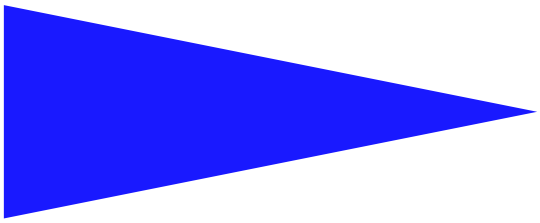
The Committee Decision Problem

Eli Gafni, Sergio Rajsbaum, Michel Raynal, Corentin Travers

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1745



THE COMMITTEE DECISION PROBLEM

E. GAFNI S. RAJSBAUM M. RAYNAL C. TRAVERS

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

The Committee Decision Problem*

E. Gafni** S. Rajsbaum*** M. Raynal**** C. Travers*****

Systèmes communicants

Publication interne n ° 1745 — Septembre 2005 — 17 pages

Abstract: We introduce the (b, n)-*Committee Decision Problem* (CD) - a generalization of the consensus problem. While set agreement generalizes consensus in terms of the number of decisions allowed, the CD problem generalizes consensus in the sense of considering many instances of consensus and requiring a processor to decide in at least one instance. In more detail, in the CD problem each one of a set of n processes has a (possibly distinct) value to propose to each one of a set of b consensus problems, which we call *committees*. Yet a process has to decide a value for at least one of these committees, such that all processes deciding for the same committee decide the same value. We study the CD problem in the context of a wait-free distributed system and analyze it using a combination of distributed algorithmic and topological techniques, introducing a novel reduction technique.

We use the reduction technique to obtain the following results. We show that the $(2, 3)$-CD problem is equivalent to the musical benches problem of Gafni and Rajsbaum (DISC 2005), and both are equivalent to $(2, 3)$-set agreement, closing an open question left there. Thus, all three problems are wait-free unsolvable in a read/write shared memory system, and they are all solvable if the system is enriched with objects capable of solving $(2, 3)$-set agreement. While the previous proof of the impossibility of musical benches was based on the Borsuk-Ulam (BU) Theorem, it now relies on Sperner's Lemma, opening intriguing questions about the relation between BU and distributed computing tasks.

Key-words: Asynchronous distributed system, Wait-free computing, Shared memory, Consensus, Set Agreement, Musical benches.

(Résumé : tsvp)

* This work has been supported by grants from LAFMI (Franco-Mexican Lab in Computer Science) and PAPIIT-UNAM.
** Department of Computer Science, UCLA, Los Angeles, CA 90095, USA eli@cs.ucla.edu
*** Instituto de Matemáticas, UNAM, D. F. 04510, Mexico rajsbaum@math.unam.mx
**** IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, raynal@irisa.fr
***** IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes Cedex, France, travers@irisa.fr

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Le problème des décisions de comités

Résumé : Ce rapport présente un problème de prise de décisions multiples (qui généralise le problème du consensus) et étudie sa calculabilité (à l'aide de techniques de réductions).

Mots clés : Systèmes répartis asynchrones, synchronisation sans attente, mémoire partagée, consensus, accord ensembliste, bancs musicaux.

1 Introduction

In a distributed asynchronous system of n processes where at most t of them can fail by stopping, the (k, n)-*set agreement* problem [7] abstracts away a basic coordination problem: processes have input values, and they must agree on at most k of these values. The problem has no solution if the shared-memory has only read/write registers when $k \leq t$ [4, 16, 20] but is solvable if either $k > t$ or else more powerful communication primitives are available in the system. Set agreement and *consensus*, when $k = 1$, have motivated a lot of research (e.g., [2, 17]) and helped to expand our understanding of distributed computing. The *wait-free* case of $t = n - 1$ has been shown to be fundamental (e.g., [11, 12, 16]), because from this case results can be derived for any value of t [4, 6], and the wait-free techniques can be generalized to other synchronous and partially synchronous models (e.g., [14, 15]), and even models with stronger communication primitives (e.g., [13]). In this paper we concentrate on the wait-free model.

One of the important uses of consensus arises in a distributed state machine (e.g., [19]): the processes are executing a sequence of operations, and they need to agree on the result of each one of the operations, before they can execute the next one. This and other forms of long-lived versions of consensus (e.g., [3]) that we are aware of are sequential, in that processes propose values, then they agree on one of them, and only then they proceed to the next instance of consensus and propose another value. However, it is also very natural to consider concurrent versions of the problem, where a process p_i proposes a vector V_i of values, and each one of them is intended to one of b different consensus problems, called *committees*. We require that processes deciding on the same committee must decide the same value for that committee. Thus, if the processes participate concurrently in b different applications, we can guarantee wait-free progress in at least one application, without using strong communication objects.

We call this generalization of consensus the *committee decision problem* (CD). Notice that the usual termination requirement of consensus is weakened: a process has to decide a value v for only one of the committees, which it can choose; that is, if its decision is the pair (j, v), then all processes choosing to decide for the j-th committee decide the same value v. The decisions should satisfy the standard agreement and validity requirements of consensus: the value decided for a committee was proposed by some process to that committee, and every process deciding on the committee decides the same value. In addition to its possible applications, there seem to be various interesting generalizations that may motivate new research, such as:

- The number of different committees that are decided is at most k.
- At most k different values are decided for each committee.
- A process that decides must decide in at least k committees.

The CD problem cannot be solved when $n = 2$ and $b = 1$, since this is exactly equal to consensus for two processes, which has no solution [11]. On the other hand, it is easily solvable when $b \geq n$: p_i decides on its own proposal, for the i-th committee, $(i, V_i[i])$. In this paper we concentrate on the binary $(2, 3)$-CD problem, where the proposals are taken from the set $\mathcal{V} = \{0, 1\}$, and there are $b = 2$ committees, and $n = 3$ processes. We state our results for this fundamental case to simplify the presentation (avoiding more algebraic topology notation), and defer the most general phrasing to the full version. We prove that the $(2, 3)$-CD problem is equivalent to the *musical benches* problem of Gafni and Rajsbaum [10], and both are equivalent to $(2, 3)$-set agreement, closing an open question left there. Thus, all three problems are wait-free unsolvable in a read/write shared memory system, and they are all solvable if the system is enriched with objects capable of solving $(2, 3)$-set agreement (such as Test&Set).

Our paper is a follow up to [10], that introduced the musical benches problem, and showed the first connection between distributed computing and the Borsuk-Ulam theorem.[1] In the musical benches problem there are 3 processes, the first two, p_{-1}, p_1, wake up in the first bench (consensus instance), while a third one wakes up in the 2nd bench, either p_{-2} or p_2, but not both. In executions without conflict, namely when only one of p_{-1}, p_1 wakes up, each process decides its own index. Otherwise, the only requirement is that processes decide at most one index in $\{-1, 1\}$ and one index in $\{-2, 2\}$.

