Musical Benches

Eli Gafni
Computer Science Department, UCLA
eli@cs.ucla.edu

Sergio Rajsbaum
Math Institute, UNAM
rajsbaum@math.unam.mx

Abstract

We propose the *musical benches problem* to model a wait-free coordination difficulty that is orthogonal to previously studied ones such as agreement or symmetry breaking (leader election or renaming). A *bench* is the usual binary consensus problem for 2 processes. Assume $n+1$ processes want to sit in n benches as follows. Each one starts with a preference, consisting of a bench and one place (left or right) in the bench where it wants to sit. Each process should produce as output the place of the bench where it decides to sit. It is required that no two processes sit in different places of the same bench. Upon the observance of a conflict in one of the benches an undecided process can "abandon" its initial bench and place and try to sit in another bench at another place.

The musical benches problem is so called because processes jump from bench to bench trying to find one in which they may be alone or not in conflict with one another. If at most one process starts in each bench, the problem is trivially solvable– each process stays in its place. We show that if there is just one bench where two processes rather than one, start, the problem is wait-free unsolvable in read/write shared memory. This impossibility establishes a new connection between distributed computing and topology, via the Borsuk-Ulam theorem.

The musical benches problem seems like just a collection of consensus problems, where by the pigeon hole principle at least one of them will have to be solved by two processes. Consequently, one is tempted to try to find a bivalency impossibility proof of the FLP style. Our second result shows that there is no such proof: We present an algoritm to solve the musical benches problem using set agreement, a primitive stronger than read/write registers, but weaker than consensus. Thus, an FLP-style impossibility for musical benches will imply an FLP-style impossibility of set-consensus.

The musical benches problem can be generalized by considering benches other than consensus, such as set agreement or renaming, leading to a very interesting class of new problems.

1. Introduction

We consider an n processes asynchronous, single-write/multi-reader shared memory system, where any number of processes may fail by crashing. A protocol in this model is *wait-free*: it guarantees that any process will terminate within a fixed number of steps, independent of the level of contention and the execution speeds of the other processes. Understanding the possibilities and limitations of this model is central to distributed computing in general for several reasons. Impossibility results in this model translate to impossibility results for a model where at most t processes can crash [9], sometimes with more powerful primitives [21], or translate into lower bound on the round complexity of synchronous systems [14, 17]. Indeed, some papers [24, 25] have developed unified frameworks to study synchrony, asynchrony and even partial synchrony, where our wait-free model plays a central role. Furthermore, this model has been shown to have the same computational power as other models, such as message passing when $t < n/2$ [2].

The fundamental problem of wait-free computation is to characterize the circumstances under which synchronization problems have wait-free solutions, and to derive efficient solutions when they exist. This is a difficult problem because one must reason about complex algorithms that operate in the presence of uncertainty and partial information created by non-determinism, asynchrony, and failures. Powerful new tools have been developed based on algebraic topology for analyzing the semantics and complexity of distributed algorithms in a variety of models and architectures; see e.g. [18] for an historical survey and references herein. We use such tools in this paper both to prove impossibility results and to derive algorithms.

Distributed computing theory development has been fostered by the identification of particular problems, that capture the essence of wait-free coordination difficulties. First, *consensus* [13] and *set agreement* [11] serve to model the difficulty of processes to converge on a small number of decisions. Other similar problems have been identified, like *approximate agreement* [12], or *loop agreement* [23]. The problem of *renaming* [3] models the opposite difficulty: breaking symmetry.

This paper proposes a new schema to model a coordination difficulty that is orthogonal to the ones described above. It is inspired by the *musical chairs* game, where players march to music around a row of chairs numbering one less than the players and scramble for places when the music stops. In our scenario we have *benches* with $k-1$ *places* for players to sit in. If at most $k-1$ players want to sit in a bench, they can easily do it. But if k or more want to sit in a bench, some must leave and look for another bench. Assume $k-1$ players want to sit in each bench, except for one bench in which k players want to sit. Is there a wait-free algorithm that allows all the players to find a bench where to sit? Indeed, we show in this paper that this problem is related to a topological theorem which is different in nature from previously used results such as Sperner's lemma.

