All of us are smarter than any of us: more on the robustness of the Consensus hierarchy (Part II)

Wai-Kau Lo* Vassos Hadzilacos†
wklo@cs.utoronto.ca vassos@cs.utoronto.ca

Technical Report CSRI-351

October 8, 1996

Abstract

A *wait-free hierarchy* classifies shared object types according to their strength in supporting wait-free implementations of other types. Such a hierarchy is *robust* if it is not possible to implement types the hierarchy classifies as strong by using only types that it classifies as weak. The Consensus hierarchy is the most closely studied wait-free hierarchy. In [3] we proved that the Consensus hierarchy is not robust if nondeterministic types are allowed. This naturally raises the question whether some other wait-free hierarchy is robust. In this paper we show that, if nondeterministic types are allowed, the only robust wait-free hierarchy is the trivial one, which lumps all types into a single level.

1 Introduction

This paper is a continuation of [3] (henceforth referred to as "Part I"). We assume that the reader is already familiar with that paper where, among other things, we provide the necessary background, describe related research, and define the basic terminology and the underlying model of computation.

In Part I we showed that, in the context of nondeterministic object types, the Consensus hierarchy is not robust. To do that, we exhibited two nondeterministic types, named negation and booster, satisfying the following two properties:

*Supported by a Canadian Commonwealth Scholarship.

†Supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

1. There is no algorithm for Consensus between two processes using either only registers and objects of type negation, or only registers and objects of type booster.

2. Using only registers and objects of types negation and booster, we can solve Consensus between two processes.

Taken together, these two properties imply that the Consensus hierarchy is not robust.

Jayanti [2] defined a hierarchy of types to be a *wait-free hierarchy* if any type assigned to level n, in conjunction with registers, can be used to implement *any* type in a system of n or fewer processes. The Consensus hierarchy is the special case of a wait-free hierarchy where a type assigned to level n, in conjunction with registers, can be used to implement any type in a system of n, but not $n+1$, processes. Jayanti proved that if a robust wait-free hierarchy exists, it is the Consensus hierarchy or a coarsening of it — i.e., a hierarchy obtained by merging consecutive levels of the Consensus hierarchy into a single level.

Even though our result in Part I shows that the Consensus hierarchy is not robust, it does not settle the more general question of whether a *nontrivial* robust wait-free hierarchy exists. (A hierarchy is nontrivial if it partitions the set of types into at least two levels.) In particular, it is consistent with our result that the nontrivial hierarchy obtained by merging levels one and two of the Consensus hierarchy is robust. In this paper we show that, in the context of nondeterministic object types, there is no nontrivial wait-free hierarchy that is robust. Specifically, we introduce two nondeterministic object types, named $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$, each of which is at level one of the Consensus hierarchy (i.e., neither can be used to solve Consensus between two processes) but which taken together are strong enough to solve Consensus for *any* number of processes.

By the universality of Consensus (cf. [1]), this implies that no robust coarsening of the Consensus hierarchy can assign a type to a level higher than that of both $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$. Since both $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$ are at level one of the Consensus hierarchy (and hence, of any coarsening of it), it follows that any robust coarsening of the Consensus hierarchy must place all types into level one. In other words, any robust coarsening of the Consensus hierarchy must be trivial. In conjunction with Jayanti's above-cited result this implies that, in the context of nondeterministic types, there is no nontrivial robust wait-free hierarchy of types.

The rest of this paper is organised as follows. In Section 2 we define the Equality Negation problem for n processes; this is a generalisation of the Equality Negation problem for two processes that was introduced in Part I. In Section 3 we define the types $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$. In Section 4 we give an algorithm that uses only registers and objects of types $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$, and solves Consensus for an arbitrary number of processes. In Sections 5 and 6, we prove that $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$ have Consensus number one.

2 The Equality Negation problem for n processes

In this section we define the Equality Negation problem for n processes, $P_0, \ldots, P_{n-1}$, where $n \geq 2$. This problem generalises Equality Negation for two processes, defined in Section 3.2 of Part I, by replacing one of the two processes with a group $\mathcal{P}_1 = \{P_1, \ldots, P_{n-1}\}$ of $n-1$ processes. Each process has an initial value drawn from the set $\{0, 1, 2\}$. Additionally, we assume that all processes in $\mathcal{P}_1$ have the same initial value; if not, the problem imposes no constraints whatsoever on the behaviour of processes. Every process must eventually decide irrevocably either 0 or 1, so that no two processes in $\mathcal{P}_1$ decide different values; and if P_0 and any process in $\mathcal{P}_1$ decide, then their decisions are the same if and only if their initial values are different. An Equality Negation algorithm for processes $P_0, \ldots, P_{n-1}$ must therefore satisfy the following three properties, starting from any initial configuration where all processes in $\mathcal{P}_1$ have the same initial value:

Termination: If a process reaches a final state, then it must have decided 0 or 1.

Agreement: If any two processes in $\mathcal{P}_1$ decide, their decisions are the same.

Negation: If process P_0 and any process in $\mathcal{P}_1$ decide, their decisions are the same *if and only if* their initial values are different.

Note that, when $n = 2$, Agreement is trivially satisfied, and the above definition is equivalent to the one given in Section 3.2 of Part I.

3 Specification of types $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$

The specification of $\mathsf{negation}_\infty$ appears in Figure 1.[1] This type supports only one kind of operation, called NEGATE, with two parameters $i \in \{0, 1\}$ and $v \in \{0, 1, 2\}$. (The NEGATE operation of type $\mathsf{negation}$ in Section 5 of Part I has only one parameter $i \in \{0, 1\}$.) We shall refer to i as the *process* parameter and v as the *initial-value* parameter. A $\mathsf{negation}_\infty$ object will become "upset" if and only if two NEGATE operations with process parameter 0 are applied, or if two NEGATE operations with process parameter 1 but different initial-value parameters are applied. If it is upset, the $\mathsf{negation}_\infty$ object will return arbitrarily either 0 or 1 to the invoking process, and will remain upset forever. As long as the $\mathsf{negation}_\infty$ object is not upset, it responds to invocations with values that satisfy the following: All NEGATE invocations with process parameter 1 receive the same response; any two NEGATE invocations with different process parameters receive the same response if and only if their initial-value parameters are different. Thus, an object O of type $\mathsf{negation}_\infty$ can be used to solve Equality Negation for processes $P_0, \ldots, P_{n-1}$ in a direct way: P_0 applies NEGATE$(0, v)$ to O, where v is P_0's initial value; any process in

[1] The conventions used in the pseudocode that describes the state transition relation are the same as those used in Part I; see the second paragraph of Section 5 in Part I.

$\{P_1, \ldots, P_{n-1}\}$ applies NEGATE$(1, v')$ to O, where v' is the process' initial value; each process decides the value returned by object O.