The musical benches problem tries to model a new distributed coordination difficulty: processes jump from bench to bench trying to find one in which they may be alone or not in conflict with one another. It resembles the consensus problem in the sense that at least two processes must agree on the value for one committee. However, it is not as clean a generalization as the CD is. Our first aim was to show that the

[1] Although we do not use it in this paper, the reader may be interested to know that the theorem is "one of the most useful tools offered by by elementary algebraic topology to the outside world"[18]. It implies Sperner's lemma, but not the opposite.

two problems are equivalent, but while investigating the CD problem, we found that both are equivalent to $(2,3)$-set agreement, while in [10] we only knew that musical benches is somewhere in between $(2,3)$-set agreement and read-write memory in terms of difficulty. We believe these equivalences are interesting, because although the problems are equivalent in the sense that one can be reduced to any other, they are not the same, a situation reminiscent of NP-complete problems. Having an arsenal of problems that we know are not solvable in read-write memory allows us to judge other problems unsolvable through reductions [9], rather than only through direct topological arguments. Indeed, distributed computing theory development has been promoted by the identification of problems that capture essential coordination difficulties.

The results in this paper are obtained through a novel reduction technique that combines distributed algorithmic ideas with topological intuition. The reduction technique consists of taking a read/write shared memory wait-free protocol, A, and identifying one or more executions, at the end of which an object solving some problem B is invoked. If the resulting protocol solves a problem C (for *any* object that implements a solution to problem B), we have shown that a solution to B implies a solution to C. Although reducing one problem to another is an old idea, our version here has some novel features that stem from the topological perspective of papers such as [14, 15, 16, 20]. We first consider the set of executions of A as a geometric object, called a *complex.* In the case of $n = 3$, each execution is drawn as a triangle, or *simplex*, where its corner vertices are labeled with the views (local states) of each one of the processes at the end of the execution. We then identify the triangles (or sometimes edges corresponding to 2-process executions) on which we are going to invoke the object B. Then we replace these triangles by the complex representing the set of possible responses of an implementation instance of B, and obtain the combined complex representing the protocol reduction. The goal is to obtain a protocol whose complex gives enough flexibility[2] to associate a decision function with each one of its vertices and solve the desired problem, C. See for example Figure 1, where we start with the simplex representing the inputs to the $(2,3)$-set agreement problem, we then execute a wait-free protocol where we identify two triangles to be removed and replaced by the set of possible responses of an arbitrary musical benches implementation, and the vertices of the resulting complex (obtained by gluing in the later complex into the hole of the former), can be colored with decisions (placed in the figure by each one of the vertices) that map into the $(2,3)$-set agreement outputs, represented by a hollow triangle. We have thus created a hole, which gives the desired flexibility to the final complex, and allows for an appropriate decision function to be designed. More details appear in Section 3.1, the corresponding Figure 6, and in Appendix A that includes more formal topology definitions and explanations about Figure 1. A good introduction to basic topology is [1].

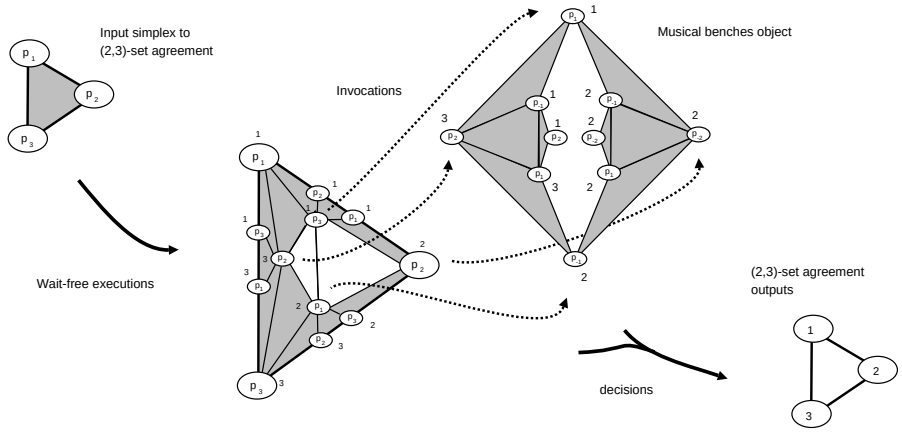


Figure 1: Solving $(2,3)$-set agreement using (one example of) a musical benches object.

The rest of the paper is organized as follows. Section 2 defines the problems of CD, set agreement and musical benches, and some additional preliminaries. Section 3 describes an algorithm to solve $(2,3)$-set agreement using a musical benches object, and an algorithm to solve $(2,3)$-set agreement using a CD object.

[2] The actual complex obtained depends on the actual solution to B used, but any such complex should exhibit that flexibility. Two features add flexibility: holes and more vertices.

Section 4 shows that the CD problem is wait-free solvable using a $(2,3)$-set agreement object. At the end of the paper there is an Appendix where proofs and additional details are provided.

2 Three Problems and Preliminaries

This paper considers the usual asynchronous shared memory model, composed of single-writer/multi-reader registers, and studies wait-free algorithms, where any number of processes can fail by crashing. A full description of these concepts can be found in textbooks such as [2, 17].

2.1 The Problems

The usual notion of *task* is 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. An *input vector* I specifies in its i-th entry, $I[i]$, the input value of process p_i, and we say p_i *proposes* $I[i]$ in the execution; similarly, an *output vector* J specifies a decision value $J[i]$ for each process p_i. The task defines a set of legal input vectors, and for each one, Δ specifies a set of legal output vectors. Thus, given input vector I, the processes decide a vector J such that individually p_i decides $J[i]$. It is sometimes convenient to consider *inputless tasks,* where a process has only one possible input value, namely its own id. See Appendix B.1 for more details.

2.1.1 Set Agreement

The *k-set agreement* problem is a generalization of consensus (see Appendix B.2) where processes must decide on at most k different values, out of the input values. The corresponding inputless version for three processes, p_1, p_2, p_3, and $k = 2$, denoted $(2,3)$-set agreement, is illustrated in Figure 2 (ids associated to each output value are omitted for clarity). It is defined by the set of input vectors consisting of (p_1, p_2, p_3) and all its subvectors, and the relation:

$$\begin{aligned} \Delta(p_i) &= \{(i)\} \\ \Delta(p_i, p_j) &= \{(i,i),(j,j),(i,j),(j,i)\}, \end{aligned}$$

and $\Delta(p_i, p_j, p_k)$ equal to all vectors of i, j, k with at most two different values (this requirement is represented in the figure by the hole; the possible outputs have no triangle, only edges and vertices). Set agreement is not

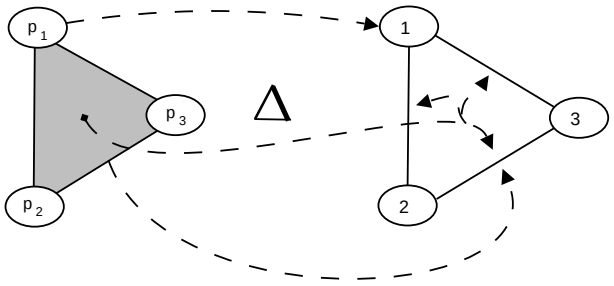


Figure 2: The inputless $(2,3)$-set agreement problem (some arrows of Δ omitted)

wait-free solvable [4, 16, 20], due to a generalization of the consensus impossibility connectivity argument to higher dimensions; wait-free executions induce a "flat structure" subdividing the input triangle, and in the figure one can see that a flat triangle is required to be mapped to a hollow one (preserving the boundary), which is impossible.