The general idea is to consider a problem that is not solvable once the number of participants exceed some threshold, $k-1$. We then replicate the problem as many times as we wish, calling each replica a bench, and wake up $k-1$ processes in every bench but one, where we wake up k processes. Now we allow processes to jump from bench to bench each trying to acquire a valid seat in one of the benches, such that all those on the same bench comply with the bench seating requirement. For example, a $(2,3)$-set agreement bench is wait-free solvable with threshold $k=3$, when there are at most 2 participants; we would wake up 2 processes in every bench but one, where we wake up 3 processes. The *generalized musical benches conjecture* is that the schema always leads to an impossible problem.

In this paper we do the first step in this direction. We take a bench of two places, with the requirement that all that choose the same bench have to choose the same place. This corresponds to binary consensus. We take n benches and we wake up one process per bench, aside from a single bench (apriori determined) in which we wake up two processes, in distinct seats. For lower bounds it is sufficient to consider only two benches (3 processes); we call this consensus instantiation of the general scheme the *musical benches problem.*

Our contributions are the following.

- The introduction of a schema that captures a new kind of distributed coordination difficulty, and the study of the binary consensus instantiation, the musical benches problem.
- An impossibility proof showing that the musical benches problem is wait-free unsolvable in read/write shared memory.
- This impossibility establishes a new connection between distributed computing and topology, via the Borsuk-Ulam theorem.
- An algorithm to solve the musical benches problem using set agreement, a primitive stronger than read/write registers, but weaker than consensus.

The 2-places musical benches problem seems like just a collection of consensus problems, and thus one is tempted to try to find a bivalency impossibility proof of the FLP style. More precisely, by the pigeon hole principle at least one of the benches would have to be solved by two processes, since there are more processes than benches. Thus, two processes would have to solve consensus on that particular bench, contradicting FLP. The algorithm in the last item is significant because it shows that no bivalency argument of this style exists. Namely, it implies that if the musical benches problem is impossible then $(3, 2)$-set agreement is impossible. Thus, the existance of such a bivalency proof would imply the celebrated $(3, 2)$-set agreement impossibility [6, 26, 30]. But this impossibility requires Sperner's lemma, a higher dimensional statement for which no bivalency arguments are known.

The connection to a result as important as the Borsuk-Ulam theorem establishes one more significant bridge between distributed computing and topology. The theorem is "one of the most useful tools offered by elementary algebraic topology to the outside world" [28]. Recall that it implies Sperner's lemma (which is equivalent to Brouwer fixed point theorem), but not the opposite. There are several equivalent versions of the Borsuk-Ulam theorem, the easisest to remember is illustrated in Figure 1 (from [28]), for the $n = 2$ dimensional case. It states that if you take a rubber ball, deflate and crumble it, and lay it flat, then there are two points on the surface of the ball that were diametrically opposite and now are lying on top of one another. More formally, we see in part (b) of the figure, that[1] for every continuous map $f : S^n \rightarrow I\!R^n$, there

Figure 1. Borsuk-Ulam Theorem in 2 dimensions

exist a point $x \in S^n$ such that $f(x) = f(-x)$.

The rest of this paper is organized as follows. In Section 2 we describe formally the musical benches problem. In Section 3 we show that it is impossible to wait-free solve in a read/write shared memory system. In Section 4 we present the algorithm that solves it using a $(2, 3)$-set agreement object. Section 5 contains the conclusions.

[1] Where S^n is the n-dimensional unit sphere, and $I\!R^n$ the Euclidean space of dimension n.

2 The Musical Benches Problem

We consider the usual asynchronous shared memory model, composed of single-writer multi-reader registers. In Section 4 we will extend the shared memory with $(2,3)$-set agreement objects. We now describe the binary-consensus musical benches problem, first intuitively and then more formally.

We can think of 2-process binary consensus as a bench with two places, designated 1 and -1. Processes p_1 and p_{-1}, wake up at places 1 and -1, respectively. In a solo execution they must return the places they wake up in. Otherwise, in an execution where both participate, they return the same places. This consensus task[2] is impossible to solve wait-free in the read-write shared-memory model [13, 20], and trivially solvable if at most one process wakes up. We call an instance of this problem a *bench.* What will happen if we add a second bench, with places $2, -2$, and wake up either process p_2 at slot 2, or p_{-2} at slot -2, but not both? In executions with no conflict, i.e., either p_{-1} or p_1 wake up but not both, the participating processes return the places they wake up in. Only if both p_{-1} and p_1 wake up, then it is free-for-all and any participating process can go to any seat. Is the binary-consensus 2 benches problem read-write wait-free solvable? One feels a strong intuition (unlike in the set agreement impossibility) as to why it should not be solvable: if a process from bench 1 jumps to bench 2, it just creates the same problem in bench 2, since we have the freedom of who to wake up in bench 2 as to try to defeat consensus there. Indeed, the problem has no solution, but surprisingly, we later essentially prove that this intuition is not exactly right.