To achieve this behaviour, type $\mathsf{negation}_\infty$ has two distinguished states: the *fresh* state, denoted ☺; and the *upset* state, denoted ☹. The remaining states are of the form $\langle v_0, v_1, u_0, u_1 \rangle$, where $v_0, v_1 \in \{\perp, 0, 1, 2\}$ and $u_0, u_1 \in \{\perp, 0, 1\}$. The intended meaning of state $\langle v_0, v_1, u_0, u_1 \rangle$, assuming the object is initialised to the fresh state, is as follows: v_0 is the initial-value parameter of the first NEGATE operation with process parameter 0; u_0 is the value returned to that operation; v_1 and u_1 have corresponding interpretations for the first NEGATE operation with process parameter 1. By maintaining this information, the object knows all it needs in order to respond appropriately to the NEGATE operations applied to it.

The specification of type $\mathsf{booster}_\infty$ is shown in Figure 2. This type supports two kinds of operations: ENROLL and REVEAL. An ENROLL operation requires a parameter $i \in \{0, 1\}$ and returns a value in $\{0, 1, 2\}$. A REVEAL operation requires three parameters $i \in \{0, 1\}$, $v \in \{0, 1, 2\}$ and $u \in \{0, 1\}$, and returns a value in $\{0, 1\}$; we shall refer to these three parameters as the *process*, *challenge initial-value*, and *decision* parameters, respectively. The intended use of a $\mathsf{booster}_\infty$ object by processes $P_0, \ldots, P_{n-1}$ is as follows: The first thing process P_0 does to a $\mathsf{booster}_\infty$ object is to apply ENROLL(0) to "enroll" itself in the object. Similarly, the first thing any process in $\mathcal{P}_1 = \{P_1, \ldots, P_{n-1}\}$ does to a $\mathsf{booster}_\infty$ object is to apply ENROLL(1) to enroll its group in the object. The object maintains a secret: whether P_0 or some process in the group $\mathcal{P}_1$ was the first process that enrolled to the object. After a process has enrolled, it can ask the object for this secret by applying a REVEAL operation. The object will reveal the secret to a process only if the process can "prove" to the object that it can solve Equality Negation. We describe how this works in more detail below.

In response to the ENROLL operation it applies, a process receives a value in $\{0, 1, 2\}$ from the $\mathsf{booster}_\infty$ object. This value is a "challenge" given by the object with which the process is required to solve Equality Negation. More specifically, process P_0 uses the challenge, say v_0, received from the $\mathsf{booster}_\infty$ object as its initial value to solve Equality Negation. The different processes in $\mathcal{P}_1$, however, may receive different challenges from the $\mathsf{booster}_\infty$ object. Thus, before embarking upon solving Equality Negation, processes in $\mathcal{P}_1$ must first choose a unique initial value, say v_1, from the set of challenge initial values they received. After solving Equality Negation and deciding a value $u \in \{0, 1\}$, process P_0 (respectively, a process in $\mathcal{P}_1$) applies the operation REVEAL$(0, v_0, u)$ (respectively, REVEAL$(1, v_1, u)$) to the $\mathsf{booster}_\infty$ object to obtain the secret. If the decision parameter of a REVEAL operation is a legitimate decision for the challenge initial value(s) returned by the ENROLL operation(s) applied to the object so far, the object will respond to the REVEAL operation by returning the secret: whether P_0 or some process in $\mathcal{P}_1$ was the first process that enrolled to the object. If not, the object returns an arbitrary answer.

Thus, there is an etiquette that a process must follow in accessing a $\mathsf{booster}_\infty$ object: First enroll and obtain a challenge initial value, then get the secret by presenting to the object a decision for Equality Negation that is legitimate with respect to the challenge

Let $\mathcal{A}$ be the set of four-tuples of the form $\langle v_0, v_1, u_0, u_1 \rangle$, where $v_0, v_1 \in \{\perp, 0, 1, 2\}$, $u_0, u_1 \in \{\perp, 0, 1\}$, and the following conditions apply: $v_0 = \perp$ if and only if $u_0 = \perp$; $v_1 = \perp$ if and only if $u_1 = \perp$; and at least one of v_0, v_1 is not $\perp$.

Set of states: {☺,☹} $\cup \mathcal{A}$.

Set of operations: $\{$ NEGATE$(i, v) : i \in \{0, 1\}, v \in \{0, 1, 2\} \}$.

Set of responses: $\{0, 1\}$.

NEGATE(i, v), $i \in \{0, 1\}, v \in \{0, 1, 2\}$, return either 0 or 1
  case *current_state*
    ☹ : **return** Choose($\{0, 1\}$)
    ☺ : $u \leftarrow$ Choose($\{0, 1\}$)
      if $i = 0$ **then** *current_state* $\leftarrow \langle v, \perp, u, \perp \rangle$
      else *current_state* $\leftarrow \langle \perp, v, \perp, u \rangle$
      return u
    $\langle v_0, v_1, u_0, u_1 \rangle$:
      if $i = 0$ **then**
        if $v_0 \neq \perp$ **then** *current_state* $\leftarrow$ ☹; **return** Choose($\{0, 1\}$)
        else
          if $v \neq v_1$ **then** $u \leftarrow u_1$
          else $u \leftarrow \overline{u}_1$
          current_state $\leftarrow \langle v, v_1, u, u_1 \rangle$; **return** u
      else (* $i = 1$ *)
        if $v_1 \neq \perp$ **then**
          if $v \neq v_1$ **then** *current_state* $\leftarrow$ ☹; **return** Choose($\{0, 1\}$)
          else return u_1
        else (* $v_1 = \perp$ and thus $v_0, u_0 \neq \perp$ *)
          if $v \neq v_0$ **then** $u \leftarrow u_0$
          else $u \leftarrow \overline{u}_0$
          current_state $\leftarrow \langle v_0, v, u_0, u \rangle$; **return** u

Figure 1: Specification of type $\mathsf{negation}_\infty$.

Let $\mathcal{A}$ be the set of six-tuples of the form $\langle V, v_0, v_1, u_0, u_1, d\rangle$, where $V \subseteq \{0,1,2\}$, $v_0, v_1 \in \{\perp, 0, 1, 2\}$, $u_0, u_1 \in \{\perp, 0, 1\}$, $d \in \{0,1\}$, and the following conditions apply: if $v_0 = \perp$ then $u_0 = \perp$ and $d = 1$; $v_1 = \perp$ if and only if $u_1 = \perp$; if $v_1 \neq \perp$ then $v_1 \in V$; if $d = 0$ then $v_0 \neq \perp$; and if $d = 1$ then $V \neq \emptyset$.

Set of states: $\{$☺,☹$\} \cup \mathcal{A}$.
Set of operations: $\{\,\text{ENROLL}(i) : i \in \{0,1\}\,\} \cup \{\,\text{REVEAL}(i, v, u) : i, u \in \{0,1\}, v \in \{0,1,2\}\,\}$.
Set of responses: $\{0,1,2\}$.