2.1.2 Committee Decision Problem

In the (b,n)-*committee decision* (CD) problem n processes are trying to solve b consensus instances, called *committees*, and each process is required to make a decision for at least one of them. More explicitly, in an execution, each process p_i proposes a vector V_i of b entries: $V_i[\ell]$ is the value proposed by p_i for committee ℓ.

A process decides a pair (ℓ, v) where ℓ, $1 \leq \ell \leq b$ denotes a committee, and v a value proposed by a process for committee ℓ. The problem is defined by the three requirements:

- **Termination** No process takes infinitely many steps without deciding.
- **Validity** If a process decides (ℓ, v) then $\exists\ j$ such that $v = V_j[\ell]$.
- **Agreement** Assume p_i, p_j decide (ℓ_i, v_i) and (ℓ_j, v_j) respectively. Then $\ell_i = \ell_j \Rightarrow v_i = v_j$.

We concentrate our attention on the binary $(2,3)$-CD problem, where $n = 3, b = 2$ and the proposed values are taken from $\mathcal{V} = \{0, 1\}$. We refer to this version as the *CD problem.*

2.1.3 Musical Benches

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 a process must return the place it wakes up in. Otherwise, in an execution where both participate, they return the same place. 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 any participating process can go to any seat. This is the *musical benches* problem of [10], shown there to have no wait-free solution. One feels an intuition as to why it should not be solvable, different from the set agreement impossibility: 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 wait-free solution, but surprisingly, this intuition is not exactly right: in [10] it is shown that the problem is reducible to $(2,3)$-set agreement, and hence a higher dimensional connectivity argument is needed.

The musical benches problem is illustrated in Figure 3, disregarding ids and omitting the dotted arrows of Δ for single vertices, to avoid cluttering the figure. In the figure there is also an example of an object implementing the musical benches problem. Each vertex is labeled on the inside with a process p_i, and on the outside with the value d returned from the object to p_i. The corner vertices correspond to executions where the process invokes the object alone, and therefore, a p_i vertex is labeled with value i. An edge joining two such vertices represents an execution where both processes invoke the object alone. Notice that there are two paths connecting the corners p_1, p_{-1}, with vertices labeled p_1 or p_{-1}, representing executions where only these processes invoke the object. For example, they are two edges incident to the p_1 corner, one representing an execution where the object returns 1 to p_{-1} and another where it returns -2 to p_{-1}. Executions where p_{-2} participates appear on the left side of the hole, while executions where p_2 participates appear on the right side of the hole. Notice also that no two vertices with the same id have the same value. One can check that this object indeed satisfies the musical benches specification given by Δ. See Appendix B.3 for more details.

2.2 Participating Set Problem

Preparing for the next section we recall the k-participating set problem [10], a generalization of the one in [5] that can access a set agreement object. We present here the case of 3 levels, and either $k = 2$, that has access to $(2,3)$-set agreement, or $k = 3$, the original problem of [5] that has no access to set agreement. That is, we have our first simple example of a reduction, in this case from the 2-participating set problem to $(2,3)$-set agreement. The 3-participating set problem shows that read write shared memory complex can be flattened to a subdivided simplex, as in the left side of Figure 4. Using a $(2,3)$-set agreement implementation, as in the right side of the figure, the center triangle is removed and we can create a subdivided simplex with a hole. A process p_i computes a set of ids S_i, such that

1. $\forall i : i \in S_i$,
2. $\forall i, j : S_i \subseteq S_j \vee S_j \subseteq S_i$,
3. $\forall i, j : i \in S_j \Rightarrow S_i \subseteq S_j$,
4. $|\{j : |S_j| = 3\}| \leq k$.

The first three are the requirements of the participating set problem in [5]. Sets satisfying these properties correspond to the subdivided simplex in Figure 4.

For completeness a protocol solving the k-participating set appears in Figure 5. The 4-th property is achieved through the set agreement object, invoked by p_i with the operation setAg(i), when $k = 2$. Invoking the set agreement operation has the effect of removing the simplex in the center of the subdivision (impossible

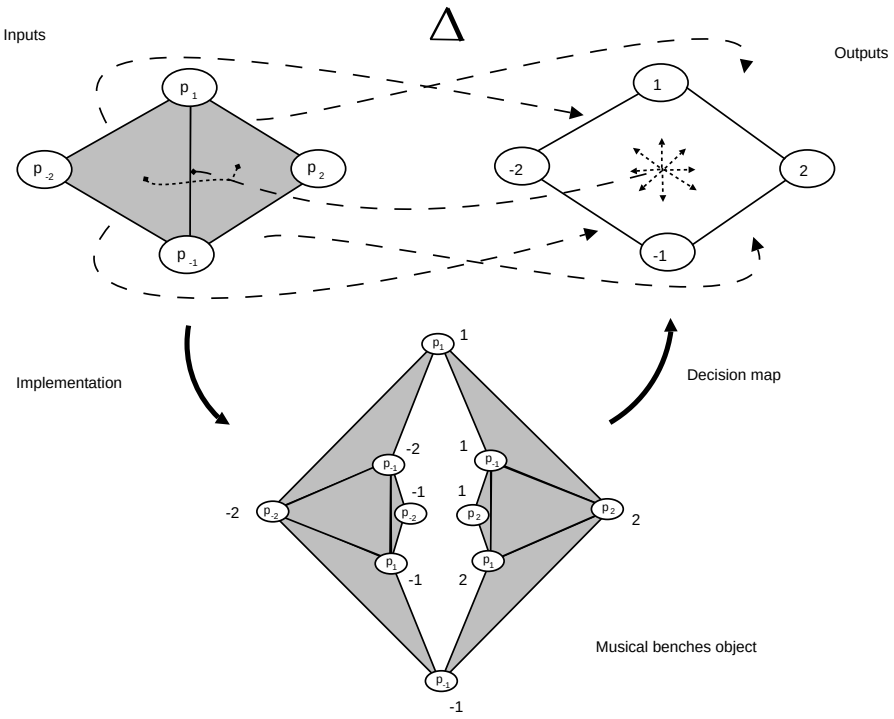


Figure 3: Musical benches task with a musical benches object

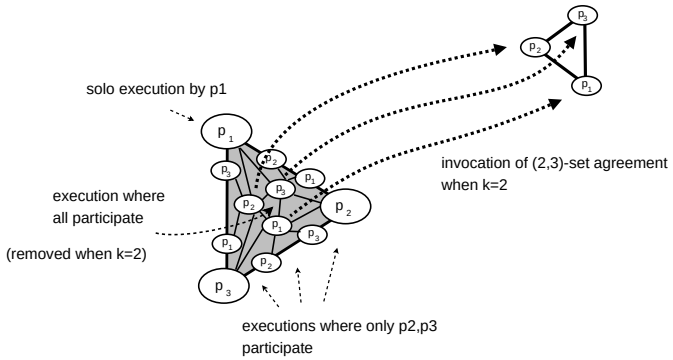


Figure 4: The k-participating set views for $k = 3$; when $k = 2$ the center triangle is removed.