The musical benches problem is formalized in terms of the usual notion of *task,* a one-shot decision problem specified in terms of an input/output relation Δ. The processes start with private input values, and must eventually decide on output values, by writing to a write-once variable. The relation Δ specifies for each set of (ids, input values) pairs, what are the allowed output values for each id. In our case, each id is associated to a single input, so we may describe Δ as follows. The i-th *bench* is defined by the set of input vectors $\{(p_{-i}, p_i), (p_i), (p_{-i})\}$, and the relation:

$$\begin{aligned} \Delta(p_{-i}, p_i) &= \{(-i, -i), (i, i)\}, \\ \Delta(p_{-i}) &= \{(-i)\}, \\ \Delta(p_i) &= \{(i)\}. \end{aligned}$$

This is illustrated in Figure 2(a). It is sometimes convenient to consider only the output values, and disregard the processes ids, as depicted in Figure 2(b).

Figure 2. The first consensus bench

Consider an algorithm for processes p_{-i}, p_i where the first operation by a process is to write its id to shared memory, and that includes an operation to a write-once *decision variable.* A process *participates* in an execution if it executes its first operation. The *input vector* of an execution contains the ids of the

[2] In the more usual description of consensus there is a set of possible inputs, and a process can wake up with any of these inputs. In our description a process has only one possible input, and different processes have different inputs. Both descriptions are equivalent, but the one we use is more comfortable for our purposes.

participating processes. A process *decides* in an execution if it writes to the decision variable, and the value decided is the value written to the variable. The *output vector* of an execution contains the values decided by the processes, or $\perp$ if the process did not decide. The algorithm *solves* the i-th bench problem if in every execution with input vector I, the output vector O can be extended (by replacing $\perp$ entries with other values) to a vector in $\Delta(I)$, and a process that does not fail decides.

The *musical benches problem* of size b is also a task specified in terms of a relation Δ. The input vectors are over $\{p_{-i}, p_i | 1 \leq i \leq b\}$, and the output vectors over $\{-i, i, \perp | 1 \leq i \leq b\}$. In this paper we study the case of $b = 2$, as illustrated in Figure 3, disregarding ids and omitting the dotted arrows of Δ for single vertices, to avoid cluttering the figure. Formally, Δ is:

Figure 3. Musical benches task

$$\begin{aligned}
\Delta(p_{-1}, p_1, p_2) &= \{(x_1, x_2, x_3) \,|\forall i, j,\ x_i \in \{1, -1, 2, -2\},\ x_i + x_j \neq 0\} \\
\Delta(p_1, p_2) &= \{(1, 2)\} \\
\Delta(p_{-1}, p_1) &= \{(-1, -1), (1, 1), (-2, -2), (2, 2)\} \\
\Delta(p_{-1}) &= \{(-1)\} \\
\Delta(p_1) &= \{(1)\}
\end{aligned}$$

and so on for $\Delta(p_{-1}, p_1, p_{-2})$, $\Delta(p_1, p_{-2})$, $\Delta(p_{-1}, p_{-2})$, $\Delta(p_{-1}, p_2)$, $\Delta(p_{-2})$, and $\Delta(p_2)$. Notice it includes the first bench, and a restriction of the 2nd bench that disallows p_{-2} and p_2 participating together.

3 Impossibility of the Musical Benches Problem

Here we prove that the musical benches problem is wait-free unsolvable. For clarity we just prove the 2 benches case. Extension to any b is simple. For the proof we use the Borsuk-Ulam theorem, or rather, its discrete version, known as Tucker's lemma. We will need some basic topology notions, presented in the Appendix.

Tucker's lemma is described in [28] as follows. Let T be some (finite) triangulation of the n-dimensional ball B^n. We call T *antipodally symmetric on the boundary* if the set of simplices of T contained in $S^{n-1} = \partial B^n$ is a triangulation of S^{n-1} and it is antipodally symmetric; that is, if $\sigma \subset S^{n-1}$ is a simplex of T, then $-\sigma$ is also a simplex of T.

