neither a subdivision of the input complex nor a (pseudo)manifold in general. They analyzed the topological structure of the protocol complexes in detail and presented a topological model, in which subsequent rounds of the protocol collapse those extra simplexes to end up with the standard chromatic subdivision. This collapsing series of complexes, though topologically insightful, does not explain well the correspondence to the underlying operational (execution) model. Assuming the colored simplicial topological model, where every simplex is a collection of vertexes of distinct colors (process ids), there can be no operational counterpart to collapsing of a simplex: Since a simplex is always collapsed to a degenerate simplex of strictly smaller dimension (i.e., of fewer colors), it absurdly indicates that such an operation would shrink a process group into a strictly smaller one (even if no process in the group crashed). In the present paper, we reformulate the multi-round protocol by Borowsky and Gafni so that each round is precisely a subdivision using Schlegel diagram, providing a simpler topological model. The neat correspondence between the topological model and

the operational model also allows us to optimize the protocol for reduced shared memory access. (See Section 4.)

Hoest and Shavit [14] proposed the nonuniform iterated immediate snapshot model, a refinement of the immediate snapshot model, for the purpose of a precise analysis of protocol complexity. The nonuniform iterated immediate snapshot topologically corresponds to the iterated nonuniform chromatic subdivision, which generalizes the iterated standard chromatic subdivision so that individual simplexes are allowed to be subdivided different numbers of times. Their nonuniform model may also be applied to protocol optimization. That is, a task solvable by a protocol in the uniform model would be substituted by a protocol in the nonuniform model that solves the same task with a coarser iterated nonuniform subdivision. However, it seems nontrivial in general to find an optimal nonuniform protocol. The reformulated multi-round protocol in the present paper, due to the smooth correspondence between the topological and operational models, allows mechanical optimization on any protocol given in the uniform model.

**Outline.** The rest of the paper is organized as follows. Section 2 describes the shared memory distributed computing model and the wait-free multi-round protocol by Borowsky and Gafni and reviews the topological theory concerning wait-free solvability of distributed tasks. Section 3 proposes a reformulation of the multi-round protocol by Borowsky and Gafni. We prove the reformulated protocol also computes the immediate snapshot, showing that each round of the reformulated protocol precisely corresponds to a subdivision using Schlegel diagram. In Section 4, we further argue that the reformulated protocol is amenable to optimization. We show that, for each concrete protocol that solves a task using the iterated immediate snapshot, the protocol can be optimized to decide the final output at an earlier round, as soon as the collection of possible outputs to be reached is narrowed down to a singleton set. Finally, Section 5 concludes the paper.

# 2 Distributed Computing and Topological Model

Throughout the paper we consider a distributed system of $n+1$ faulty processes, which have process ids numbered 0 through $n$. We assume the asynchronous read-write shared memory model, where the distributed system has a shared memory consisting of single-writer, multi-reader atomic registers and processes can communicate solely by atomic reads and writes on these registers. We further assume that each process $i$ receives its initial private input value through the variable $v_i$, which is local to the process.

The rest of this section is devoted to give an overview of the immediate snapshot model and the basics of combinatorial topology related to the topological theory of wait-free solvability of distributed tasks. For a more complete exposition on the subject, see [11, 16].

## 2.1 Immediate snapshot in the read-write model

Borowsky and Gafni proposed the wait-free implementation of the immediate snapshot in the read-write shared memory model [5]. Algorithm 1 gives the Borowsky-Gafni protocol in a recursive style implementation by Gafni and Rajsbaum [10]. The protocol is a multi-round protocol, where the series of recursive calls $\mathsf{IS}(n)$, $\mathsf{IS}(n-1)$, $\ldots$, $\mathsf{IS}(0)$ correspond to $n+1$ multiple rounds and the processes communicate through a series of $n+1$ shared memory arrays $mem_d$ ($n \geq d \geq 0$), each of which consists of $n+1$ registers indexed 0 through $n$. Each process $i$, per each recursive call $\mathsf{IS}(d)$ for the $(n+1-d)$-th round, invokes the *write&scan* operation WScan on the shared memory array $mem_d$. When WScan is invoked by process $i$, it first writes the private value $v_i$ of the process $i$ to the register $mem_d[i]$, then scans the view, i.e., the set of values that have been written in the array $mem_d$, and returns the view paired with the process id. The view is gathered by collect, which reads the registers of the array in an unspecified order. If process $i$ witnesses $d+1$ values in the view, it returns the pair of process id and the view as the result of immediate snapshot, terminating recursion; Otherwise, it continues recursive call $\mathsf{IS}(d-1)$ for another round.

We notice that the view returned by WScan per each round is simply discarded, when not sufficiently many values are witnessed.

**Algorithm 1** Multi-round write&scan code for the process $i$

**procedure** $\mathsf{WScan}(d)$
  $mem_d[i] \leftarrow v_i$
  $view \leftarrow \mathsf{collect}(mem_d)$
  **return** $(i, view)$

**procedure** $\mathsf{IS}(d)$
  $(i, view) \leftarrow \mathsf{WScan}(d)$
  **if** $|view| = d + 1$ **then return** $(i, view)$
  **else** $\mathsf{IS}(d-1)$

## 2.2 Simplicial complexes and subdivisions

Let $V$ be a set of vertexes. A *simplex* $\sigma$ is a finite subset of $V$. The dimension of $\sigma$, denoted by $\dim(\sigma)$, is given by $|\sigma| - 1$. A simplex $\sigma$ of dimension $d$ is called a *$d$-simplex*. In particular, the empty simplex $\emptyset$ is a $(-1)$-simplex. A simplex $\sigma$ is called a *face* of $\tau$ if $\sigma \subseteq \tau$ and is particularly called a *proper* face if the inclusion is strict.

A *simplicial complex* (or a *complex* for short) $\mathcal{C}$ is a set of simplexes such that $\sigma \in \mathcal{C}$ and $\tau \subseteq \sigma$ implies $\tau \in \mathcal{C}$. The dimension of $\mathcal{C}$, written $\dim(\mathcal{C})$, is the maximum dimension of simplexes contained in $\mathcal{C}$. A complex of dimension $d$ is also called a *$d$-complex*. We write $V(\mathcal{C})$ for the set of vertexes contained in $\mathcal{C}$. A complex $\mathcal{D}$ is called a *subcomplex* of $\mathcal{C}$, if $\mathcal{D} \subseteq \mathcal{C}$. The *$k$-skeleton* of a complex $\mathcal{C}$, written $\mathrm{skel}^{(k)}\,\mathcal{C}$, is the maximum subcomplex of $\mathcal{C}$ of dimension $k$ or less, namely, $\mathrm{skel}^{(k)}\,\mathcal{C} = \{\sigma \in \mathcal{C} \mid \dim(\sigma) \leq k\}$.

A *facet* $\sigma$ of $\mathcal{C}$ is a maximal simplex, i.e., $\sigma$ is not a proper face of any $\tau \in \mathcal{C}$. We write $\overline{\sigma}$ for a complex whose sole facet is the simplex $\sigma$, i.e., $\overline{\sigma} = \{\tau \mid \tau \subseteq \sigma\}$. A complex is called *pure*, if all its facets have the same dimension.