Initially: $Level[j] = 4\ \forall j \in \{1,2,3\}$; $k = 2$ or $k = 3$;

Function k-PARTICIPATINGSET(i)
(01) **Init** $OK_i \leftarrow false$;
(02) **repeat** $Level[i] \leftarrow Level[i] - 1$;
(03) **for** $j = 1$ **to** 3 **do** $level_i[j] \leftarrow Level[j]$ **enddo**
(04) $S_i \leftarrow \{j : level_i[j] \leq Level[i], j \in \{1,2,3\}\}$;
(05) **if** $|S_i| = 3$ **and** $k = 2$ **then** $ans_i \leftarrow (2,3)$-SETAG(i);
(06) **if** $ans_i = i$ **then** $OK_i \leftarrow true$ **endif**
(07) **else** $OK_i \leftarrow true$ **endif**
(08) **until** $(|S_i| \geq Level_i[i]) \wedge OK_i$;
(09) return(S_i)

Figure 5: From $(2,3)$-set agreement to k-Participating set (code for p_i)

that the three processes produce sets of size 3), and leaving just its boundary (at most two processes may produce sets of size 3).

3 Solving $(2,3)$-Set Agreement

An algorithm to solve musical benches using $(2,3)$-set agreement is described in [10]. In Section 3.1 we describe an algorithm to solve $(2,3)$-set agreement using a musical benches object. Therefore, the musical benches problem is equivalent to $(2,3)$-set agreement. In Section 3.2 we describe an algorithm to solve $(2,3)$-set agreement using a CD object.

3.1 Solving $(2,3)$-Set Agreement with Musical Benches

Informally, the idea is very simple. In the musical benches one of two combinations of 3 processors start with 3 distinct inputs. They eventually halt with at most 2 distinct outputs. Thus the problem possess that "narrowing of choices" property that set agreement exhibit. The only problem we face is how to interface between the requirement of set agreement and those of musical benches. Resolving this is the crux of the paper: Employ read-write first and then glue the musical benches to replace two adjacent simplexes.

A protocol that solves $(2,3)$-set agreement using musical benches appears in Figure 7, and it is illustrated in Figure 6. Each process p_i starts by invoking the participating set protocol of Figure 5 with $k=3$. Once it gets back a set S_i, it invokes a musical benches protocol with a parameter $h_{mb}(i,S_i)$ defined as follows:

$$h_{mb}(i,S_i) = \begin{cases} -1 & \text{if } i=1 \text{ and } S_i=\{1,2,3\} \\ +1 & \text{if } i=3 \text{ and } S_i=\{1,2,3\} \\ +2 & \text{if } i=2 \text{ and } S_i=\{1,2,3\} \\ -2 & \text{if } i=2 \text{ and } S_i=\{2\} \\ - & \text{otherwise} \end{cases}$$

That is, the musical benches protocol is invoked only when $h_{mb}(i,S_i) \neq -$, and if so, each process p_i makes a decision, $f_{mb}(bench)$, that depends on the answer $bench$ returned by the musical benches protocol, as follows

$$f_{mb}(bench) = \begin{cases} 2 & \text{if } bench=-1 \\ 1 & \text{if } bench=1 \\ 3 & \text{if } bench=2 \\ 2 & \text{if } bench=-2 \end{cases}$$

or if p_i did not invoke the musical benches protocol, then it returns $g(i,S_i)$. The only requirement is that $g(i,S_i)$ returns an id in S_i, to satisfy the *validity* requirement of the set agreement problem (a decision was proposed by somebody).

Each vertex on the left of Figure 6 is labeled in the inside with the corresponding process p_i, and on the outside with its decision. The boundary of the removed triangles fits the boundary of the musical benches object. We stress that the object in the figure is just an example of one possible implementation of the musical benches problem; the protocol works for any implementation. Each of the vertices of the musical benches object is labeled in the inside with the corresponding process p_i, and on the outside with the value returned by the object. Thus, if we consider a vertex on the boundary of the hole (left side of the figure), say the corner p_2, it corresponds to an execution where p_2 runs solo, gets $S_2=\{2\}$ from the participating set object, invokes the musical benches with $h_{mb}(2,\{2\})=-2$, and gets back -2 (the label by the corresponding vertex on the right side of the figure) and decides $f_{mb}(-2)=2$ (the label by p_2's corner vertex on the left side of the figure). A p_i vertex of the left side of the figure where the musical benches object is not invoked is labeled with $g(i,S_i)$ (this particular g is just an example).

Lemma 1 *The* $(2,3)$-SetAg-from-Benches *protocol solves* $(2,3)$*-set agreement using any musical benches implementation.*

3.2 Solving $(2,3)$-Set Agreement with Committee Decision

The technique of Section 3.1 can be used to solve $(2,3)$-set agreement with CD. The SetAg-from-CD protocol of Figure 9 is similar to the one in Figure 7, except that a CD object is invoked instead of invoking a musical benches object, and the the functions h_{mb}, f_{mb} and g change.

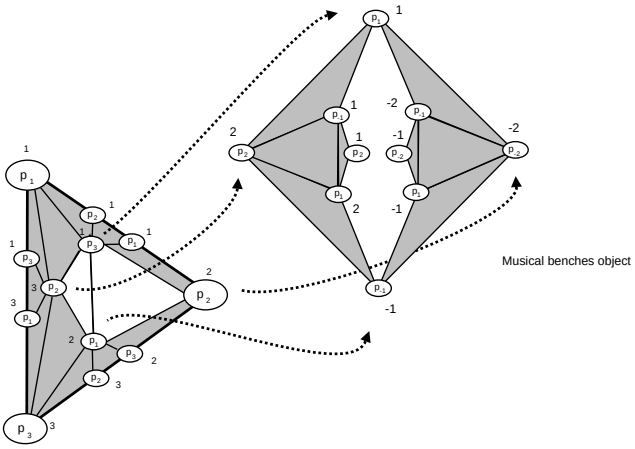


Figure 6: Solving $(2,3)$-set agreement using musical benches. The two triangles are removed and replaced by all the possible executions of a particular musical benches implementation.