Theorem 3.1 (Tucker's lemma) *Let T be a triangulation of B^n that is antipodally symmetric on the boundary. Let*

$$\lambda : V(T) \longrightarrow \{1, -1, 2, -2, \ldots, n, -n\}$$

be a labeling of the vertices of T that satisfies $\lambda(-v) = -\lambda(v)$ for every vertex $v \in \partial B^n$ (that is, λ is antipodal on the boundary). Then there exists a 1-simplex (an edge) in T that is **complementary***; i.e., its two vertices are labeled by opposite numbers.*

Figure 4. Illustration of 2-dimensional Tucker's lemma

We will only need the 2-dimensional version, illustrated in Figure 4.

A task and an algorithm solving it can be represented geometrically using topology terminology, e.g. [22, 26]. A task specification for $n+1$ processes is given by an input complex $\mathcal{I}$, an output complex $\mathcal{O}$, and a relation Δ carrying each input simplex of $\mathcal{I}$ to a set of n-simplexes of $\mathcal{O}$. This definition has the following operational interpretation: $\Delta(S^m)$ is the set of legal final states in executions where only certain $m+1$ processes corresponding to the vertices of S^m out of $n+1$ processes participate (the rest fail without taking any steps). A protocol *solves* a task if when the processes run their programs, they start with mutually compatible input values, represented by a simplex S, communicate with one another, and eventually halt with some set of mutually compatible output values, representing a simplex in $\Delta(S)$. The musical benches problem of the previous section can be represented in this form by using simplices instead of vectors.

Any protocol that solves a task has an associated *protocol complex* $\mathcal{P}$, in which each vertex is labeled with a process id and that process's final state, called its *view*. Each simplex thus corresponds to an equivalence class of executions that "look the same" to the processes at its vertexes. For $0 \leq m \leq n$, we understand $\mathcal{P}(S^m)$ for a given S^m in the input complex to be the complex generated by all executions starting in S^m, in which only the processes in $ids(S^m)$ take part (the rest fail without taking any steps). If a simplex R is in $\mathcal{P}(S^m)$, we say that R is *reachable from* S^m.

Let $\mathcal{P}$ be the protocol complex for a protocol. If S is an input simplex, let $\mathcal{P}(S) \subset \mathcal{P}$ denote the complex of final states reachable from the initial state S. Expressed in the topology notation, we can see that a protocol solves a decision task $\langle \mathcal{I}^n, \mathcal{O}^n, \Delta \rangle$ if and only if there exists a color-preserving (i.e., process id-preserving) simplicial map $\delta : \mathcal{P} \rightarrow \mathcal{O}^n$, called a *decision map*, such that for every input simplex S, $\delta(\mathcal{P}(S)) \subset \Delta(S)$.

Our basic strategy is the following. We assume that we have a protocol with complex $\mathcal{P}$ that wait-free solves a task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$. As in [8] without loss of generality we can assume that $\mathcal{P}$ is the result of some large enough number of iterated immediate snapshots. Let S^ℓ be an input simplex, $\mathcal{S}^\ell$ the complex of its faces, and $\mathcal{P}$ a protocol. For a wait-free model of computation, prior research (e.g. [5, 6, 30]) has shown that $\mathcal{P}(S^\ell)$ can be regarded as a subdivision of S^ℓ.

Assume there is an algorithm solving the musical benches problem with protocol complex $\mathcal{P}$. Let T be the input complex to the problem (illustrated in Figure 3). The next lemma shows that $\mathcal{P}(T)$ is a subdivision of T, as illustrated in the example of Figure 5.

Lemma 3.2 *If T is the input complex to the musical benches problem then $\mathcal{P}(T)$ is a triangulation of B^2 that is antipodally symmetric on the boundary. Moreover, if λ is the labeling of the vertices of $\mathcal{P}(T)$ induced by the processes decisions, then λ is antipodal on the boundary.*

Sketch of Proof First notice that $|T| \cong B^2$. Then, as mentioned above previous results imply that $\mathcal{P}(T)$

Figure 5. A 1-round protocol subdividing the musical benches input complex

can be regarded as a subdivision of T, and hence as a triangulation of B^2. The boundary corresponds to executions with no conflict in bench 1, thus the problem specification implies that on a face (p_i, p_j) in the boundary processes return i and j, respectively. Since we take the same number of iterations on each face we can easily see that the triangulation is antipodally symmetric on the boundary of $\mathcal{P}(T)$. □