ENROLL(i), $i \in \{0,1\}$, return a value in $\{0,1,2\}$
case *current_state*
☹ : **return** Choose($\{0,1,2\}$)
☺ : $v \leftarrow$ Choose($\{0,1,2\}$)
if $i = 0$ **then** *current_state* $\leftarrow \langle \emptyset, v, \perp, \perp, \perp, 0\rangle$
else *current_state* $\leftarrow \langle \{v\}, \perp, \perp, \perp, \perp, 1\rangle$
return v
$\langle V, v_0, v_1, u_0, u_1, d\rangle$:
$v \leftarrow$ Choose($\{0,1,2\}$)
if $i = 0$ **then**
if $v_0 \neq \perp$ **then** *current_state* $\leftarrow$ ☹
else *current_state* $\leftarrow \langle V, v, v_1, u_0, u_1, d\rangle$
else $(*\ i = 1\ *)$
current_state $\leftarrow \langle V \cup \{v\}, v_0, v_1, u_0, u_1, d\rangle$
return v

REVEAL(i, v, u), $i, u \in \{0,1\}$, $v \in \{0,1,2\}$, return a value in $\{0,1\}$
case *current_state*
☺ : *current_state* $\leftarrow$ ☹; **return** Choose($\{0,1\}$)
☹ : **return** Choose($\{0,1\}$)
$\langle V, v_0, v_1, u_0, u_1, d\rangle$:
if $i = 0$ **then**
if $v = v_0$ **and** $u_0 = \perp$ **and** ($u_1 = \perp$ **or** ($v = v_1 \Leftrightarrow u \neq u_1$)) **then**
current_state $\leftarrow \langle V, v_0, v_1, u, u_1, d\rangle$; **return** d
else
current_state $\leftarrow$ ☹; **return** Choose($\{0,1\}$)
else $(*\ i = 1\ *)$
if $v \in V$ **and** (($u_1 = \perp$ **and** ($u_0 = \perp$ **or** ($v = v_0 \Leftrightarrow u \neq u_0$))) **or**
($v = v_1$ **and** $u = u_1$)) **then**
current_state $\leftarrow \langle V, v_0, v, u_0, u, d\rangle$; **return** d
else
current_state $\leftarrow$ ☹; **return** Choose($\{0,1\}$)

Figure 2: Specification of type $\mathsf{booster}_\infty$.

initial values provided by the object. Any deviation from this etiquette will result in the object becoming "upset". Once upset, the object remains upset forever, responding to any operation applied to it by returning an arbitrary value.

To achieve this behaviour, type $\mathsf{booster}_\infty$ has two distinguished states: the *fresh* state, denoted ☺, and the *upset* state, denoted ☹. The remaining states are of the form $\langle V, v_0, v_1, u_0, u_1, d\rangle$, where $V \subseteq \{0,1,2\}$, $v_0, v_1 \in \{\perp, 0, 1, 2\}$, $u_0, u_1 \in \{\perp, 0, 1\}$, and $d \in \{0, 1\}$. The intended meaning of state $\langle V, v_0, v_1, u_0, u_1, d\rangle$ is as follows (assuming the object is initialised to the fresh state):

- d is the "secret" — whether P_0 or some process in $\mathcal{P}_1$ is the process that applied the first ENROLL operation (more accurately, the parameter of the first ENROLL operation).
- V is the set of challenge initial values given to processes in $\mathcal{P}_1$, using one of which these processes must solve Equality Negation (more accurately, the set of values returned to ENROLL(1) operations).
- v_0 is the challenge initial value using which P_0 must solve Equality Negation (more accurately, the value returned to ENROLL(0)).
- v_1 is the challenge initial value chosen from V by processes in $\mathcal{P}_1$, using which these processes must solve Equality Negation (more accurately, the challenge initial-value parameter of the first REVEAL$(1, *, *)$ operation).
- u_0 is the decision reached for Equality Negation by P_0 (more accurately, the decision parameter of the first REVEAL$(0, *, *)$ operation).
- u_1 is the decision reached for Equality Negation by the first process in $\mathcal{P}_1$ that tries to get the secret (more accurately, the decision parameter of the first REVEAL$(1, *, *)$ operation).

With this information, the object knows all it needs in order to respond appropriately to operations.

4 The main result

In the next two sections we shall prove that each of $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$ has Consensus number one. Taking this for granted, in this section we prove the main result of this paper: in the context of nondeterministic types, no nontrivial wait-free hierarchy can be robust. The key to this proof is the following lemma.

Lemma 1 *For any $n \geq 1$, if Consensus among n processes can be solved using only $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$ objects and registers, then Consensus among $n+1$ processes can also be solved using only $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$ objects and registers.*

Shared: all $\mathsf{booster}_\infty$ and $\mathsf{negation}_\infty$ objects and registers used in $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$, each has the initial state specified by $\mathbf{A}_c^1$ or $\mathbf{A}_c^2$

O_b: $\mathsf{booster}_\infty$, initialised to ☺

O_n: $\mathsf{negation}_\infty$, initialised to ☺

R_0, R_1: $\mathsf{register}$, each initialised to $\perp$

For processes P_0

```
R_0 := initial value of P_0
v := Apply(P_0, ENROLL(0), O_b)
u := Apply(P_0, NEGATE(0, v), O_n)
d := Apply(P_0, REVEAL(0, v, u), O_b)
decide R_d
```

For process P_k, $k \in \{1, \ldots, n\}$

```
init := initial value of P_k
R_1 := Execute(P_k, A_c^1, init)
w := Apply(P_k, ENROLL(1), O_b)
v := Execute(P_k, A_c^2, w)
u := Apply(P_k, NEGATE(1, v), O_n)
d := Apply(P_k, REVEAL(1, v, u), O_b)
decide R_d
```

Figure 3: A Consensus algorithm for $n+1$ processes using $\{\mathsf{negation}_\infty, \mathsf{booster}_\infty, \mathsf{register}\}$.

PROOF. Let $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ be algorithms that solve Consensus among n processes, $P_1, P_2, \ldots, P_n$, and use only $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$ objects and registers. Without loss of generality, we may assume that $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ solve Consensus with initial values drawn from the set {0,1,2}.[2]

As we shall show, the algorithm in Figure 3 uses $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ to solve Consensus among $n+1$ processes, $P_0, P_1, \ldots, P_n$. By inspection, in addition to the (registers, $\mathsf{booster}_\infty$ and $\mathsf{negation}_\infty$) objects used by $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$, that algorithm uses only two registers, one $\mathsf{booster}_\infty$ object and one $\mathsf{negation}_\infty$ object. This proves the lemma.

Before proving that the algorithm in Figure 3 does, in fact, solve Consensus, we informally explain how it works.

- P_0 first writes its initial value into register R_0. It then enrolls to the $\mathsf{booster}_\infty$ object O_b by applying $\textsc{enroll}(0)$, and obtains a challenge initial value, say v, with which it is required to solve Equality Negation. To do so, P_0 applies $\textsc{negate}(0, v)$ to the $\mathsf{negation}_\infty$ object O_n. Let u be the value returned by O_n. By applying $\textsc{reveal}(0, v, u)$ to O_b, P_0 determines whether it or one of $P_1, \ldots, P_n$ was the first to enroll to O_b.

- Processes $P_1, \ldots, P_n$ first agree on one of their initial values; they use $\mathbf{A}_c^1$ for that purpose. They then write the agreed-upon initial value into register R_1 (multiple processes may write into R_1, but they all write the same value). Each P_k,

[2]In our definition of Consensus (sometimes called *binary* Consensus for emphasis), the initial values of processes are drawn from the set $\{0, 1\}$. It is well-known [4] that if a set of types that includes $\mathsf{register}$ can be used to solve (binary) Consensus, it can also be used to solve Consensus with initial values drawn from an arbitrary set.