Whenever $\sigma \cap \tau = \emptyset$, we write $\sigma * \tau$ for the *join* of the simplexes $\sigma$ and $\tau$, namely the union $\sigma \cup \tau$. Likewise, whenever $V(\mathcal{C}) \cap V(\mathcal{D}) = \emptyset$, we define the *join* of $\mathcal{C}$ and $\mathcal{D}$ by $\mathcal{C} * \mathcal{D} = \{\sigma * \tau \mid \sigma \in \mathcal{C}, \tau \in \mathcal{D}\}$. The *star* of a vertex $v \in \mathcal{C}$ is defined by $\mathrm{St}(v, \mathcal{C}) = \{\tau \in \mathcal{C} \mid \{v\} \cup \tau \in \mathcal{C}\}$, which is the maximum subcomplex of $\mathcal{C}$ whose every facet contains $v$.

Throughout the paper, complexes are assumed to be pure. Also, we solely consider the so-called *chromatic* simplexes and complexes. Suppose we have a *coloring function* $\mathrm{color} : V \to \{0, \ldots, n\}$. A simplex $\sigma$ is chromatic if $\mathrm{color}(v) = \mathrm{color}(v')$ implies $v = v'$ for every $v, v' \in \sigma$; A complex $\mathcal{C}$ is chromatic if every simplex $\sigma \in \mathcal{C}$ is chromatic.

A *simplicial map* is a total vertex map $\mu : V(\mathcal{C}) \to V(\mathcal{D})$ such that $\mu(\sigma) \in \mathcal{D}$ for every $\sigma \in \mathcal{C}$. Every simplicial map must be *color-preserving*, i.e., $\mathrm{color}(v) = \mathrm{color}(\mu(v))$ for every $v \in V(\mathcal{C})$.

A *subdivision* of a complex $\mathcal{C}$, written $\mathrm{Div}\,\mathcal{C}$, is a finer complex obtained by dividing each simplex of the complex into smaller pieces of (chromatic) simplexes. (In this paper, though we occasionally refer to geometric presentation of subdivisions, we are solely concerned with combinatorial definition of subdivision in formality. For example, $\overline{\sigma}$ should be understood as the trivial subdivision, which does not geometrically refine $\sigma$ at all.) For a simplex $\tau \in \mathrm{Div}\,\mathcal{C}$, we write $\mathrm{Carr}(\tau, \mathcal{C})$ for the *carrier* of $\tau$, namely, the smallest simplex $\sigma \in \mathcal{C}$ such that $\tau \in \mathrm{Div}\,\sigma$.

A subdivision induces a *parent map* $\pi$, which carries each vertex of the subdivision to the unique vertex $\pi(v)$ of matching color in its carrier. That is, for $v \in V(\mathrm{Div}\,\mathcal{C})$, $\pi(v)$ is the unique vertex of $\mathcal{C}$ such that $\pi(v) \in \mathrm{Carr}(\{v\}, \mathcal{C})$ and $\mathrm{color}(\pi(v)) = \mathrm{color}(v)$. The parent map $\pi : V(\mathrm{Div}\,\mathcal{C}) \to V(\mathcal{C})$ is a color-preserving simplicial map.

## 2.3 Distributed task and asynchronous computability

A (colored) *task* for a distributed system with $n+1$ faulty processes is defined by a triple $(\mathcal{I}, \mathcal{O}, \Phi)$, where $\mathcal{I}$ is an input complex (of dimension $n$), $\mathcal{O}$ is an output complex (of dimension $n$), and $\Phi : \mathcal{I} \to 2^{\mathcal{O}}$ is a *carrier map*, a monotonic function that maps every input simplex $\sigma \in \mathcal{I}$ to an output subcomplex $\Phi(\sigma) \subseteq \mathcal{O}$ such that $\mathrm{color}(\sigma) = \bigcup\{\mathrm{color}(\tau) \mid \tau \in \Phi(\sigma)\}$.

The asynchronous computability theorem [13] gives a topological characterization for the class of tasks that are wait-free solvable in the shared memory read-write model. The primary topological tool employed in the theorem is subdivision on simplicial complexes, especially the *standard chromatic subdivision*. See Figure 1 for geometric presentation of the standard chromatic subdivision on 1-simplex

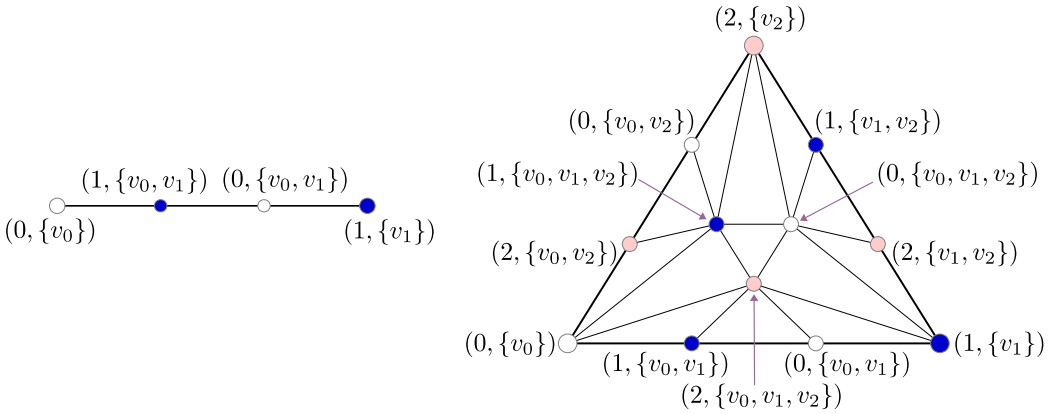


Figure 1: The standard chromatic subdivisions on 1-simplex and 2-simplex

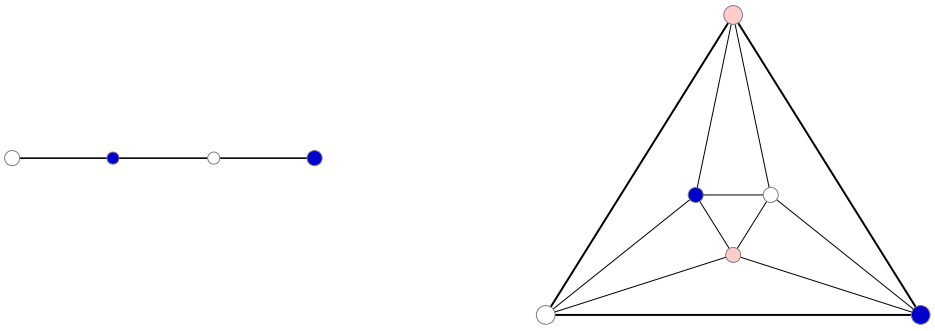

Figure 2: Schlegel diagrams for 1-simplex and 2-simplex

$\{v_0, v_1\}$ and 2-simplex $\{v_0, v_1, v_2\}$: The standard chromatic subdivision refines each $d$-simplex $\sigma$ ($d > 0$) by introducing new $d + 1$ vertexes (of different colors) at antiprismatic positions displaced from the barycenter of the simplex. Combinatorially, as indicated in Figure 1, each vertex of color $i$ contained in the subdivision is designated by a pair $(i, \sigma)$, where $\sigma$ is a nonempty subset of the original simplex and $i \in \mathrm{color}(\sigma)$. (Herein and after we will depict the vertexes for different processes with colors white, dark blue, and light red to indicate processes of id 0, 1, and 2, respectively.) Furthermore, a set $\{(i_0, \tau_0), \ldots, (i_d, \tau_d)\}$ of vertexes of distinct colors forms a $d$-simplex of the subdivision if and only if the subsets of the original simplex are linearly ordered (as $\tau_0 \subseteq \cdots \subseteq \tau_d$, up to appropriate permutation).