Theorem 3.3 *There is no wait-free solution to the musical benches problem.*

Proof: It follows from Lemma 3.2 that we can apply Theorem 3.1 and conclude that there is an edge in $\mathcal{P}(T)$ that is complementary. Thus, its two vertices are labeled by opposite numbers, which means there is an execution where two processes decide opposite numbers, violating the musical benches problem consensus requirement. ■

4 Solving the musical benches problem with more powerful primitives

We have seen that the binary-consensus musical benches problem is wait-free unsolvable in read/write shared memory. We show here that this impossibility implies the wait-free impossibility of solving $(2, 3)$-set agreement in read/write shared memory [6, 26, 30]. We prove this by presenting an algorithm that solves the musical benches problem using a shared memory extended with $(2, 3)$-set agreement objects. Our algorithm is described for the case of two benches (3 processes) since our main motivation is proving that no bivalency argument in the FLP style [13] exists for the musical benches. We know that the impossibility of $(2, 3)$-set agreement cannot be proven without reference to the 2-connectivity of $\mathcal{P}$ for 3 processes [6, 26, 30].

A $(2, 3)$-*set agreement object* can be accessed by 3 processes, and the object returns to a process one of the ids of a process that invoked it, such that at most 2 different ids are returned by the object. We assume w.l.o.g. that if a process p_i gets back from the object p_j, then p_j gets back itself, p_j [6].

The musical benches protocol appears in Figure 7. It accesses a single $(2, 3)$-set agreement object. The main idea is to create a "hole" inside the immediate snapshots subdivision (of Figure 5), and we do this by doing participating set protocol [7] and sending all those stuck at level 3, to a $(2, 3)$-set agreement object. A winner process that gets back its own id from the object stays at level 3. A loser, that gets back a different id, continues down to level 2 and proceeds with the participating set protocol. Since if all 3 are stuck at level 3, then at least one will lose, and we create the hole by preventing the formation of the all-see-all simplex in the center of the subdivision.

The code invokes a decision function f that takes as input a final *view* of a process and produces a decision value; it is specified in the Appendix. The corresponding protocol complex appears in Figure 6, together with the decision function f on each one of the final views. A process p_i computes its view, the

pair (S_i, view_i), by executing the protocol code, and it produces its decision value by applying f to its view (in the last line of the code).

The protocol works as follows. Local variables are subindexed by the process ids, and shared variables are not subindexed. The first part of the protocol is the Participating Set protocol of [7] for the case of 3 levels, except that it accesses a set agreement object. A process p_i computes in this part a set of ids S_i, such that

1. For all i, $i \in S_i$.
2. For all i, j, either $S_i \subseteq S_j$ or $S_j \subseteq S_i$.
3. For all i, j, if $i \in S_j$ then $S_i \subseteq S_j$.
4. There are at most two indices i, j such that $|S_i| = |S_j| = 3$.

The first three are the requirements of the participating set problem in [7]. Sets satisfying these properties correspond to the subdivided simplex in either side of Figure 5 (i.e., spanned by the corners p_{-1}, p_1, p_2 or p_{-1}, p_1, p_{-2}). The 4-th property is achieved through the set agreement object, invoked by p_i with the operation $\mathrm{setAg}(i)$. It has the effect of removing the simplex in the center of the subdivision (impossible that the three processes produce sets of size 3), and leaving just its boundary (at most two processes may produce sets of size 3). In the second part of the protocol only processes with sets of size 3 participate, and they compute the view_i variables, which have the effect of subdividing this boundary. The following simple lemma implies that the complex of Figure 6 corresponds to the computed views (S_i, view_i).

Lemma 4.1 *Let the views (S_i, view_i) be the vertices of a complex $\mathcal{C}$. A simplex of $\mathcal{C}$ contains a set of vertices if they can be ordered such that both their S_i's and their view_i's are ordered by containment. This complex $\mathcal{C}$ is well defined.*

The correctness of the MB protocol follows from this lemma, by proving that the decision function f does not produce a complementary edge, something that is easily verified in Figure 6 (and noting that on the boundary of $\mathcal{C}$ a process always decides its own id).