$1 \leq k \leq n$, then enrolls to O_b by applying ENROLL(1), and obtains a challenge initial value. Object O_b may hand out different challenge initial values to processes $P_1, \ldots, P_n$. These processes use the Consensus algorithm $\mathbf{A}_c^2$ to agree upon one of these challenge initial values, say v. Each P_k, $1 \leq k \leq n$, uses v as its initial value to solve Equality Negation. To do so, P_k applies NEGATE$(1, v)$ to the $\mathsf{negation}_\infty$ object O_n. Let u be the value returned by O_n. By applying REVEAL$(1, v, u)$ to O_b, P_k determines whether P_0 or one of the other processes was the first to enroll to O_b.

Since all processes agree on whether P_0 or one of the other processes was the first to enroll to O_b, they can agree on the initial value of some process: If P_0 was the first to enroll, all processes agree on the value they read in R_0; otherwise, they all agree on the value they read in R_1.

We now prove that the algorithm in Figure 3 solves Consensus. It is clear that the algorithm is wait-free (since both $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ are) and that it satisfies Termination. Consider any execution of the algorithm. From the specification of $\mathsf{booster}_\infty$ and $\mathsf{negation}_\infty$ it is easy to verify that the algorithm accesses O_b and O_n in such a way that these objects never become upset. From the specification of $\mathsf{booster}_\infty$, if d is the parameter of the first ENROLL operation applied to O_b, then that ENROLL operation will cause O_b to enter a state of the form $\langle *, *, *, *, *, d \rangle$. The last component of the state is never changed, once it is set. Furthermore, by the specification of $\mathsf{booster}_\infty$, any REVEAL operation applied to O_b that does not cause the object to become upset returns the last component of the state. Thus,

Every REVEAL operation applied to O_b returns d, where d is the parameter of the first ENROLL operation applied to O_b. $(*)$

Next we claim that, for each $k \in \{0, 1\}$, at most one value is written into register R_k, and that value is an initial value of some process. This is clearly true for $k = 0$, since only P_0 writes into R_0, and the value it writes there is its initial value. For $k = 1$, this follows from the Agreement and Validity properties of the Consensus algorithm $\mathbf{A}_c^1$ used by the processes in $\{P_1, \ldots, P_n\}$ to determine the value they each write into R_1.

Thus, to prove the algorithm in Figure 3 satisfies Validity, it suffices to show that if a process decides by reading register R_d, then some process has previously written a value into R_d. This follows from $(*)$, since in the algorithm a process writes into R_k before applying ENROLL(k), for each $k \in \{0, 1\}$.

It remains to show that the algorithm satisfies Agreement. Consider any execution in which two distinct processes P and P' have both decided. By $(*)$, processes P and P' decide by reading the same register R_d, where d is the parameter of the first ENROLL operation applied to O_b. By Validity, a process decides by reading R_d only if some process has written a value into R_d. As argued earlier, only one value is written into R_d in any given execution. Therefore, P and P' decide the same value, the one written in R_d. Therefore, the algorithm satisfies Agreement. $\square$

Theorem 1 *For any $n \geq 1$, Consensus among n processes can be solved using only registers and objects of types* $\mathsf{negation}_\infty$ *and* $\mathsf{booster}_\infty$.

Proof. By induction on n. The basis, $n = 1$ is trivial (because Consensus for one process is trivial). The induction step is Lemma 1. □

Corollary 1 *There is no nontrivial wait-free hierarchy that is robust.*

Proof. Suppose h is a robust wait-free hierarchy of types; $h(T)$ denotes the level to which h assigns type T. By Theorems 2 and 3 (cf. Sections 5 and 6, respectively) since h is wait-free, $h(\mathsf{negation}_\infty) = h(\mathsf{booster}_\infty) = 1$. By universality of Consensus (cf. [1]) and Theorem 1, for any type T and any n, T can be implemented in a system of n processes using only registers and objects of types $\mathsf{negation}_\infty$ and $\mathsf{booster}_\infty$. Since h is robust, $h(T) \leq \max\{h(\mathsf{negation}_\infty), h(\mathsf{booster}_\infty)\} = 1$. In other words, h assigns every type to level one — that is, h is trivial. □

5 Type $\mathsf{negation}_\infty$ has consensus number one

In this section we prove that $\mathsf{negation}_\infty$ has Consensus number one, i.e., that Consensus between two processes is unsolvable using only $\mathsf{negation}_\infty$ objects and registers. The proof of the result in this section resembles the one that proves type $\mathsf{negation}$ has Consensus number one in Section 7 of Part I. Suppose, to the contrary, that there exists a Consensus algorithm $\mathbf{A}_c$ for processes P_0 and P_1 using only objects of type $\mathsf{negation}_\infty$ and registers. Let I be the initial configuration of $\mathbf{A}_c$ where P_0 has initial value 0 and P_1 has initial value 1. Let $\mathcal{F}$ be the full computation tree of $\mathbf{A}_c$ from I (cf. Section 2.3 of Part I), and $\mathcal{L}$ be the pruned computation tree of $\mathbf{A}_c$ from I (cf. Section 7.2 of Part I).

Lemma 5 of Part I (cf. Section 7.2) states some elementary properties of the pruned computation tree of an algorithm that solves Consensus between two processes and therefore applies here as well. Lemmata 6–10 of Part I (cf. Section 7.3) state properties of the pruned computation tree of an assumed algorithm for Consensus between two processes that uses only registers and $\mathsf{negation}$ objects. It is straightforward to verify that the proofs of all these results apply verbatim when the assumed Consensus algorithm uses only registers and $\mathsf{negation}_\infty$ (instead of $\mathsf{negation}$) objects. We will therefore make use of these results without repeating their proofs. For convenience of reference, the statements of these and other results proved in Section 7 of Part I that are used in this section are reproduced in Appendix A.

By Lemma 8 of Part I, the root of $\mathcal{L}$ is bivalent. Since every leaf of $\mathcal{L}$ is univalent, there is a critical node E in $\mathcal{L}$ so that:

- All children of E in $\mathcal{L}$ are univalent in $\mathcal{L}$.
- There are steps e_0 and e_1 of processes P_0 and P_1, respectively, such that $E \cdot e_0$ and $E \cdot e_1$ are nodes in $\mathcal{L}$ and have opposite valence in $\mathcal{L}$.

Without loss of generality, assume that $E \cdot e_0$ is 0-valent and $E \cdot e_1$ is 1-valent in $\mathcal{L}$. (Otherwise, we rename e_0 and e_1 and apply the same argument.)

By Lemma 9 of Part I, e_0 and e_1 must access the same object. Let $\mathcal{O}$ be that object. By Lemma 10 of Part I, $\mathcal{O}$ is not a register. We now show that $\mathcal{O}$ is not a $\mathsf{negation}_\infty$ object either. Our proof strategy for this is similar to the one used in Lemma 11 of Part I, except that it is a bit more complicated since the structure of type $\mathsf{negation}_\infty$ is more complicated than that of $\mathsf{negation}$.