Function $(2,3)$-SetAg-from-Benches(i)
(01) $S_i \leftarrow$ 3-ParticipatingSet(i);
(02) **if** $h_{mb}(i, S_i) \neq -$ **then**
(03) $\quad bench_i \leftarrow MusicalBenches(h_{mb}(i, S_i))$;
(04) $\quad$ return $f_{mb}(bench_i)$
(05) **else** return $g(i, S_i)$ **endif**

Figure 7: From Musical Benches to $(2,3)$-Set Agreement (code for p_i)

Each process p_i starts by invoking the participating set protocol of Figure 5 with $k = 3$. Once it gets back a set S_i, it checks if $h_{cd}(i, S_i) = -$. If so it decides according to the function $g_{cd}(i, S_i)$ (values by the vertices on the left side of Figure 8):

$$g_{cd}(i, S_i) = \begin{cases} i & \text{if } |S_i| = 1 \text{ else:} \\ 1 & \text{if } (i = 1 \text{ and } 2 \in S_i) \text{ or } (i = 2 \text{ and } 1 \in S_i) \text{ or } (i = 3 \text{ and } 1 \in S_i), \\ 2 & \text{if } i = 3 \text{ and } 2 \in S_i, \\ 3 & \text{otherwise.} \end{cases}$$

Else, $h_{cd}(i, S_i) \neq -$, and it invokes a CD protocol with the parameter $h_{cd}(i, S_i)$ defined as follows. This is illustrated in the right side of Figure 8, where an example of a CD object is presented (not all the object is depicted, only the values returned for the proposed input vectors).

$$h_{cd}(i, S_i) = \begin{cases} (-1, -2) & \text{if } i = 1 \text{ and } S_i = \{1, 2, 3\}, \\ (+1, +2) & \text{if } i = 3 \text{ and } S_i = \{1, 2, 3\}, \\ (-1, +2) & \text{if } i = 2 \text{ and } S_i = \{1, 2, 3\}, \\ (+1, -2) & \text{if } i = 2 \text{ and } S_i = \{2\}, \\ - & \text{otherwise.} \end{cases}$$

Once the CD object returns a value the process p_i stores it in a local variable *bench*. In the right side of Figure 8, the vectors proposed to the CD are depicted only in the 4 corners for lack of space; every vertex is labeled with the value returned by the object. Notice that no two vertices with the same id and proposed vectors have the same returned value associated (this is why the boundary can be subdivided here, but not

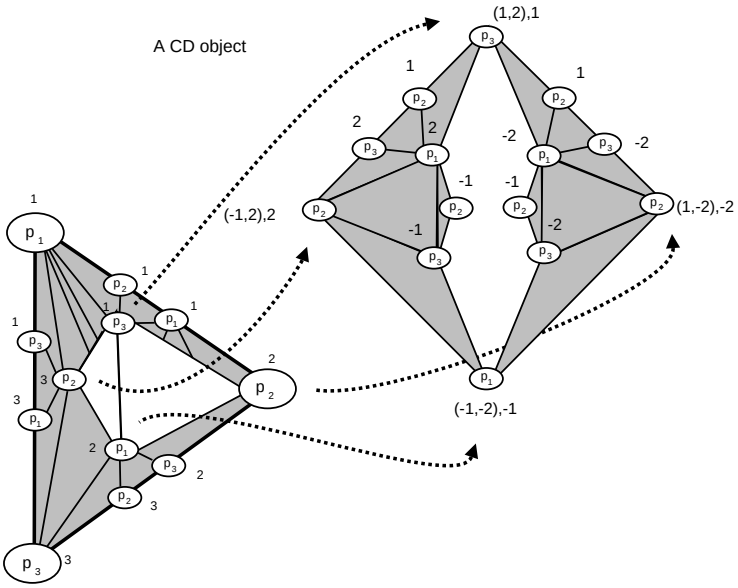


Figure 8: To solve set agreement each process p_i invokes a CD object. On the left figure, decisions are the values by the vertices; on the right figure values by the vertices are returned by the object.

in a musical benches object). The process then computes a decision $f_{cd}(i, bench)$, defined as follows:

$$f_{cd}(i, bench) = \begin{cases} 3 & \text{if } i = 1 \text{ and } bench = -1, \\ 2 & \text{if } i = 1 \text{ and } bench = -2, \\ 1 & \text{if } i = 1 \text{ and } bench = +1, \\ 1 & \text{if } i = 1 \text{ and } bench = +2, \\ 3 & \text{if } i = 2 \text{ and } bench = -1 \text{ or } bench = 2, \\ 2 & \text{if } i = 2 \text{ and } bench = 1 \text{ or } bench = -2, \\ 1 & \text{if } i = 3 \text{ and } bench = 1, \\ 1 & \text{if } i = 3 \text{ and } bench = 2, \\ 3 & \text{if } i = 3 \text{ and } bench = -1, \\ 2 & \text{if } i = 3 \text{ and } bench = -2. \end{cases}$$

```
Function (2,3)-SetAg-from-CD(i)
(01) S_i ← 3-ParticipatingSet(i);
(02) if h_cd(i, S_i) ≠ − then
(03)     bench_i ← CD(h_cd(i, S_i));
(04)     return f_cd(bench_i)
(05) else return g_cd(i, S_i) endif
```

Figure 9: From CD to $(2,3)$-Set Agreement (code for p_i)

Lemma 2 *The* $(2,3)$-SetAg-from-CD *protocol solves* $(2,3)$*-set agreement using any CD implementation.*

4 Solving Committee Decision with $(2,3)$-Set Agreement

This section shows that the $(2,3)$-CD problem is wait-free solvable using a $(2,3)$-set agreement object. Since in Section 3.2 we showed the opposite reduction, we have that both problems are equivalent. The wait-free impossibility of solving $(2,3)$-set agreement [4, 16, 20] implies that $(2,3)$-CD is wait-free unsolvable.

```
Function (2,3)-CD-FROM-SETAG(V_i)
Init  view_i ← ∅; id_i ← [−,−,−];
(01)  Prop[i] ← V_i;
(02)  S_i ← 2-PARTICIPATINGSET(i);
(03)  if |S_i| = 3 then id[i] ← i;
(04)                    for j = 1 to 3 do id_i[j] ← id[j] enddo
(05)                    view_i ← {j : id_i[j] ≠ −, j ∈ {1,2,3}}
(06)  endif
(07)  return f(S_i, view_i)
```

Figure 10: From $(2,3)$-set agreement to $(2,3)$-CD (code for p_i)

In [10] a protocol that solved the musical benches problem with access to a $(2,3)$-set agreement object is described. This protocol can be adapted to solve the CD problem; the main difference is the decision function. The protocol works as follows. Each process p_i gets a vector V_i as input to the CD problem. It first writes it to a shared array, *Prop*, in position *Prop*$[i]$. Then p_i invokes the 2-PARTICIPATINGSET(i) function of Figure 5, and gets back a set S_i of process ids, satisfying the 2-PARTICIPATINGSET properties:

1. $\forall i : i \in S_i$,
2. $\forall i,j : S_i \subseteq S_j \vee S_j \subseteq S_i$
3. $\forall i,j : i \in S_j \Rightarrow S_i \subseteq S_j$
4. $|\{j : |S_j| = 3\}| \leq 2$

Note that it follows from these properties that there is at most one index i such that $|S_i| = 1$ and, at most two indices i,j such that $|S_i| = |S_j| = 2$. There are also at most two indices such that i,j such that $|S_i| = |S_j| = 3$. Notice also that if $j \in S_i$ then, p_j participates in the protocol. Once p_i gets a set S_i back from the 2-PARTICIPATINGSET object, if $|S_i| = 3$ it executes lines (03)–(05) which have the effect of proposing its *id* to a read/write object, and gets back a set $view_i$ of ids, of processes that invoked the object. This is seen in Figure 13 as subdividing the boundary of the removed center triangle (the read/write object's complex consists of a 3-edge path: in the middle edge both processes see each other, while in the 2 end edges exactly one sees the other). Finally, process p_i decides a value $f(S_i, view_i)$. Due to space limitation, the definition of the decision function f is given in the appendix (figure 12).

Lemma 3 *The* $(2,3)$-CD-FROM-SETAG *protocol solves* $(2,3)$*-CD using any* $(2,3)$*-set agreement object.*

As a consequence of Lemmas 1, 2, and 3 we have our main result.

Theorem 1 *Musical benches can be wait-free solved iff CD can be wait-free solved iff* $(2,3)$*-set agreement can be wait-free solved.*

References

[1] Armstrong M.A., *Basic Topology,* Springer-Verlag, 251 pages, 1983.

[2] Attiya H. and Welch J., *Distributed Computing: Fundamentals, Simulations and Advanced Topics,* McGraw–Hill, 451 pages, 1998.

[3] Bar-Noy A., Deng X., Garay J., Kameda T., Optimal amortized distributed consensus. *Info. and Comp.,* 120(1):93-100, 1995. Prel. version in WDAG'91.

[4] 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, 1993.

[5] 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, 1993.

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

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

[8] 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.

[9] Gafni E. DISC/GODEL presentation: R/W Reductions (DISC'04), 2004. http://www.cs.ucla.edu/~ eli/eli/godel.ppt

[10] Gafni E. and Rajsbaum S., Musical Benches. *Proc. 19th Int. Symposium on Distributed Computing (DISC'05)*, Springer Verlag LNCS *To Appear*, Cracow (Poland), September 2005.

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

[12] Herlihy M., Rajsbaum S., New Perspectives in Distributed Computing. *Proc. 24th International Symposium Mathematical Foundations of Computer Science (MFCS),* Springer Verlag LNCS #1672, pp. 170–186, 1999.

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

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

[15] 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),* 2004.

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

[17] Lynch N., Distributed Algorithms. *Morgan Kaufmann Pub.*, San Francisco (CA), 872 pages, 1996.

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

[19] Lamport L., The Part-Time Parliament. *ACM Transactions On Computer Systems*, 16(2):133-169, 1998.

[20] 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 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.

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 the left side of Figure 4 is a subdivision of a 2-dimensional simplex.

Consider Figure 1 representing the execution of the protocol of Figure 7. We start with a simplex labeled with the three process ids p_1, p_2, p_3 representig the input to the $(2,3)$-set agreement problem (p_i's input is i). The corners correspond to executions where a process runs solo; the edges to executions where two processes participate; and the solid triangle to executions where the three of them participate. After the read/write wait-free 3-ParticipatingSet protocol of Figure 5 is executed, we get the next figure, where all possible executions of this protocol are represented. Each vertex has associated an id p_i (drawn in the inside of the vertex), a view S_i returned by the protocol (not drawn in the figure), and a decision value (drawn on the side of the vertex). The corners represent solo executions (p_i gets back $S_i = \{i\}$ from the ParticipatingSet protocol); the vertices along the boundary represent executions where only two processes participate, and see each other (p_i, p_j get back $S_i = \{i, j\}$ from the ParticipatingSet protocol); the triangle in the center represents the execution where all three processes see each other (and they get back $S_i = \{1, 2, 3\}$ from the ParticipatingSet protocol). A process p_i that gets back a set $S_i = \{1, 2, 3\}$ invokes the muscial benches object, and also p_2 invokes it if it gets back the set $S_2 = \{2\}$. Therefore, the center triangle and its adjacent triangle with a corner to p_2's solo execution will be replaced by all possible executions of the object. In the figure are depicted a set of such executions, of an object implementing the musical benches problem. But we stress that other implementations of the musical benches problem are possible and would produce a different complex (e.g. removing the two middle vertices labeled p_2, p_{-2} would also be an implementation of the musical benches problem). The decisions made by the processes define a simplicial map that goes to the hollow (because never are three different values decided) triangle at the right of the figure.

B More Details about Problems

B.1 Inputless Tasks

It is sometimes convenient to consider *inputless tasks,* where a process has only one possible input value, namely its own id. In this case, the difficulty of solving the task does not come from the uncertainty that other processes have on what is the local values of each other, but rather on what are the processes that are participating (i.e., taking steps) in an execution. Consider an algorithm for some set of processes 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 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 $-$ if the process did not decide. The algorithm *solves* the task if in every execution with input vector I, the output vector O can be extended (by replacing $-$ entries with other values) to a vector in $\Delta(I)$, and a process that does not fail decides.

It turns out that both notions of tasks are equivalent. An inputless task is a task with just one possible input configuration. Also, given a task, one can define an equivalent inputless task by introducing more processes. Namely, if there are x possible input values for a process p, then we introduce x copies of process p, and consider only executions where the processes corresponding to different original processes participate. Although both notions are equivalent, inputless tasks are convenient because one can concentrate on particular combinations of input values that make a problem difficult to solve.

B.2 Consensus

In the *consensus* problem each process starts with a local input value out of some set of possible input values, $\mathcal{V}$, and must decide on an output value such that the following *agreement* and *validity* requirements are satisfied: (i) if v_1 and v_2 are the decision values of two processes in an execution, then $v_1 = v_2$, and (ii) if v is the decision value of a process, then v is the input value of some process in the execution. In the binary consensus problem $\mathcal{V} = \{0, 1\}$.