Theorem 4.2 *The MB protocol solves the musical benches problem.*

5 Conclusions

We have introduced a scheme that models a new coordination difficulty, the *generalized musical benches problem,* that can be instantiated with any problem, which we call a *bench,* that is solvable for $k-1$ participants, and is not solvable once the number of participants is at least k, for some threshold k. We have conjectured that the generalized musical benches problem is unsolvable for any such instantiation.

In this paper we studied the *musical benches problem,* which is the case obtained by using binary, $(-1, 1)$, consensus benches. Recall that the consensus problem is impossible to wait-free solve in the read-write shared-memory model [13, 20]. It is solvable if we add a new possible output place, 2, which is allowed to be returned in a non-solo execution (e.g. [27, 29]). The "Ambiguity of Choosing" of [10] says that in the solvable version there exists an execution in which one process is "sitting" in one of two places, and like Heisenberg's uncertainty principle we cannot predict where it will appear, and hence the impossibility of solving the problem without place 2. We have seen the problem is unsolvable if we also add place -2 and wake up either process p_2 at slot 2, or p_{-2} at slot -2. The intuition behind this impossibility is more evident to us, than the impossibility of solving wait-free 3-processes 2-set agreement [6, 26, 30]. Processes

Figure 6. The Musical Benches Protocol complex using $(2, 3)$-Set Agreement

p_1 and p_{-1} can resolve their differences only by at least one of them moving to the second bench. But then if (w.lo.g.) one of them moves to place 2, we wake up p_{-2}. Now we are almost at the mirror situation. To resolve the conflict at the second bench we would like to choose the place of a process in the first bench which might have stayed there, but this tantamount to 2-process consensus! It is then surprising that the two bench impossibility problem is based on Borsuk-Ulam, a theorem more difficult to prove than Sperner's lemma, and not on a traditional bivalency argument.

In the "complement" to consensus benches, which we call 2-renaming benches (based on the renaming problem [3]), we have again two benches with places $1, -1, 2, -2$. Processes p_1 and p_{-1} both may wake up at bench 1, while p_2 and p_{-2} wake up at the second bench, in place 2 or -2, respectively. We wake up either p_2 or p_{-2}, but not both. In executions in which not both p_1 and p_{-1} participate, processes return the place they woke up at. Else, all processes return distinct chairs. Again the intuition is evident. Processes p_1 and p_{-1} cannot resolve their differences in the first bench [5, 22, 26]. W.l.o.g at least one of them goes to the second bench, say place 2. We then wake up p_2, and have a mirror situation. We have an impossibility proof for this problem, similar to the one presented in this paper. But it is intriguing to find a single "meta-proof" that applies to both the renaming and the consensus instantiations of the generalized musical benches problem.

We showed that the impossibility of the musical benches problem implies the impossibility of 3-processes 2-set agreement, while the impossibility of renaming implies the impossibility of 4-musical-benches. This puts the two problems somewhere between set agreement and renaming. Which is another step in understanding the R/W implementation relation between the set of R/W unsolvable tasks [15].

```
Initially:
level[j] := 4 and id[j] := ⊥ for j ∈ {1,2,3}; OK_i := false; view_i = ∅;
begin MB protocol(p_i)
  repeat
        level[i] := level[i] − 1;
        for j = 1 to 3 do level_i[j] := level[j] end for
        S_i := {j : level_i[j] ≤ level[i], j ∈ {1,2,3}};
        if |S_i| = 3 then (* level[i] = 3 *)
                          ans_i := setAg(i);
                          if ans_i = i then OK_i := true
                            end if
          else            (* level[i] < 3 or |S_i| < 3 *)
                          OK_i := true
          end if
  until |S_i| ≥ level[i] and OK_i;
  (* end of participating set section *)
  if |S_i| = 3 then
                    id[i] := i;
                    for j = 1 to 3 do id_i[j] := id[j] end for
                    view_i := {j : id_i[j] ≠ ⊥, j ∈ {1,2,3}};
    end if
  decide f(S_i, view_i)
  end protocol
```

Figure 7. Musical Benches Protocol using $(2,3)$-Set Agreement (for process p_i)

References

[1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic Snapshots of Shared Memory. *Journal of the ACM*, 40(4):873–890, 1993.

[2] Attiya H., Bar-Noy A. and Dolev D., Sharing Memory Robustly in Message-Passing Systems. *Journal of the ACM*, 42(1): 124–142, 1995.

[3] Attiya H., Bar-Noy A., Dolev D., Peleg D. and Reischuk R., Renaming In An Asynchronous Environment. *Journal of the ACM*, 37(3):524–548, 1990.