In what follows, let us write $\mathrm{Ch}\,\mathcal{C}$ for the standard chromatic subdivision applied to every simplex in $\mathcal{C}$ and also write $\mathrm{Ch}^m\mathcal{C}$ ($m \geq 0$) for the $m$-iterated application of Ch on $\mathcal{C}$.

The multi-round protocol for the immediate snapshot (Algorithm 1) in effect implements the standard chromatic subdivision [2]: Every non-faulty process $i$ executing the protocol returns a vertex $(i, \tau)$ of the subdivision.

**Theorem 2.1** (Theorem 5.29 and Corollary 5.31 of [13])**.** A colored task $(\mathcal{I}, \mathcal{O}, \Phi)$ has a wait-free protocol in the asynchronous shared memory read-write model if and only if there exists a subdivision $\mathrm{Div}\,\mathcal{I}$ and a *decision map* $\delta$, which is a color-preserving simplicial map $\delta : V(\mathrm{Div}\,\mathcal{I}) \to V(\mathcal{O})$ such that $\delta(\sigma) \in \Phi(\mathrm{Carr}(\sigma, \mathcal{I}))$ for every $\sigma \in \mathrm{Div}\,\mathcal{I}$.

In particular, the subdivision Div can be taken $\mathrm{Ch}^K$ for some $K \geq 0$.

# 3 Immediate Snapshot as Iterated Subdivision Using Schlegel Diagram

This section presents a multi-round protocol for the immediate snapshot, by reformulating the protocol by Borowsky and Gafni [5]. We show that each round of the protocol corresponds exactly to a subdivision using Schlegel diagram.

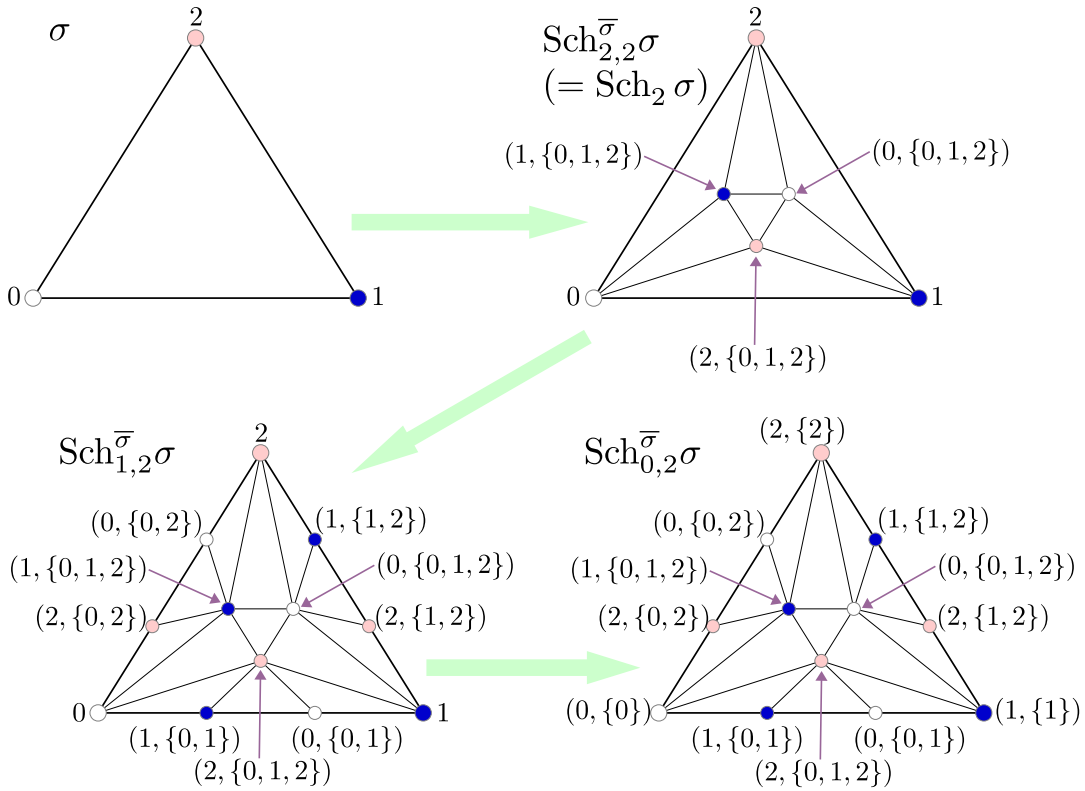


Figure 3: Standard chromatic subdivision on $\sigma = \{0, 1, 2\}$ using Schlegel diagrams

### 3.1 Schlegel diagram and subdivision

A *Schlegel diagram* is a projection of a polytope onto one of its facets [19]. In the present paper, we are solely concerned with Schlegel diagrams on cross-polytopes. The Schlegel diagram on $(d+1)$-dimensional cross-polytope, which consists of $2(d+1)$ vertexes, gives a subdivision of $d$-simplex. Figure 2 shows subdivisions of 1-simplex and 2-simplex by Schlegel diagrams, where the former is derived from the quadrilateral and the latter from the octahedron. (Note that the standard chromatic subdivision is a refinement of Schlegel diagram in general, but they coincide for 1-simplexes.)

The complex of Schlegel diagram that subdivides a $d$-simplex $\sigma = \{v_0, \ldots, v_d\}$, denoted by $\mathrm{Sch}_d\,\sigma$, is formally defined as follows:

$$\mathrm{Sch}_d\,\sigma = \bigcup \left\{ \overline{\{v_i \mid i \in \sigma \setminus I\} * \{(i,\sigma) \mid i \in I\}} \mid \emptyset \subsetneq I \subseteq \{0, \ldots, d\} \right\},$$

where each $(i, \sigma)$ is a new vertex introduced for subdivision, with coloring $\mathrm{color}((i,\sigma)) = i$. Geometrically, the Schlegel diagram subdivides a $d$-simplex $\sigma$ into smaller facets, namely, the central facet $\{(0,\sigma), \ldots, (d,\sigma)\}$, which is solely comprised of the new vertexes, and other facets, each of which shares a lower dimensional simplex with the central facet. (See Figure 3 for the subdivision of 2-simplex.) Notice that $\sigma$ is no more a face of Schlegel subdivision $\mathrm{Sch}_d\,\sigma$, as it does not share any simplex with the central facet. When $\dim(\sigma) \neq d$, we define $\mathrm{Sch}_d\,\sigma$ by a trivial subdivision, i.e., $\mathrm{Sch}_d\,\sigma = \overline{\sigma}$.