Suppose, for contradiction, that $\mathcal{O}$ is of type $\mathsf{negation}_\infty$. For each $k \in \{0,1\}$, let $i_k \in \{0,1\}$ and $w_k \in \{0,1,2\}$ be the process and initial-value parameters, respectively, of the pending NEGATE operation to $\mathcal{O}$ that process P_k has in configuration $E(I)$. In other words, the operation of P_k that is pending in $E(I)$ is $\text{NEGATE}(i_k, w_k)$. Recall that the response of a NEGATE operation is 0 or 1. For each $k, u \in \{0,1\}$, let $d_k^u = (P_k, \text{NEGATE}(i_k, w_k), \mathcal{O}, u)$ — i.e., d_k is a step in which P_k applies $\text{NEGATE}(i_k, w_k)$ to $\mathcal{O}$ and receives u as response. Thus, $e_k \in \{d_k^0, d_k^1\}$ for $k \in \{0,1\}$.

For each $k \in \{0,1\}$, let d_k be an element of $\{d_k^0, d_k^1\}$ that is applicable to $E \cdot e_{\overline{k}}(I)$. (Note that at least one of d_k^0 or d_k^1 is applicable to $E \cdot e_{\overline{k}}(I)$ because P_k has $\text{NEGATE}(i_k, w_k)$ pending in $E \cdot e_{\overline{k}}(I)$. If both d_k^0 and d_k^1 are applicable to $E \cdot e_{\overline{k}}(I)$, then we arbitrarily choose one of them to be d_k.)

Lemma 2 *For each $k \in \{0,1\}$, $\mathcal{O}$ is not upset in $E \cdot e_k \cdot d_{\overline{k}}(I)$.*

PROOF. First we prove that $\mathcal{O}$ is not upset in $E(I)$. Assume, for contradiction, that $\mathcal{O}$ is upset in $E(I)$. Thus, $\mathcal{O}$ is also upset in $E \cdot e_0(I)$ and $E \cdot e_1(I)$ (because, once upset, a $\mathsf{negation}_\infty$ object remains upset forever). Since $\mathcal{O}$ is upset in $E \cdot e_1(I)$, by the specification of $\mathsf{negation}_\infty$, e_0 is applicable to $E \cdot e_1(I)$; by definition, $E \cdot e_1 \cdot e_0$ is a node in the full computation tree $\mathcal{F}$. By hypothesis, $E \cdot e_1$ is 1-valent in $\mathcal{L}$. Thus, by Lemma 5(e) of Part I, $\mathit{subtree}(E \cdot e_1, \mathcal{L}) = \mathit{subtree}(E \cdot e_1, \mathcal{F})$. Therefore, $E \cdot e_1 \cdot e_0$ is a node in $\mathcal{L}$; it is a 1-valent node in $\mathcal{L}$ because $E \cdot e_1$ is. P_0 and each object (including $\mathcal{O}$) is in the same state in $E \cdot e_0(I)$ as in $E \cdot e_1 \cdot e_0(I)$. This contradicts Lemma 7 of Part I, since $E \cdot e_0$ and $E \cdot e_1 \cdot e_0$ are univalent nodes of opposite valence in $\mathcal{L}$.

Next we prove that $\mathcal{O}$ is not upset in $E \cdot e_0(I)$. Assume, for contradiction, that $\mathcal{O}$ is upset in $E \cdot e_0(I)$. From the preceding paragraph, $\mathcal{O}$ is not upset in $E(I)$. Then, by the specification of $\mathsf{negation}_\infty$, it must be that $\mathcal{O}$ is in state $\langle v_0, v_1, u_0, u_1 \rangle$ in $E(I)$ and one of the following holds about the $\text{NEGATE}(i_0, w_0)$ operation applied to $\mathcal{O}$ in step e_0: (i) $i_0 = 0$ and $v_0 \neq \perp$; or (ii) $i_0 = 1$, $v_1 \neq \perp$, and $w_0 \neq v_1$. In each of these cases, since the state of $\mathcal{O}$ in $E \cdot e_1(I)$ is either ☹ or $\langle v_0, v_1, u_0, u_1 \rangle$, it can be verified that e_0 is applicable to $E \cdot e_1(I)$ (and hence $E \cdot e_1 \cdot e_0$ is a node in $\mathcal{F}$), and that $\mathcal{O}$ is upset in $E \cdot e_1 \cdot e_0(I)$. By hypothesis, $E \cdot e_1$ is 1-valent in $\mathcal{L}$. Thus, by Lemma 5(e) of Part I, $\mathit{subtree}(E \cdot e_1, \mathcal{L}) = \mathit{subtree}(E \cdot e_1, \mathcal{F})$. Since $E \cdot e_1 \cdot e_0$ is a node in $\mathcal{F}$, it is also a node in $\mathcal{L}$. Furthermore, it is 1-valent in $\mathcal{L}$ because $E \cdot e_1$ is. P_0 and each object (including $\mathcal{O}$) is in the same state in $E \cdot e_0(I)$ as in $E \cdot e_1 \cdot e_0(I)$. This contradicts Lemma 7 of Part I, since $E \cdot e_0$ and $E \cdot e_1 \cdot e_0$ are univalent nodes of opposite valence in $\mathcal{L}$.

Next we prove that $\mathcal{O}$ is not upset in $E \cdot e_1(I)$. Assume, for contradiction, that $\mathcal{O}$ is upset in $E \cdot e_1(I)$. By definition, $E \cdot e_0$ is a node in $\mathcal{L}$. Also, at least one of $E \cdot e_0 \cdot d_1^0$ or $E \cdot e_0 \cdot d_1^1$ is in $\mathcal{L}$ (because P_1 has NEGATE(i_1, w_1) pending in $E(I)$ and, thus, in $E \cdot e_0(I)$). Let $u \in \{0,1\}$ be such that $E \cdot e_0 \cdot d_1^u$ is in $\mathcal{L}$. Since $E \cdot e_0$ is, by definition, 0-valent in $\mathcal{L}$, so is $E \cdot e_0 \cdot d_1^u$. By assumption, $\mathcal{O}$ is upset in $E \cdot e_1(I)$, and, by the first paragraph of this proof, it is not upset in $E(I)$. By a case analysis of the two causes that can make $\mathcal{O}$ upset (as in the previous paragraph), it can be verified that both d_1^0 and d_1^1 are applicable to $E(I)$ (and, therefore, $E \cdot d_1^0$ and $E \cdot d_1^1$ are both nodes in $\mathcal{F}$), and that $\mathcal{O}$ is upset in $E \cdot d_1^u(I)$ and $E \cdot e_0 \cdot d_1^u(I)$. At least one of $E \cdot d_1^0$ and $E \cdot d_1^1$ is in $\mathcal{L}$. By the definition of the critical node E, any of these two nodes that is in $\mathcal{L}$ is univalent, and at least one of them is 1-valent in $\mathcal{L}$. Then, by Lemma 6 of Part I, both of these nodes are, in fact, in $\mathcal{L}$ and are 1-valent in $\mathcal{L}$. In particular, $E \cdot d_1^u$ is a 1-valent node in $\mathcal{L}$. As argued earlier, $\mathcal{O}$ is upset in both $E \cdot d_1^u(I)$ and $E \cdot e_0 \cdot d_1^u(I)$. Therefore, P_1 and each object is in the same state in $E \cdot d_1^u(I)$ as in $E \cdot e_0 \cdot d_1^u(I)$. This contradicts Lemma 7 of Part I, because $E \cdot d_1^u$ and $E \cdot e_0 \cdot d_1^u$ are univalent nodes of opposite valence in $\mathcal{L}$.