We will consider also an inputless version of consensus: a process must decide on the id of a participating process. The binary consensus case for two process, p_{-1}, p_1, illustrated in Figure 11(a), is defined formally by the set of input vectors $\{(p_{-1}, p_1), (p_1), (p_{-1})\}$, and the relation:

$$\begin{aligned} \Delta(p_{-1}, p_1) &= \{(-1, -1), (1, 1)\}, \\ \Delta(p_{-1}) &= \{(-1)\}, \\ \Delta(p_1) &= \{(1)\}. \end{aligned}$$

It is sometimes convenient to consider only the output values, and disregard the processes ids, as depicted

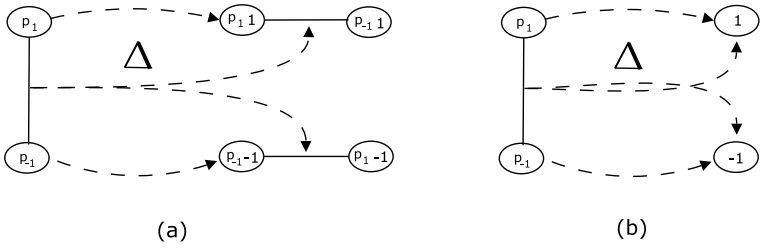


Figure 11: The inputless consensus problem

in Figure 11(b). It is well-known that consensus is wait-free unsolvable [8, 11]. The reason is that wait-free executions preserve the connectivity of the input configurations. In the figure we can observe that while the input configurations are connected, the output configurations are not, and Δ requires to send different regions of a connected object to two disconnected output configurations.

B.3 Musical Benches

The *musical benches problem* of size b is 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 $\{-i, i, - | 1 \leq i \leq b\}$. In this paper we consider the case of $b = 2$. Formally, Δ is:

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

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 that it includes the first bench, and a restriction of the 2nd bench that disallows p_{-2} and p_2 participating together.

C Decision Function of the Protocol CD-from-SetAg

The decision function is described in Figure 12.

$(level_i)$	id.	S_i	$view_i$	$f(S_i, view_i)$
1	1,2	$\{1\}, \{2\}$	$\emptyset$	(i, i)
1	3	$\{3\}$	$\emptyset$	$(2, 3)$
2	1	$\{1, 2\}$	$\emptyset$	$(2, 2)$
2	1	$\{1, 3\}$	$\emptyset$	$(2, 3)$
2	2	$\{1, 2\}$	$\emptyset$	$(1, 1)$
2	2	$\{2, 3\}$	$\emptyset$	$(1, 3)$
2	3	$\{1, 3\}$	$\emptyset$	$(1, 1)$
2	3	$\{2, 3\}$	$\emptyset$	$(2, 2)$
3	1	$\{1, 2, 3\}$	$\{1\}$	$(1, 3)$
3	1	""	$\{1, 2\}$	$(2, 3)$
3	1	""	$\{1, 3\}$	$(1, 3)$
3	2	""	$\{2\}$	$(2, 3)$
3	2	""	$\{1, 2\}$	$(2, 3)$
3	2	""	$\{2, 3\}$	$(1, 1)$
3	3	""	$\{3\}$	$(2, 2)$
3	3	""	$\{1, 3\}$	$(1, 3)$
3	3	""	$\{2, 3\}$	$(1, 1)$

Figure 12: Decision function of the $(2, 3)$-CD-FROM-SETAG protocol

D Proofs

Lemma 1 The $(2, 3)$-SETAG-FROM-BENCHES protocol solves $(2, 3)$-set agreement using any musical benches implementation.

Proof We need to check that in any execution at most 2 different ids are decided. On executions where the musical benches object is not invoked this can be checked directly from Figure 6, noticing that along the boundary decisions are consistent. For example, consider the boundary of the removed triangles connecting p_2 and p_3 to the corner (solo execution) vertex p_1. In any vertex here p_2 decides 3, because any musical benches implementation must return $+2$ (p_2, p_1 are in no conflict), and the map $f_{mb}(2)$ returns 3. In general, along the boundary, the musical benches object returns to a process that invokes it with id i the same id i, and the map f_{mb} transform it into a value j such that at most 2 different values are returned by processes that did not invoke the object, together with the processes that did invoke it.

The other case that needs to be checked is for executions where the three processes invoke the musical benches object. In this case, assume for contradiction that in one execution 3 different values are decided. This is impossible because by the definition of f_{mb} this implies that the musical benches object returned either $-1, +1$, or $-2, +2$. $\square_{Lemma\ 1}$

Lemma 2 The $(2, 3)$-SETAG-FROM-CD protocol solves $(2, 3)$-set agreement using any CD implementation.

Proof We need to check that in any execution at most 2 different ids are decided. We first check it for executions where at least one process does not invoke the CD object, namely, when it decides in line (05). Assume for contradiction that 3 values are decided. When we consider the possible cases below, we use the following notation. For $h_{cd}(i, S_i) \neq -$, we use $p_i, (a, b) : j \rightarrow k$ to denote p_i with view S_i has $h_{cd}(i, S_i) = (a, b)$, gets back from the CD the value j and decides k because $f_{cd}(i, j) = k$. For $h_{cd}(i, S_i) = -$, we use simply $g_{cd}(i, S_i) \rightarrow k$. We start with the cases where at least one process does not invoke the CD object:

1. A process on the boundary decides 1 (i.e., $g_{cd}(i, S_i) \rightarrow 1$).
   (a) Assume $i = 1$ with view S_1. If $S_1 = \{1\}$ then $h_{cd}(2, S_2) \neq -$ and $h_{cd}(3, S_3) \neq -$. The only process that can decide 2 in this situation is p_2 with $p_2, (-1, 2) : 1 \rightarrow 2$, but then for p_3 to decide 3 it must get back -1 from the CD, i.e., $p_3, (1, 2) : -1 \rightarrow 3$ which is impossible, because the CD cannot return $-1, 1$.

Otherwise, $S_1 = \{1,2\}$, and then $h_{cd}(2,S_2) \neq -$ and $h_{cd}(3,S_3) \neq -$. If process p_2 decides 2 in this situation then $p_2, (1,-2) : 1 \vee -2 \rightarrow 2$, but then for p_3 to decide 3 it must get back -1 from the CD, which is impossible because neither p_2 nor p_3 proposed -1 (and p_1 does not propose to the CD).

(b) Assume $i = 2$, then the only view is $S_2 = \{1,2\}$. This case is trivial because the two neighbors of p_2 decide 1.

(c) Finally, assume $i = 3$ with view $S_3 = \{1,3\}$. This case is again trivial because no neighbor of p_3 decides 2.

2. A process on the boundary decides 2 (i.e., $g_{cd}(i,S_i) \rightarrow 2$).
   (a) Assume $i = 2$ with view S_2. Then $S_2 = \{2\}$ and $h_{cd}(2,S_2) \neq -$. The neighbor p_3 with $S_3 = \{2,3\}$ also decides 2, so this case is trivial. Consider then process p_1 with $S_1 = \{1,2\}$, that decides 1. In this situation $p_3, (1,2) : 1 \vee 2 \rightarrow 1 \vee 2$, and then nobody decides 3.
   (b) Assume $i = 3$ with view S_3. Then $S_3 = \{2,3\}$. The only process that can decide 1 in this situation is p_1 with $p_1, (-1,-2) : 1 \rightarrow 1$ (it cannot get back 2 because nobody proposes 2 to the CD). But then for p_2 to decide 3 it must get back -1 from the CD (cannot get back 2), which is impossible because the CD cannot return $-1, 1$.