Kozlov [17] has shown that the standard chromatic subdivision can be obtained by a series of Schlegel diagram that subdivides simplexes in the order of decreasing dimension. To put it formal, let $\mathcal{D}$ be any subdivision of a $d$-complex $\mathcal{C}$ such that $\mathcal{C} \cap \mathcal{D} = \mathrm{skel}^{(k)}\,\mathcal{C}$, meaning that $\mathcal{D}$ subdivides simplexes of $\mathcal{C}$ up to dimension $k+1$ and higher but no simplexes of lower dimension. Let us write $\mathrm{Sch}^{\mathcal{C}}_k \mathcal{D}$ for the subdivision of $\mathcal{D}$ applied to every $k$-simplex of $\mathcal{C} \cap \mathcal{D}$ by Schlegel diagram, namely,

$$\mathrm{Sch}^{\mathcal{C}}_k \mathcal{D} = \{\tau * \sigma' \mid \tau * \sigma \in \mathcal{D} \text{ and } \sigma' \in \mathrm{Sch}_k\,\sigma \text{ for some } \sigma \in \mathcal{C} \cap \mathcal{D}\}.$$

Let us also write $\mathrm{Sch}^{\mathcal{C}}_{h,j}$ for the composition $\mathrm{Sch}^{\mathcal{C}}_h \circ \mathrm{Sch}^{\mathcal{C}}_{h+1} \circ \cdots \circ \mathrm{Sch}^{\mathcal{C}}_j$ $(0 \leq h, j \leq d)$ of subdivisions on simplexes in the order of decreasing dimension. (When $h > j$, $\mathrm{Sch}^{\mathcal{C}}_{h,j}$ denotes the trivial subdivision.)

**Algorithm 2** Multi-round write&oblivious scan code for the process $i$

**procedure** $\mathsf{WOScan}(d)$
  $mem_d[i] \leftarrow v_i$
  $view \leftarrow \mathsf{collect}(mem_d)$
  **if** $|view| = d+1$ **then return** $(i, view)$
  **else return** $v_i$

**procedure** $\mathsf{IS'}(d)$
  $u \leftarrow \mathsf{WOScan}(d)$
  **if** $u \neq v_i$ **then return** $u$
  **else** $\mathsf{IS'}(d-1)$

**Theorem 3.1** ([17])**.** For any pure complex $\mathcal{C}$ of dimension $d$, $\mathrm{Ch}\,\mathcal{C} = \mathrm{Sch}^{\mathcal{C}}_{0,d}\,\mathcal{C}$.

Figure 3 shows how the standard chromatic subdivision on a 2-simplex $\sigma = \{0,1,2\}$ with $\mathrm{color}(i) = i$ for every $i \in \{0,1,2\}$ is obtained by the series of subdivisions using Schlegel diagram. For example, the 2-simplex $\{0,1\} * \{(2,\{0,1,2\})\}$ in $\mathrm{Sch}^{\overline{\sigma}}_{2,2}$ is further refined in the next step of subdivision, say, the 1-simplex $\{0,1\}$ is subdivided into three parts $\{0,(1,\{0,1\})\}$, $\{(0,\{0,1\}),(1,\{0,1\})\}$, $\{(0,\{0,1\}),1\}$, which are each *joined* with $\{(2,\{0,1,2\})\}$.

## 3.2 The immediate snapshot protocol with oblivious scan

Algorithm 2 gives a multi-round immediate snapshot protocol, which reformulates Algorithm 1 in Section 2.1. It is easy to see that they are indeed equivalent protocols in different presentations. Remember that in Algorithm 1 the view information collected at each particular round is discarded unless the view witnesses the expected number of writes. Algorithm 2 just makes this explicit by employing the *write&oblivious scan* operation $\mathsf{WOScan}$ on shared memory array, in place of write&scan operation. When process $i$ calls $\mathsf{WOScan}(d)$ and the view does not witness $d$ writes, $\mathsf{WOScan}(d)$ returns $v_i$, discarding the view collected at the scan phase.

With this reformulation, we can prove that the protocol computes the standard chromatic subdivision, with an exact correspondence of a write&oblivious scan operation at a particular round with Schlegel diagram.

**Lemma 3.2.** Suppose $\mathsf{WOScan}(d)$ has ever been called by $d+1$ distinct processes. If $\sigma = \{v_0,\dots,v_d\}$ is the collection of private values that have been assigned to the $d+1$ processes and $\tau$ is the set of results returned by non-faulty processes, then $\tau \in \mathrm{Sch}_d\,\sigma$.

*Proof.* When $d+1$ processes invoked $\mathsf{WOScan}(d)$ and none of them were faulty, at least one process witnesses the writes by all the $d+1$ processes in its view. This means $\tau \setminus \sigma \neq \emptyset$ and thus $\tau \in \mathrm{Sch}_d\,\sigma$. When some of the processes were faulty, $\dim(\tau) < d$ and $\tau \in \mathrm{Sch}_d\,\sigma$. □

**Lemma 3.3.** Consider an execution of the protocol $\mathsf{IS'}(n)$ by $n+1$ processes, in which each process $i$ has started with its own initial private value $v_i$ and has either successfully returned a result or crashed. Let $\sigma_d$ $(n \geq d \geq 0)$ denote the set of results that have been successfully returned by a call $\mathsf{WOScan}(d)$ by some process in the execution. Also, let $\tau_d$ $(n \geq d \geq 0)$ denote the set of values that have been returned by a call $\mathsf{IS'}(k)$ for some $k$ greater than $d-1$. We define $\sigma_{n+1} = \{v_0,\dots,v_n\}$ and $\tau_{n+1} = \emptyset$.

Then the following properties hold for every $d$ $(n+1 \geq d \geq 0)$.

(i) $\dim(\sigma_d \cap \sigma_{n+1}) < d$;

(ii) $\tau_d = \tau_{d+1} * (\sigma_d \setminus \sigma_{n+1})$ and $\tau_d \cap \sigma_{n+1} = \emptyset$;

(iii) $\tau_{d+1} * \sigma_d \in \mathrm{Sch}^{\overline{\sigma_{n+1}}}_{d,n}\sigma_{n+1}$.

*Proof.* The property (i) follows from lemma 3.2 by induction on $d$. The property (ii) immediately follows from the definition by an inductive argument.