[4] Attiya H. and Rachman O., Atomic Snapshots in O($n \log n$) Operations. *SIAM Journal of Computing,* 27(2):319-340, 1998.

[5] Attiya H., Rajsbaum S., The Combinatorial Structure of Wait-Free Solvable Tasks. *SIAM J. Comput.* 31(4): 1286–1313, 2002.

[6] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t-Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on the Theory of Computing (STOC'93)*, ACM Press, pp. 91-100, June 1993.

[7] Borowsky E. and Gafni E., Immediate Atomic Snapshots and Fast Renaming (Extended Abstract). *Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC'93)*, ACM Press, pp. 41-51, August 1993.

[8] Borowsky E. and Gafni E., A Simple Algorithmically Reasoned Characterization of Wait-Free Computations (Extended Abstract). *Proc. 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, ACM Press, pp. 189–198, August 1997.

[9] Borowsky E., Gafni E., Lynch N. and Rajsbaum S., The BG Distributed Simulation Algorithm. *Distributed Computing,* 14(3):127–146, 2001.

[10] Burns J. and Peterson G, The Ambiguity of Choosing. *Proc. 8th ACM Symposium on Principles of Distributed Computing (PODC),* August 14–16, 1989, Edmonton, Alberta, Canada, pp. 145–157.

[11] Chaudhuri S., More *Choices* Allow More *Faults:* Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation,* 105:132-158, 1993.

[12] Fekete A., Asymptotically Optimal Algorithms for Approximate Agreement. *Distributed Computing,* 4:9–29, 1990.

[13] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[14] Gafni, E. Round-by-Round Fault Detectors: Unifying Synchrony and Asynchrony (Extended Abstract). *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC),* June 28–July 2, 1998, Puerto Vallarta, Mexico, pp. 143–152.

[15] Gafni E. DISC/GODEL presentation: R/W Reductions. Oct. 4 2004. http://www.cs.ucla.edu/˜ eli/eli/godel.ppt

[16] Gafni E., Koutsoupias E., Three-Processor Tasks Are Undecidable. *SIAM J. Comput.* 28(3): 970–983, 1999.

[17] Gafni E., Guerraoui R. and Pochon B., From a Static Impossibility to an Adaptive Lower Bound: The Complexity of Early Deciding Set Agreement. *Proc. 37th ACM Symposium on Theory of Computing (STOC'05)*, Baltimore (MD), May 2005.

[18] Eric Goubault, *A historical note on "Geometry and Concurrency",* http://www.di.ens.fr/˜ goubault/index1.html

[19] Havlicek J. Computable Obstructions to Wait-Free Computability. *Distributed Computing* 13(2): 59–83, 2000.

[20] Herlihy M.P., Wait-Free Synchronization. *ACM Transactions on programming Languages and Systems*, 11(1):124-149, 1991.

[21] Herlihy M., Rajsbaum S., The Decidability of Distributed Decision Tasks (Extended Abstract). STOC 1997: 589-598

[22] Herlihy H., Rajsbaum S., Algebraic spans. *Mathematical Structures in Computer Science* 10(4): 549–573, 2000.

[23] Herlihy H., Rajsbaum S., A classification of wait-free loop agreement tasks. *Theor. Comput. Sci.* 291(1): 55–77, 2003.

[24] Herlihy, M. Rajsbaum, S. and Tuttle, M. Unifying Synchronous and Asynchronous Message-Passing Models. *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC),* June 28–July 2, 1998, Puerto Vallarta, Mexico, pp. 133–142.

[25] Herlihy, M. Rajsbaum, S. and Tuttle, M. An axiomatic approach to computing the connectivity of synchronous and asynchronous systems. *Proc. of the 6th workshop on Geometric and Topological Methods in Concurrency and Distributed Computing (GETCO),* October 4, 2004.

[26] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923, 1999.

[27] Jayanti P., Chandra T. and Toueg S., Fault-tolerant wait-free shared objects. *Journal of the ACM*, 45(3):451-500, 1998.

[28] Jiri Matousek, *Using the Borsuk-Ulam Theorem,* Lectures on Topological Methods in Combinatorics and Geometry, 2003, Springer.

[29] Raynal, M., 1997. Real-time dependable decisions in timed asynchronous distributed systems. *Proc. 3rd Int. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 97).* IEEE Computer Society Press, Newport Beach, 283–290.