3. A process on the boundary decides 3 (i.e., $g_{cd}(i,S_i) \rightarrow 3$).
   (a) Assume $i = 2$ with view S_2. Then $S_2 = \{2,3\}$ and this case is trivial because its neighbors do not decide 1.
   (b) Assume $i = 1$ with view S_1. Then $S_1 = \{1,3\}$ and this case is trivial because its neighbors do not decide 2.

We now consider the cases where all processes invoke the CD object. We consider two cases, according to which process decides 1 (p_2 never decides 1):

1. Process p_1 decides 1. We have two cases depending on why it decides 1:
   (a) $p_1, (-1,-2) : 1 \rightarrow 1$. Then p_2 or p_3 propose 1 to the CD. We analyze who decides 3. First notice that p_3 cannot decide 3 because it would have to get -1 from the CD. Hence, p_2 decides 3: $p_2, (1,-2) : 2 \rightarrow 3$, because it cannot get back -1. Then p_3 has to decide 2, but the only way is if it gets -2, a contradiction because the CD returns $-2, 2$.
   (b) $p_1, (-1,-2) : 2 \rightarrow 1$. Then p_2 or p_3 propose 2 to the CD. We analyze who decides 2. First notice that p_3 cannot decide 2 because it would have to get -2 from the CD. Hence, p_2 decides 2: $p_2, (-1,2) : 1 \rightarrow 2$, because it cannot get back -2. Then p_3 has to decide 3, but the only way is if it gets -1, a contradiction because the CD returns $-1, 1$.

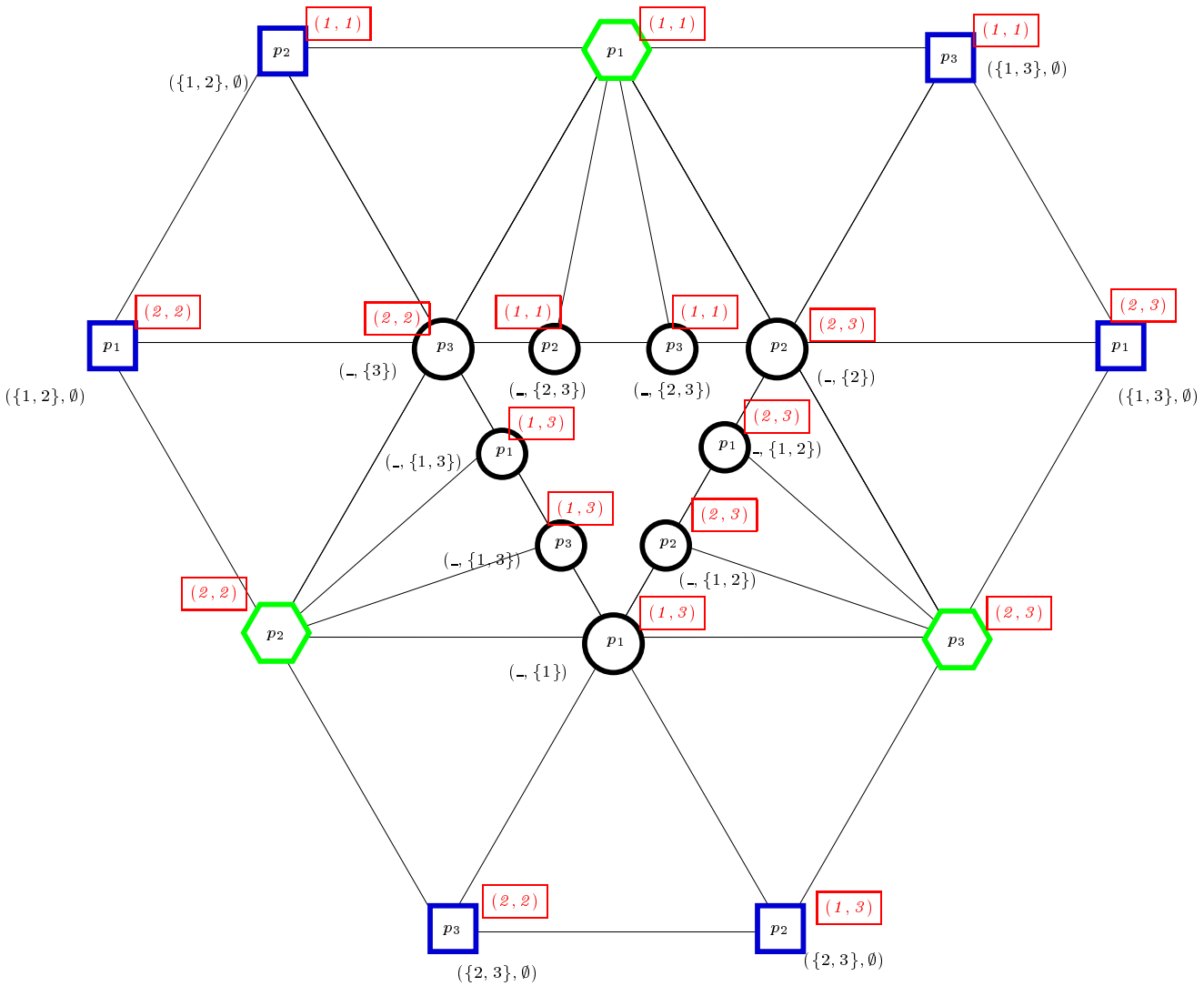
2. Process p_3 decides 1. We have two cases depending on why it decides 1:
   (a) $p_3, (1,2) : 1 \rightarrow 1$. If p_2 decides 2, it is because it got back 1 or -2. Then p_1 cannot decide 3 because it would have to get back -1. Thus, assume the one deciding 2 is p_1. This is because it got back -2. But then p_2 cannot decide 3 because it would have to get back -1 or 2, and the CD would return either $-1, 1$ or $-2, 2$.
   (b) $p_3, (1,2) : 2 \rightarrow 1$. If p_2 decides 2, it is because it got back 1 (it cannot get back -2). Then p_1 cannot decide 3 because the CD would return $-1, 1$. Thus, assume the one deciding 2 is p_1. This is because it got back -2, and the CD would return $-2, 2$.

□$_{Lemma\ 2}$

Lemma 3 The $(2,3)$-CD-FROM-SETAG protocol solves $(2,3)$-CD using any $(2,3)$-set agreement object.

Proof The proof of the protocol follows from the figure 13 which represents the final views of the processes with their decision. □$_{Lemma\ 3}$

E The CD-from-SetAg views



p_i process at level 1 ($|S_i| = 1$, i.e., $S_i = \{i\}$)

p_i process at level 2 ($|S_i| = 2$)

p_i process at level 3 ($|S_i| = 3$) $(\{1,3\},\_)$: final state $(S_i, view_i)$.

$(2,1)$: decision (p_i decides in committee 2 the value proposed for committee 2 by p_1)

Figure 13: The CD-FROM-SETAG views.