Finally, we prove that $\mathcal{O}$ is not upset in $E \cdot e_k \cdot d_{\overline{k}}$, for each $k \in \{0,1\}$. Suppose, for contradiction, that for some $k \in \{0,1\}$, $\mathcal{O}$ is upset in $E \cdot e_k \cdot d_{\overline{k}}$. By the previous two paragraphs, $\mathcal{O}$ is not upset in $E \cdot e_k(I)$. By a case analysis of the two causes that can make $\mathcal{O}$ upset, it can be verified that $\mathcal{O}$ is upset in $E \cdot d_{\overline{k}}(I)$, and therefore also in $E \cdot e_{\overline{k}}(I)$. This contradicts the result proven in one of the previous two paragraphs. □

In the proof of Lemma 2, we have shown that $\mathcal{O}$ is not upset in $E(I)$. The next lemma rules out the remaining possibilities for the state of $\mathcal{O}$.

Lemma 3 *$\mathcal{O}$ is neither fresh nor in a state of the form $\langle v_0, v_1, u_0, u_1 \rangle$ in $E(I)$.*

PROOF. Suppose, for contradiction, that $\mathcal{O}$ is either fresh or in a state of the form $\langle v_0, v_1, u_0, u_1 \rangle$ in $E(I)$. There are two cases, depending on the values of i_0 and i_1 (the process parameters of the two NEGATE operations pending in $E(I)$).

CASE 1. *For some $k \in \{0,1\}$, $i_k = 0$*: By Lemma 2, it must be that $i_{\overline{k}} = 1$. (If not, then $i_0 = i_1 = 0$ and, for each $k \in \{0,1\}$, $\mathcal{O}$ is upset in $E \cdot e_k \cdot d_{\overline{k}}(I)$, which contradicts Lemma 2.) We distinguish two subcases:

SUBCASE 1(A). *$\mathcal{O}$ is fresh in $E(I)$*: Since $\mathcal{O}$ is fresh in $E(I)$, by the specification of $\mathsf{negation}_\infty$, both d_1^0 and d_1^1 are applicable to $E(I)$. Thus, $E \cdot d_1^0$ and $E \cdot d_1^1$ are in $\mathcal{F}$. By definition of the critical node E, each node in *children*$(P_1, E, \mathcal{L})$ is univalent, and at least one of them (namely $E \cdot e_1$) is 1-valent. By Lemma 6 of Part I, both $E \cdot d_1^0$ and $E \cdot d_1^1$ are nodes in $\mathcal{L}$ and are 1-valent in $\mathcal{L}$. We further distinguish two cases, depending on whether the initial-value parameters, w_0 and w_1, of the two pending NEGATE operations are equal or not:

(i) $w_0 \neq w_1$: Let $u \in \{0,1\}$ be such that $e_0 = d_0^u$ (i.e., u is the response that P_0 receives in step e_0). By the specification of $\mathsf{negation}_\infty$, e_0 is the only step of P_0

that is applicable to $E \cdot d_1^u(I)$, and d_1^u is the only step of P_1 that is applicable to $E \cdot e_0(I)$. Therefore, by Lemma 5(d) of Part I, $E \cdot d_1^u \cdot e_0$ and $E \cdot e_0 \cdot d_1^u$ are in $\mathcal{L}$. Furthermore, $E \cdot d_1^u \cdot e_0$ is 1-valent in $\mathcal{L}$ (because, as we just showed, $E \cdot d_1^u$ is 1-valent in $\mathcal{L}$), and $E \cdot e_0 \cdot d_1^u$ is 0-valent in $\mathcal{L}$ (because, by assumption, $E \cdot e_0$ is 0-valent in $\mathcal{L}$).

By the specification of $\mathsf{negation}_\infty$, it can be checked that $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot d_1^u(I)$ as in $E \cdot d_1^u \cdot e_0(I)$ (namely, in state $\langle w_0, w_1, u, u \rangle$). Therefore, each process and each object is in the same state in $E \cdot e_0 \cdot d_1^u(I)$ as in $E \cdot d_1^u \cdot e_0(I)$. This contradicts Lemma 7 of Part I, since $E \cdot e_0 \cdot d_1^u$ and $E \cdot d_1^u \cdot e_0$ are univalent nodes of opposite valence in $\mathcal{L}$.

(ii) $w_0 = w_1$: The argument of Subcase 1(A)-i also applies in this case by replacing d_1^u with $d_1^{\overline{u}}$ throughout. (But now the state of $\mathcal{O}$ in $E \cdot e_0 \cdot d_1^{\overline{u}}(I)$ and $E \cdot d_1^{\overline{u}} \cdot e_0(I)$ is $\langle w_0, w_0, u, \overline{u} \rangle$ — instead of $\langle w_0, w_1, u, u \rangle$.)

SUBCASE 1(B). *$\mathcal{O}$ is in a state of the form $\langle v_0, v_1, u_0, u_1 \rangle$ in $E(I)$*: Since $i_k = 0$, it follows that $v_0 = \perp$. If not, then $\mathcal{O}$ is upset in $E \cdot e_k(I)$ and thus in $E \cdot e_k \cdot d_{\overline{k}}(I)$ as well, which contradicts Lemma 2. Since $v_0 = \perp$, by the definition of states of $\mathsf{negation}_\infty$, $v_1, u_1 \neq \perp$. By the specification of $\mathsf{negation}_\infty$, $d_{\overline{k}}^{u_1}$ is the only step of $P_{\overline{k}}$ that is applicable to $E(I)$ and does not upset $\mathcal{O}$. Thus, $e_{\overline{k}} = d_{\overline{k}}^{u_1}$. We further distinguish two cases:

(i) $w_k \neq w_{\overline{k}}$: From the specification of $\mathsf{negation}_\infty$, $d_k^{u_1}$ is the only step of P_k that is applicable to $E(I)$ and $E \cdot e_{\overline{k}}(I)$. Thus, $e_k = d_k^{u_1}$, and by Lemma 5(d) of Part I, $E \cdot e_{\overline{k}} \cdot e_k$ is in $\mathcal{L}$. Also, $e_{\overline{k}}$ is the only step of $P_{\overline{k}}$ that is applicable to $E \cdot e_k$ (since $\mathcal{O}$ is in state $\langle w_k, v_1, u_1, u_1 \rangle$ in $E \cdot e_k(I)$). By Lemma 5(d) of Part I, $E \cdot e_k \cdot e_{\overline{k}}$ is in $\mathcal{L}$.

By the specification of $\mathsf{negation}_\infty$, it can be checked that $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$ (namely, in state $\langle w_k, v_1, u_1, u_1 \rangle$). Therefore, each process and each object is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. This contradicts Lemma 7 of Part I, since $E \cdot e_0 \cdot e_1(I)$ and

(ii) $w_k = w_{\overline{k}}$: The argument in case 2(A)-i also applies in this case by replacing $d_k^{u_1}$ with $d_k^{\overline{u}_1}$ throughout.