Let us show (iii) by induction on $d$. For the base case $d = n$, since $\sigma_n \in \mathrm{Sch}_n\sigma_{n+1}$ by lemma 3.2, we have $\tau_{n+1} * \sigma_n = \sigma_n \in \mathrm{Sch}^{\overline{\sigma_{n+1}}}_{n,n}\sigma_{n+1}$. For the inductive step, assume $\tau_{d+2} * \sigma_{d+1} \in \mathrm{Sch}^{\overline{\sigma_{n+1}}}_{d+1,n}\sigma_{n+1}$. By

lemma 3.2 we have $\sigma_d \in \mathrm{Sch}_d(\sigma_{d+1} \cap \sigma_{n+1})$. Since $\sigma_{d+1} \cap \sigma_{n+1} \in \overline{\sigma_{n+1}}$, we have $\tau_{d+1} * (\sigma_{d+1} \cap \sigma_{n+1}) = \tau_{d+2} * (\sigma_{d+1} \setminus \sigma_{n+1}) * (\sigma_{d+1} \cap \sigma_{n+1}) = \tau_{d+2} * \sigma_{d+1} \in \mathrm{Sch}_{d+1,n}^{\overline{\sigma_{n+1}}} \sigma_{n+1}$ by property (ii) and the induction hypothesis. Hence $\tau_{d+1} * \sigma_d \in \mathrm{Sch}_d^{\overline{\sigma_{n+1}}}(\mathrm{Sch}_{d+1,n}^{\overline{\sigma_{n+1}}} \sigma_{n+1}) = \mathrm{Sch}_{d,n}^{\overline{\sigma_{n+1}}} \sigma_{n+1}$. □

**Theorem 3.4.** Suppose $n+1$ processes executed the protocol $\mathsf{IS}'(n)$ with the set $\sigma = \{v_0, \ldots, v_n\}$ of initial private inputs. If $\tau$ is the set of results returned by non-faulty processes, $\tau \in \mathrm{Ch}\,\sigma$.

*Proof.* Let $\sigma_d$'s and $\tau_d$'s denote the sets as defined in lemma 3.3. Then, $\sigma = \sigma_{n+1}$ and $\tau = \tau_1 * \sigma_0$. By lemma 3.3(iii) and theorem 3.1, we have $\tau = \tau_1 * \sigma_0 \in \mathrm{Sch}_{0,n}^{\overline{\sigma}} \sigma = \mathrm{Ch}\,\sigma$. □

# 4 The Generic Protocol for Solving Tasks and Its Optimization

This section gives a generic protocol for solving a task on read-write shared memory distributed system, using the immediate snapshot protocol presented in the previous section, and discusses how, for each concrete instance, the generic protocol can be optimized to reduce shared memory access.

## 4.1 A generic protocol via iterated immediate snapshot

By the asynchronous computability theorem, for any wait-free solvable task, we have a protocol $(\mathcal{I}, \mathcal{O}, \delta \circ \mathrm{Ch}^K)$ that implements the task, where the carrier map is given by a pair of the full-information protocol of the $K$-iterated standard chromatic subdivision $\mathrm{Ch}^K$ (by means of the iterated use of the multi-round immediate snapshot protocol) and a decision map $\delta : V(\mathrm{Ch}^K) \to V(\mathcal{O})$. Without loss of generality, we may assume $K \geq 1$.

**Algorithm 3** The generic code for the process $i$, using iterated immediate snapshot

```
procedure WOScan(k, d)
    mem_{k,d}[i] ← v_i
    view ← collect(mem_{k,d})
    if |view| = d + 1 then return (i, view)
    else return v_i
procedure IIS(k, d)
    u ← WOScan(k, d)
    if u ≠ v_i then
        if k = K then return δ(u)
        else v_i ← u; IIS(k + 1, n)
    else IIS(k, d − 1)
```

Algorithm 3 gives the code that implements the protocol $(\mathcal{I}, \mathcal{O}, \delta \circ \mathrm{Ch}^K)$ in the generic form. Each process $i$ initiates the protocol execution by invoking $\mathsf{IIS}(1, n)$, where its initial private input is passed through the variable $v_i$. Throughout the entire protocol execution, each process goes through a series of shared memory arrays $mem_{k,d}$ ($1 \leq k \leq K$, $n \geq d \geq 0$). For each recursive call $\mathsf{IIS}(k, d)$, process $i$ computes the $(n - d + 1)$-th round of the $k$-th iteration of the multi-round immediate snapshot protocol, by invoking the write&oblivious scan $\mathsf{WOScan}(k, d)$ on the array $mem_{k,d}$. Each round corresponds to a single step of subdivision on $d$-simplexes using Schlegel diagram, for the $k$-th iteration of standard chromatic subdivision. When the multi-round execution by process $i$ finishes the last iteration of chromatic subdivision (i.e., $k = K$), the protocol returns $\delta(u)$ as the output, where $u$ is a vertex of the $K$-iterated standard chromatic subdivision.

The following is a corollary to Theorem 3.4.

**Theorem 4.1.** Let $(\mathcal{I}, \mathcal{O}, \delta \circ \mathrm{Ch}^K)$ be a protocol for $n+1$ processes that solves a task. Suppose each process $i$ starts with a private input value $v_i$ such that $\sigma = \{v_0, \ldots, v_n\} \in \mathcal{I}$ and executes the protocol by invoking $\mathsf{IIS}(1, n)$. If $\tau$ is the set of results returned by non-faulty processes, $\tau \in \delta(\mathrm{Ch}^K \sigma)$.

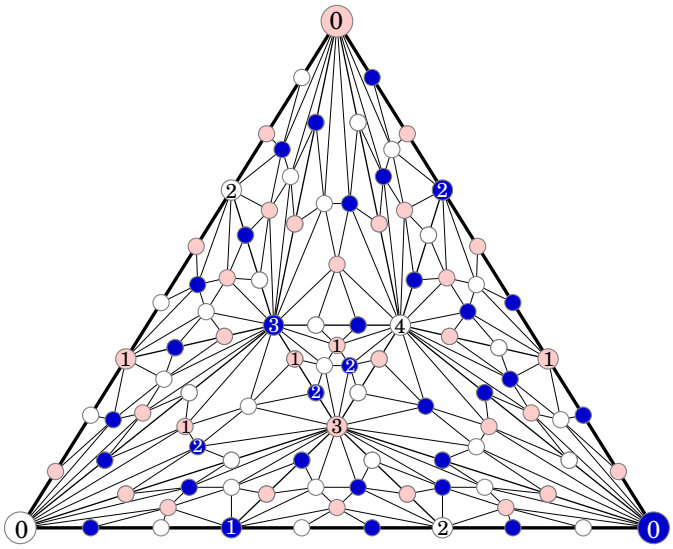

Figure 4: Decision map $\delta$ for renaming

## 4.2 Protocol optimization for reduced memory access

Algorithm 3 gives a generic protocol, but a concrete instance of it often contains redundant memory access. Below we argue that each instance of the generic protocol can be mechanically optimized to skip the redundant access, by applying the technique of program specialization. (Program specialization is a source-level program optimization technique, also known as *partial evaluation* [15]. See Appendix A for a brief overview.)