[30] Saks M. and Zaharoglou F., Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.

A Appendix: Topology Notions

The unit ball $\{x \in I\!R^d : \| x \| \leq 1\}$ is denoted by B^d, while $S^{d-1} = \{x \in I\!R^d : \| x \| = 1\}$ is the $(d-1)$-dimensional unit sphere.

A *simplex* is a set of vertices, a *complex* is a set of simplexes closed under containment. The *dimension* d of a simplex σ is one less than its number of vertices, and is said to be a d-simplex, sometimes denoted σ^d. A subset of a simplex is called a *face*. It is sometimes convenient to assume a simplex σ is embedded in Euclidean space. For this its vertices are supposed to be affinely independent, and σ is the convex hull of its vertices. The union of all embedded simplices in a comlex $\mathcal{C}$, called the *polyhedron* of $\mathcal{C}$, is denoted $|\mathcal{C}|$, and can be regarded as the (point-set) union of the simplexes in $\mathcal{C}$. The *boundary* of an n-simplex is the subcomplex of σ^n obtained by deleting the single n-dimensional simplex and retaining all its faces.

A *triangulation* of a topological space X is a complex C such that $X \cong |X|$, namely with homeomorphic spaces. The simplest triangulation of the sphere S^{n-1} is the boundary of an n-simplex.

A *vertex map* carries vertices of one complex to vertices of another. A *simplicial map* is a vertex map that preserves simplexes, that is, it sends a set of vertices that form a simplex into a (possibly smaller) set of vertices that also form a simplex. In distributed computing we often consider *properly colored* complexes, where each vertex has associed a color, namely a process id, and no two vertices of the same simplex have the same id. For example, the complex in Figure 5 is colored. A simplicial map on properly colored complexes is *color preserving* if it associates vertexes of the same color. Notice that a color-preserving map preserves dimension.

A complex $\sigma(\mathcal{K})$ is a *subdivision* of a complex $\mathcal{K}$ if:

- each simplex in $\sigma(\mathcal{K})$ is contained in a simplex in $\mathcal{K}$, and
- each simplex of $\mathcal{K}$ is the union of finitely many simplexes in $\sigma(\mathcal{K})$.

Note that $|\mathcal{K}| = |\sigma(\mathcal{K})|$. If $\vec{s}$ is a point in $|\mathcal{K}|$, the *carrier* of $\vec{s}$, denoted *carrier*$(\vec{s}, \mathcal{K})$, is the unique smallest $T \in \mathcal{K}$ such that $\vec{s} \in T$. As an example, the complex in Figure 5 is a subdivision of the complex in the left side of Figure 3.

B Appendix: Decision Function of the Protocol

The MB protocol of Figure 7 uses a decision function f that is implicitly defined in Figure 6, by putting a number besides each vertex. Here we describe it explicitly by writing what is the view of the vertex, and what the value of f on that view (S_i, view_i).

1. $f(S_i, \mathit{view}_i) = -2$ if $|S_i| = 2$ and $i = -1$.
2. $f(S_i, \mathit{view}_i) = -1$ if $|S_i| = 2$ and $i = 1$. **Otherwise:**
3. $f(S_i, \mathit{view}_i) = i$ if $|S_i| \leq 2$. **Otherwise:**
4. $f(S_i, \mathit{view}_i) = -2$ if $-2 \in S_i$. **Otherwise:**

5. $f(S_i, \textit{view}_i) = 1$ if $\textit{view}_i = \{-1\}$.
6. $f(S_i, \textit{view}_i) = 2$ if $\textit{view}_i = \{-1, 1\}$.
7. $f(S_i, \textit{view}_i) = -1$ if $\textit{view}_i = \{1\}$.
8. $f(S_i, \textit{view}_i) = -1$ if $\textit{view}_i = \{1, 2\}$ and $i = 1$.
9. $f(S_i, \textit{view}_i) = -2$ if $\textit{view}_i = \{1, 2\}$ and $i = 2$.
10. $f(S_i, \textit{view}_i) = -2$ if $\textit{view}_i = \{2\}$.
11. $f(S_i, \textit{view}_i) = -2$ if $\textit{view}_i = \{-1, 2\}$ and $i = -1$.
12. $f(S_i, \textit{view}_i) = 1$ if $\textit{view}_i = \{-1, 2\}$ and $i = 2$.