CASE 2. *For both $k \in \{0, 1\}$, $i_k = 1$*: In this case, $w_0 = w_1 = w$, for some $w \in \{0, 1\}$. (Otherwise, for each $k \in \{0, 1\}$, $\mathcal{O}$ is upset in $E \cdot e_k \cdot d_{\overline{k}}(I)$, contradicting Lemma 2.) We distinguish two subcases:

SUBCASE 2(A). *$\mathcal{O}$ is fresh in $E(I)$*: The argument of Subcase 1(A)-i applies here as well (but now the state of $\mathcal{O}$ in $E \cdot e_0 \cdot d_1^u(I)$ and $E \cdot d_1^u \cdot e_0(I)$ is $\langle \perp, w, \perp, u \rangle$, instead of $\langle w_0, w_1, u, u \rangle$).

SUBCASE 2(B). *$\mathcal{O}$ is in a state of the form $\langle v_0, v_1, u_0, u_1 \rangle$ in $E(I)$*: We further distinguish three cases:

(i) $v_1 = \perp$ and $w \neq v_0$: From the specification of $\mathsf{negation}_\infty$, for each $k \in \{0, 1\}$, $d_k^{u_0}$ is the only step of P_k that is applicable to $E(I)$. (Since $v_1 = \perp$, $u_0 \neq \perp$, by the definition of the states of $\mathsf{negation}_\infty$.) Hence, $e_k = d_k^{u_0}$. Also, e_k is the only step of P_k that is applicable to $E \cdot e_{\overline{k}}(I)$. Thus, by Lemma 5(d) of Part I, both $E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ are in $\mathcal{L}$. Furthermore, they are univalent nodes of opposite valence in $\mathcal{L}$ (because, by definition, $E \cdot e_0$ and $E \cdot e_1$ are). By the specification of $\mathsf{negation}_\infty$, $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$ (namely, in state $\langle v_0, w, u_0, u_0 \rangle$). Therefore, each process and each object is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. This contradicts Lemma 7 of Part I, since $E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ are univalent nodes of opposite valence in $\mathcal{L}$.

(ii) $v_1 = \perp$ and $w = v_0$: The argument of Subcase 2(B)-i applies in this case too, replacing $d_k^{u_0}$ with $d_k^{\overline{u}_0}$ throughout. (But now the state of $\mathcal{O}$ in $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$ is $\langle v_0, v_0, u_0, \overline{u}_0 \rangle$, instead of $\langle v_0, w, u_0, u_0 \rangle$.)

(iii) $v_1 \neq \perp$: As argued in Subcase 1(B), it must be that $v_1 = w$ (otherwise, $\mathcal{O}$ is upset in $E \cdot e_k \cdot d_{\overline{k}}(I)$ for each $k \in \{0, 1\}$, which contradicts Lemma 2.) Given this, the argument of Subcase 2(B)-i also applies in this case by replacing $d_k^{u_0}$ with $d_k^{u_1}$ throughout. (Since $v_1 \neq \perp$, $u_1 \neq \perp$, by the definition of the states of $\mathsf{negation}_\infty$.) Object $\mathcal{O}$ remains in state $\langle v_0, v_1, u_0, u_1 \rangle$, instead of in state $\langle v_0, w, u_0, u_0 \rangle$, in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

Since all cases lead to a contradiction, the lemma follows. □

By Lemmata 2 and 3, $\mathcal{O}$ is not in any of the legitimate states of type $\mathsf{negation}_\infty$ in $E(I)$. Thus, $\mathcal{O}$ is not a $\mathsf{negation}_\infty$ object. As stated earlier, by Lemma 10 of Part I, $\mathcal{O}$ is not a register either. This contradicts our hypothesis that $\mathbf{A}_c$ uses only $\mathsf{negation}_\infty$ objects and registers. Thus,

Theorem 2 *Type* $\mathsf{negation}_\infty$ *has consensus number one.*

6 Type booster$_\infty$ has consensus number one

In this section we prove that type $\mathsf{booster}_\infty$ has Consensus number one. The proof is analogous to the one that proves booster has Consensus number one in Section 8 of Part I. To prove our result, we must show that Equality Negation between two processes is unsolvable using only $\mathsf{booster}_\infty$ objects and registers. Suppose, for contradiction, that $\mathbf{A}_{en}$ is an algorithm that solves Equality Negation between two processes, say P_0 and P_1, and uses only $\mathsf{booster}_\infty$ objects and registers.

In Section 8.1 of Part I we defined valence of nodes in computation trees of algorithms that solve Equality Negation (as opposed to Consensus) between two processes. Lemma 14 of Part I states that any algorithm that solves Equality Negation between two processes has an initial configuration I so that the root of the full computation tree $\mathcal{F}$ from I of that algorithm has a bivalent root. (The statement of this and other results proved in Part I that are used in this section are reproduced in Appendix B.) In the rest of this section, we focus exclusively on the initial configuration I and the full computation tree $\mathcal{F}$ (of $\mathbf{A}_{en}$ from I) whose existence is asserted by this lemma. Thus, from now on, we refer to the valence of a node without explicitly specifying the computation tree, since that will always be $\mathcal{F}$.

Since the root of $\mathcal{F}$ is bivalent and every leaf is univalent, there is a critical node E such that:

- All children of E in $\mathcal{F}$ are univalent.
- There are steps e_0 and e_1 of processes P_0 and P_1, respectively, such that $E \cdot e_0$ and $E \cdot e_1$ are nodes in $\mathcal{F}$ and have opposite valence.

Without loss of generality, assume that $E \cdot e_0$ is 0-valent and $E \cdot e_1$ is 1-valent (if not, we simply rename e_0 and e_1). A standard application of the "bivalence argument" shows that e_0 and e_1 access the *same* object $\mathcal{O}$ and that object cannot be a register.[3] Thus, $\mathcal{O}$ is a $\mathsf{booster}$ object. We say that $\mathcal{O}$ is *fresh* (respectively, *upset*) in a configuration C if the state of $\mathcal{O}$ in C is ☺ (respectively, ☹).

Lemma 4 *Object $\mathcal{O}$ is not upset in $E \cdot e_0(I)$ or in $E \cdot e_1(I)$.*

PROOF. Identical to the proof of Lemma 15 of Part I. □

Lemma 5 *$E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ are nodes in $\mathcal{F}$.*

PROOF. By definition of $\mathcal{F}$ it suffices to show that, for each $k \in \{0,1\}$, e_k is applicable to $E \cdot e_{\overline{k}}(I)$. If e_k is an ENROLL step, the lemma follows immediately by the specification of $\mathsf{booster}_\infty$. If e_k is a REVEAL step, then $\mathcal{O}$ is not fresh or upset in $E(I)$ (otherwise, $\mathcal{O}$ would be upset in $E \cdot e_k(I)$, contrary to Lemma 4). Thus, $\mathcal{O}$ is in a state of the form $\langle *, *, *, *, *, d\rangle$ in $E(I)$. By the specification of $\mathsf{booster}_\infty$, and since $\mathcal{O}$ is not upset in $E \cdot e_k(I)$, the response returned by the REVEAL operation in e_k is d. Since $\mathcal{O}$ is in a state of the form $\langle *, *, *, *, *, d\rangle$ in $E(I)$ and is not upset in $E \cdot e_{\overline{k}}(I)$, it must be in a state of the form $\langle *, *, *, *, *, d\rangle$ in $E \cdot e_{\overline{k}}(I)$ as well (because the last component of the state is never changed, once it is set). By the specification of $\mathsf{booster}_\infty$, a REVEAL operation to $\mathcal{O}$ applied in such a state can return the response d. Thus, e_k is indeed applicable to $E \cdot e_{\overline{k}}(I)$. □

Lemma 6 *The state of $\mathcal{O}$ in $E(I)$ is not of the form $\langle V, v_0, v_1, u_0, u_1, d\rangle$.*

[3]For example, this argument was used in the proofs of Lemmata 9 and 10 of Part I (see Appendix A). In fact, the arguments in this case are much simpler since we are now working with the full computation tree and need not worry about whether certain schedules applicable to I have been pruned.