To see how the protocol is optimized, let us consider a particular instance of the generic protocol that solves a renaming task [5] for 3 processes,[1] where the 3 processes, starting with initial assignment $v_0 = 0$, $v_1 = 1$, $v_2 = 2$, respectively, decide on different names taken from $\{0, 1, 2, 3, 4\}$. The protocol is given by $(\mathcal{I}, \mathcal{O}, \delta \circ \mathrm{Ch}^2)$, where $\mathcal{I} = \overline{\{0, 1, 2\}}$ is the input complex, $\mathcal{O} = \{\tau \mid \tau \subseteq \{0, 1, 2, 3, 4\}, \dim(\tau) \leq 2\}$ is the output complex of differently renamed vertexes, and $\delta : V(\mathrm{Ch}^2 \mathcal{I}) \to V(\mathcal{O})$ is the decision map defined with the corresponding parent map $\pi_{2,1} : V(\mathrm{Ch}^2 \mathcal{I}) \to V(\mathrm{Ch}\, \mathcal{I})$ as given below:

$$\delta(w) = \begin{cases} 4 & \text{if } \pi_{2,1}(w) = (0, \{0, 1, 2\}), \\ 3 & \text{if } \mathrm{Carr}(w, \mathrm{Ch}\, \mathcal{I}) \not\supseteq \{(1, \{0, 1, 2\}), (2, \{0, 1, 2\})\} \\ & \text{and } \pi_{2,1}(w) = (i, \{0, 1, 2\}) \text{ for some } i \in \{1, 2\}, \\ 2 & \text{if either } \mathrm{color}(w) = 1 \text{ and } \mathrm{Carr}(w, \mathrm{Ch}\, \mathcal{I}) \supseteq \{(1, \{0, 1, 2\}), (2, \{0, 1, 2\})\} \\ & \text{or } \pi_{2,1}(w) = (i, \{i, j\}) \text{ for some } i, j \in \{0, 1, 2\} \text{ such that } i < j, \\ 1 & \text{if either } \mathrm{color}(w) = 2 \text{ and } \mathrm{Carr}(w, \mathrm{Ch}\, \mathcal{I}) \supseteq \{(1, \{0, 1, 2\}), (2, \{0, 1, 2\})\} \\ & \text{or } \pi_{2,1}(w) = (i, \{i, j\}) \text{ for some } i, j \in \{0, 1, 2\} \text{ such that } i > j, \\ 0 & \text{if } \pi_{2,1}(w) = (i, \{i\}) \text{ for some } i \in \{0, 1, 2\}. \end{cases}$$

In the definition above, it is assumed that each vertex is appropriately colored according to the context. In particular, as every simplex $\tau \in \mathcal{O}$ of renamed processes is colored, $V(\mathcal{O})$ comprises 15 vertexes, namely, 3 differently colored vertexes per each name taken from $\{0, 1, 2, 3, 4\}$. Accordingly, as $\delta$ is a color-preserving simplicial map, the renamed output $\delta(w)$ for each $w \in V(\mathcal{O})$ is tacitly given the matching color, i.e., $\mathrm{color}(w)$.

Figure 4 shows how the decision map $\delta$ assigns an output to each vertex of $\mathrm{Ch}^2 \mathcal{I}$. For those vertexes whose outputs are left unspecified in the figure, it should be understood that such a vertex $w$ receives the output $\delta(\pi_{2,1}(w))$, namely the same output as the parent vertex $\pi_{2,1}(w) \in \mathrm{Ch}\, \mathcal{I}$ does. For instance, the white vertex (of process 0) of the very central simplex of the subdivision is assigned the output 4 by $\delta$, as its parent is $(0, \{0, 1, 2\})$, the white vertex introduced by the first subdivision using Schlegel diagram.

[1] This particular instance of renaming task is taken from [11, Chapter 12].

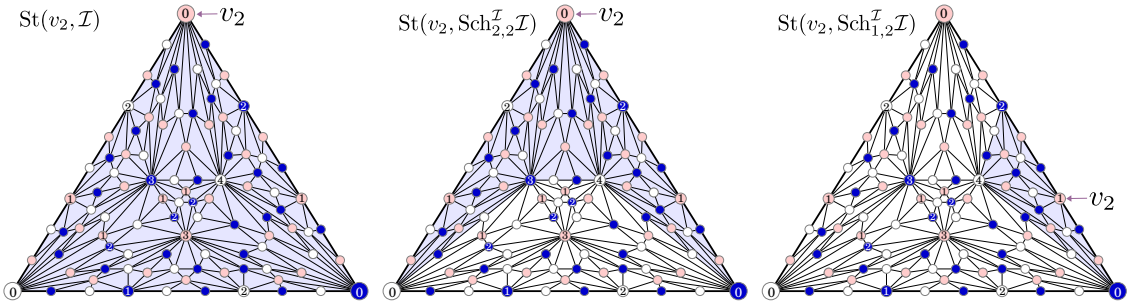


Figure 5: Descendant vertexes in the iterated subdivisions (up to the second iteration, where process 2 is ready for decision)

**Algorithm 4** An optimized generic code for process $i$

**procedure** $\mathsf{IIS}(k, d)$
  **if** $\delta(\pi^{-1}(v_i)) = \{u\}$ for some $u \in V(\mathcal{O})$ **then return** $u$
  $u \leftarrow \mathsf{WOScan}(k, d)$
  **if** $u \neq v_i$ **then** $v_i \leftarrow u$; $\mathsf{IIS}(k+1, n)$
  **else** $\mathsf{IIS}(k, d-1)$

Let us consider an execution of the generic protocol (Algorithm 3) for the renaming task. In the execution, each process $i$ executes a chain of recursive calls of $\mathsf{IIS}$, where each recursive call updates $v_i$ to a vertex of a finer subdivision. For instance, consider the following particular recursive call chain for process 2:

$$\begin{aligned} 2 \in \mathcal{I} &\xrightarrow{\mathsf{IIS}(1,2)} 2 \in \mathrm{Sch}^{\mathcal{I}}_{2,2}\mathcal{I} \xrightarrow{\mathsf{IIS}(1,1)} (2, \{1,2\}) \in \mathrm{Sch}^{\mathcal{I}}_{1,2}\mathcal{I} \\ &\xrightarrow{\mathsf{IIS}(2,2)} (2, \{1,2\}) \in \mathrm{Sch}^{\mathrm{Ch}\,\mathcal{I}}_{2,2}(\mathrm{Ch}\,\mathcal{I}) \\ &\xrightarrow{\mathsf{IIS}(2,1)} (2, \{(0, \{0,1,2\}), (2, \{1,2\})\}) \in \mathrm{Sch}^{\mathrm{Ch}\,\mathcal{I}}_{1,2}(\mathrm{Ch}\,\mathcal{I}), \end{aligned}$$

where each transition $u \in \mathrm{Div}\,\mathcal{I} \xrightarrow{\mathsf{IIS}(k,d)} u' \in \mathrm{Div}'\mathcal{I}$ indicates that a recursive call $\mathsf{IIS}(k, d)$ updates $v_2$ from $u$ to $u'$, which are the vertexes of subdivisions $\mathrm{Div}\,\mathcal{I}$ and $\mathrm{Div}'\mathcal{I}$, respectively. Process 2 terminates the execution of the protocol with a final output $\delta((2, \{(0, \{0,1,2\}), (2, \{1,2\})\})) = 1$.

This recursive call chain, however, could have decided the final output at an earlier stage of recursion, because the vertex $v_2 \in \mathrm{Div}\,\mathcal{I}$ at each recursive call can only be updated to a vertex (of matching color) covered by $\mathrm{St}(v_2, \mathrm{Div}\,\mathcal{I})$ by the successive recursive calls. Figure 5 illustrates the first few steps, where the shaded part indicates the simplexes of $\mathrm{Ch}^2\mathcal{I}$ covered by the star at each recursive step. Initially, when the input complex $\mathcal{I}$ is not yet subdivided, $\mathrm{St}(v_2, \mathcal{I})$ covers all the vertexes of $\mathrm{Ch}^2\mathcal{I}$; After the first recursive call $\mathsf{IIS}(1, 2)$, $\mathrm{St}(v_2, \mathrm{Sch}^{\mathcal{I}}_{2,2}\mathcal{I})$ covers fewer vertexes in a smaller region but they do not agree with the outputs carried by $\delta$ (they can be either 0 or 1); After the second recursive call $\mathsf{IIS}(1, 1)$, $\mathrm{St}(v_2, \mathrm{Sch}^{\mathcal{I}}_{1,2}\mathcal{I})$ covers even fewer vertexes and they are all carried to the same output vertex 1 by $\delta$. Hence, as soon as $v_2$ is updated to the vertex $(2, \{1,2\})$ by the recursive call $\mathsf{IIS}(1, 1)$, we can decide the output of process 2 by $\delta((2, \{1,2\})) = 1$, skipping the remaining recursive calls.

By the observation so far, we can see that the generic protocol (Algorithm 3) can be further optimized to perform fewer shared memory operations by skipping redundant recursive calls: Once a process has reached to a point where a sole final output is determined by the decision map $\delta$, the remaining recursive calls can be skipped. To make it precise, for every $v \in \mathrm{Div}\,\mathcal{I}$ where $\mathrm{Div}\,\mathcal{I}$ is an intermediate subdivision toward the finest subdivision $\mathrm{Ch}^K\,\mathcal{I}$, let us define the *descendants* of $v$ by $\pi^{-1}(v) = \{u \in \mathrm{Ch}^K\,\mathcal{I} \mid \pi(u) = v\}$, where $\pi : V(\mathrm{Ch}^K\,\mathcal{I}) \to V(\mathrm{Div}\,\mathcal{I})$ is the corresponding parent map. We present the optimized version of the generic protocol in Algorithm 4. (The omitted procedure $\mathsf{WOScan}$ is the same as in Algorithm 3.)

We notice that, for a particular decision map $\delta$ and each process $i$, the value $\delta(\pi^{-1}(v_i))$ can be precomputed for every possible vertex assigned to $v_i$ in advance of actual execution of the protocol.

**Algorithm 5** A customized code for the renaming task

**procedure** $\mathsf{IIS}(k, d)$ ▷ CODE FOR PROCESS 0
  **if** $v_0 = (0, \{0, 1, 2\})$ **then return** 4
  **else if** $d = 1 \wedge v_0 = 0$ **then return** 0
  **else if** $d = 1$ **then return** 2
  **else**
    $u \leftarrow \mathsf{WOScan}(k, d)$
    **if** $u \neq v_0$ **then** $v_0 \leftarrow u$; $\mathsf{IIS}(k + 1, n)$
    **else** $\mathsf{IIS}(k, d - 1)$

**procedure** $\mathsf{IIS}(k, d)$ ▷ CODE FOR PROCESS 1
  **if** $k = 1 \wedge d = 1 \wedge v_1 = 1$ **then return** 0
  **else if** $k = 1 \wedge v_1 = (1, \{0, 1\})$ **then return** 1
  **else if** $k = 1 \wedge v_1 = (1, \{1, 2\})$ **then return** 2
  **else if** $\left[\begin{array}{l} k = 2 \wedge v_1 = (1, \tau) \text{ for some } \tau \\ \text{s.t. } \{(1, \{0, 1, 2\}), (2, \{0, 1, 2\})\} \subseteq \tau \end{array}\right]$ **then return** 2
  **else if** $k = 2 \wedge d = 1$ **then return** 3
  **else**
    $u \leftarrow \mathsf{WOScan}(k, d)$
    **if** $u \neq v_1$ **then** $v_1 \leftarrow u$; $\mathsf{IIS}(k + 1, n)$
    **else** $\mathsf{IIS}(k, d - 1)$

**procedure** $\mathsf{IIS}(k, d)$ ▷ CODE FOR PROCESS 2
  **if** $k = 1 \wedge d = 1 \wedge v_2 = 2$ **then return** 0
  **else if** $k = 1 \wedge (v_2 = (2, \{0, 2\}) \vee v_2 = (2, \{1, 2\}))$ **then return** 1
  **else if** $\left[\begin{array}{l} k = 2 \wedge v_2 = (2, \tau) \text{ for some } \tau \\ \text{s.t. } \{(1, \{0, 1, 2\}), (2, \{0, 1, 2\})\} \subseteq \tau \end{array}\right]$ **then return** 1
  **else if** $k = 2 \wedge d = 1$ **then return** 3
  **else**
    $u \leftarrow \mathsf{WOScan}(k, d)$
    **if** $u \neq v_2$ **then** $v_2 \leftarrow u$; $\mathsf{IIS}(k + 1, n)$
    **else** $\mathsf{IIS}(k, d - 1)$

This means that, specializing the code of Algorithm 4 w.r.t. the particular decision map $\delta$, we can generate a further optimized implementation code.

Algorithm 5 gives such a code customized for the above renaming task for 3 processes. Observe that, precomputing descendant vertexes, program specialization has eliminated those redundant recursive calls which are not on reachable execution paths.

Here we notice that the optimization method discussed above is applicable to any protocol of the form $\delta \circ \mathrm{Ch}^K \mathcal{I}$. Furthermore, the optimized code is mechanically derived by specializing the generic protocol w.r.t. the concrete instance of decision map $\delta$.

Although the program specialization gives a general optimization method, it heavily depends on each protocol instance how much memory access can be reduced. At one extreme, a protocol $(\mathcal{C}, \mathcal{C}, \pi \circ \mathrm{Ch}\,\mathcal{C})$, where $\mathcal{C} = \overline{\{v_0, \ldots, v_d\}}$ and $\pi : V(\mathrm{Ch}\,\mathcal{C}) \to V(\mathcal{C})$ is the parent map, is optimized to a protocol $(\mathcal{C}, \mathcal{C}, \Phi)$ with a trivial carrier map such that $\Phi(\sigma) = \overline{\sigma}$ that performs no shared memory access. At the other extreme, a protocol $(\mathcal{C}, \mathrm{Ch}\,\mathcal{C}, \iota \circ \mathrm{Ch}\,\mathcal{C})$ for chromatic agreement task, where $\iota : V(\mathrm{Ch}\,\mathcal{C}) \to V(\mathrm{Ch}\,\mathcal{C})$ is an identity vertex map, is not optimized at all by specialization,[2] since each vertex of $\mathrm{Ch}\,\mathcal{C}$ is assigned a different output. Thus, there is no general theorem on the reduction in the number or complexity of memory access. Furthermore, the code derived by the optimization method is, as is often the case with those obtained by automatic program generation, inevitably less structured than those protocols which are manually devised with human insights (e.g., the protocols [5, 10] for renaming task).

[2]To be precise, any protocol is specialized to a code that at least skips the subdivisions on 0-dimensional simplexes. The original implementation of immediate snapshot by Borowsky and Gafni can also be optimized likewise.

# 5 Conclusion and Future Work

We have shown that the multi-round protocol for the immediate snapshot by Borowsky and Gafni can be reformulated to conform to, in terms of combinatorial topology, Kozlov's construction of the standard chromatic subdivision via Schlegel diagrams. This gives a topologically smoother account for the protocol, where each round is simply a subdivision using Schlegel diagram. This topological simplicity has led to a straightforward method for optimizing distributed protocols defined by means of the iterated immediate snapshot: Each process executing the protocol narrows down the set of possible outputs per each round and can decide the final output at an earlier round, beyond which the same final output is reached no matter how the remaining rounds are executed.

The present paper exemplified that a topologically simpler modeling can better incorporate the theoretical results in topological studies on distributed computing into the more practical side of distributed systems. In this respect, it would be of interest of future investigation to generalize the result to encompass shared memory systems of different failure models such as [18, 9, 8]. Developing an appropriate multi-round protocol that operates on a topological model of a particular failure model, we would be able to optimize the corresponding class of protocols. Such an enhanced multi-round protocol would necessarily need to employ an augmented set of memory operations in a way that the topological structure induced from the extra operations (e.g., the one induced from the test-and-set operation [12]) is compatible with the failure model.


## Acknowledgment

I would like to thank Nayuta Yanagisawa for his valuable comments on a draft of this paper. This work was supported by JSPS KAKENHI Grant Number 16K00016.


# References


[1] Hagit Attiya and Sergio Rajsbaum. The combinatorial structure of wait-free solvable tasks. *SIAM J. Comput.*, 31(4):1286–1313, 2002.

[2] Fernando Benavides and Sergio Rajsbaum. The read/write protocol complex is collapsible. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium*, volume 9644 of *LNCS*, pages 179–191. Springer, 2016.

[3] Richard Bird. *Pearls of Functional Algorithm Design*. Cambridge University Press, 2010.

[4] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing, STOC*, pages 91–100. ACM, 1993.

[5] Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, pages 41–51. ACM, 1993.

[6] Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computations. In *Proc. of the 16th ACM Symposium on Principles of Distributed Computing*, pages 189–198, 1997.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[8] Eli Gafni, Yuan He, Petr Kuznetsov, and Thibault Rieutord. Read-write memory and $k$-set consensus as an affine task. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, pages 6:1–6:17, 2016.

[9] Eli Gafni, Petr Kuznetsov, and Ciprian Manolescu. A generalized asynchronous computability theorem. In *ACM Symposium on Principles of Distributed Computing, PODC '14*, pages 222–231. ACM, 2014.

[10] Eli Gafni and Sergio Rajsbaum. Recursion in distributed computing. In *Stabilization, Safety, and Security of Distributed Systems: 12th International Symposium, SSS 2010*, volume 6366 of *LNCS*, pages 362–376. Springer, 2010.

[11] Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.

[12] Maurice Herlihy and Sergio Rajsbaum. Set consensus using arbitrary objects (preliminary version). In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 324–333. ACM, 1994.

[13] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, 1999.

[14] Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM J. Comput.*, 36(2):457–497, 2006.

[15] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial evaluation and automatic program generation*. Prentice Hall, 1993.

[16] Dmitry Kozlov. *Combinatorial Algebraic Topology*. Springer, 2008.

[17] Dmitry N. Kozlov. Chromatic subdivision of a simplicial complex. *Homology, Homotopy and Applications*, 14(2):197–209, 2012.

[18] Vikram Saraph, Maurice Herlihy, and Eli Gafni. Asynchronous computability theorems for t-resilient systems. In *Distributed Computing - 30th International Symposium, DISC 2016*, pages 428–441, 2016.

[19] Günter M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer, 1995.

# Appendix

## A A Quick Look at Partial Evaluation

This appendix gives a brief overview of partial evaluation, a program optimization technique by program specialization. Partial evaluation is a matured field that has a long history of research. For details that cannot be covered in the following short overview, readers are advised to consult a textbook, say [15].

The fundamental idea of partial evaluation is quite simple. Suppose we are given a program and some of the expected inputs to it are known in advance. Then certain portions of the program may be *precomputed* w.r.t. the known inputs, by which the source program is transformed to an optimized one: The transformed program contains fewer computation steps to be performed at run-time. A subpart of the program is called *static*, if it does not depend on the inputs to be given at run-time; Otherwise, it is called *dynamic*. In particular, the known inputs are called static and the remaining inputs are called dynamic. It is the task of *binding-time analysis* to identify static parts as larger as possible for the chance of better optimization.

Let us see how a simple program that computes exponentiation $n^m$ for non-negative integers $n$ and $m$ can be optimized by partial evaluation.[3] Such a program would be simply defined in a recursive style, as follows:

$$\mathrm{expt}(n, m) \;\equiv\; \mathsf{if}\ m = 0\ \mathsf{then\ return}\ 1\ \mathsf{else\ return}\ n \times \mathrm{expt}(n, m-1).$$

Suppose the second input $m$ is known 9. Instantiating $m$ with 9, we get:

$$\mathrm{expt}(n, \underline{9}) \;\equiv\; \mathsf{if}\ \underline{9 = 0}\ \mathsf{then\ return}\ \underline{1}\ \mathsf{else\ return}\ n \times \mathrm{expt}(n, \underline{9-1}),$$

where the underlined parts are the static ones, which are identified by binding-time analysis. Evaluating the static subexpressions, we obtain

$$\mathrm{expt}(n, \underline{9}) \;\equiv\; \mathsf{if}\ \underline{\mathit{false}}\ \mathsf{then\ return}\ \underline{1}\ \mathsf{else\ return}\ n \times \mathrm{expt}(n, \underline{8}).$$

Pruning the unreachable branch and unfolding the recursive call $\mathrm{expt}(n, \underline{8})$, we get

$$\mathrm{expt}(n, \underline{9}) \;\equiv\; n \times \big(\mathsf{if}\ \underline{8 = 0}\ \mathsf{then\ return}\ \underline{1}\ \mathsf{else\ return}\ n \times \mathrm{expt}(n, \underline{8-1})\big),$$

which reveals new static parts subject to further partial evaluation. Repeating this process, we will obtain the final transformation result:

$$\mathrm{expt}(n, \underline{9}) \;\equiv\; n \times n \times n \times n \times n \times n \times n \times n \times n.$$

Though the above simple example of partial evaluation improves the source program only marginally, just removing the overhead involved in conditional branching and recursive calls, the effect of optimization is amplified by applying it where execution bottleneck exists. In this paper, we are specifically concerned with application to the shared memory bottleneck. Another strength of partial evaluation is that the whole transformation process is mechanizable: Once we write a simple program, whose correctness is easier to reason about, we may automatically obtain one that is still correct yet optimized.

Optimization by partial evaluation, or program transformation by specialization in general, rarely improves computational complexity. In most cases, it improves efficiency only by a constant factor. As for the example of exponentiation, for instance, it is well known that an algorithm of logarithmic complexity is obtained by the technique of repeated squaring [7]. However, this kind of algorithmic leap usually needs human insights on the mathematical structure behind the problem to be solved. It is a central topic of program transformation how to optimize programs semi-automatically — mostly by mechanical 'calculation' on programs but with a little exploitation of the mathematical structure behind each particular problem. There has been lots of work done in this direction and several illuminating examples can be found in [3].

[3] This is the typical example that first appears in introductory texts of partial evaluation.