PROOF. Suppose, for contradiction, that the state of $\mathcal{O}$ in $E(I)$ is $\langle V, v_0, v_1, u_0, u_1, d\rangle$. By Lemma 5, both $E\cdot e_0\cdot e_1$ and $E\cdot e_1\cdot e_0$ are nodes in $\mathcal{F}$. We shall show that

$$\text{the state of } \mathcal{O} \text{ is the same in } E\cdot e_0\cdot e_1(I) \text{ as in } E\cdot e_1\cdot e_0(I). \qquad (*)$$

$\mathcal{O}$ is the only object accessed by e_0 and e_1. Thus, by $(*)$, each process and each object is in the same state in $E\cdot e_0\cdot e_1(I)$ as in $E\cdot e_1\cdot e_0(I)$. By Lemma 13 of Part I, $E\cdot e_0\cdot e_1$ and $E\cdot e_1\cdot e_0$ have the same valence. This contradicts the fact that $E\cdot e_0\cdot e_1$ is 0-valent (because $E\cdot e_0$ is), while $E\cdot e_1\cdot e_0$ is 1-valent (because $E\cdot e_1$ is).

It remains to show $(*)$. Let op_0 and op_1 be the (ENROLL or REVEAL) operations to $\mathcal{O}$ of processes P_0 and P_1 that are pending in $E(I)$, respectively. There are three cases, depending on the types of op_0 and op_1.

CASE 1. *Both operations pending in $E(I)$ are* ENROLL*:* Let $op_0 = \text{ENROLL}(i_0)$ and $op_1 = \text{ENROLL}(i_1)$, for some $i_0, i_1 \in \{0,1\}$. For each $k \in \{0,1\}$, let r_k be the response of $\mathcal{O}$ to the ENROLL(i_k) operation in step e_k. There are three subcases:

SUBCASE 1(A). $i_0 = i_1 = 0$: From the specification of $\mathsf{booster}_\infty$, $\mathcal{O}$ is upset in both $E\cdot e_0\cdot e_1(I)$ and $E\cdot e_1\cdot e_0(I)$.

SUBCASE 1(B). $i_0 = i_1 = 1$: From the specification of $\mathsf{booster}_\infty$, $\mathcal{O}$ is in state $\langle V\cup\{r_0, r_1\}, v_0, v_1, u_0, u_1, d\rangle$ in both $E\cdot e_0\cdot e_1(I)$ and $E\cdot e_1\cdot e_0(I)$.

SUBCASE 1(C). *For some $k \in \{0,1\}$, $i_k = 0$ and $i_{\overline{k}} = 1$:* Without loss of generality we assume that $i_0 = 0$ and $i_1 = 1$. Then $v_0 = \bot$; if not, $\mathcal{O}$ would be upset in $E\cdot e_0(I)$, which contradicts Lemma 4. By the definition of the set of states of $\mathsf{booster}_\infty$, $u_0 = \bot$. From this, it follows that $\mathcal{O}$ is in state $\langle V\cup\{r_1\}, r_0, v_1, \bot, u_1, d\rangle$ in both $E\cdot e_0\cdot e_1(I)$ and $E\cdot e_1\cdot e_0(I)$.

CASE 2. *One of the two operations pending in $E(I)$ is* ENROLL *and the other is* REVEAL*:* Without loss of generality, we may assume that $op_0 = \text{ENROLL}(i_0)$ and $op_1 = \text{REVEAL}(i_1, v, u)$, for some $i_0, i_1, u \in \{0,1\}$ and $v \in \{0,1,2\}$. Let r be the response of the ENROLL(i_0) operation that is returned in step e_0. There are three subcases:

SUBCASE 2(A). $i_0 = 0$: Then $v_0 = \bot$ (otherwise, $\mathcal{O}$ would be upset in $E\cdot e_0(I)$, contrary to Lemma 4), and hence $u_0 = \bot$ (by the definition of the set of states of $\mathsf{booster}_\infty$). Also, $i_1 = 1$ (otherwise, $\mathcal{O}$ would be upset in $E\cdot e_1(I)$). Thus, $\mathcal{O}$ is in state $\langle V, \bot, v_1, \bot, u_1, d\rangle$ and the operations of P_0 and P_1 pending in $E(I)$ are ENROLL(0) and REVEAL$(1, v, u)$, respectively. From the specification of $\mathsf{booster}_\infty$, the state of $\mathcal{O}$ in both $E\cdot e_0\cdot e_1(I)$ and $E\cdot e_1\cdot e_0(I)$ is

$$\begin{cases} \langle V, r, v, \bot, u, d\rangle & \text{if } v\in V \textbf{ and } (u_1=\bot \textbf{ or } (v=v_1 \textbf{ and } u=u_1)) \\ \frownie & \text{otherwise} \end{cases}$$

SUBCASE 2(B). $i_0 = 1$ *and* $i_1 = 0$: Since $\mathcal{O}$ is not upset in $E\cdot e_1(I)$, the following Boolean expression is true

$$v = v_0 \textbf{ and } u_0 = \bot \textbf{ and } (u_1 = \bot \textbf{ or } (v = v_1 \Leftrightarrow u \neq u_1))$$

and $\mathcal{O}$ is in state $\langle V, v_0, v_1, u, u_1, d\rangle$ in $E \cdot e_1(I)$. Thus, $\mathcal{O}$ is in state $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$ in $E \cdot e_1 \cdot e_0(I)$.

The state of $\mathcal{O}$ in $E \cdot e_0(I)$ is $\langle V \cup \{r\}, v_0, v_1, \perp, u_1, d\rangle$. Since the preceding Boolean expression is true, by the specification of $\mathsf{booster}_\infty$, the application of $\text{REVEAL}(0, v, u)$ to $\mathcal{O}$ in state $\langle V \cup \{r\}, v_0, v_1, \perp, u_1, d\rangle$ does not cause $\mathcal{O}$ to become upset; in fact, it causes it to enter state $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$. Thus, e_1 is applicable to $E \cdot e_0(I)$, and the state of $\mathcal{O}$ in $E \cdot e_0 \cdot e_1(I)$ is $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$, which is the same as in $E \cdot e_1 \cdot e_0(I)$.

Subcase 2(c). $i_0 = 1$ *and* $i_1 = 1$: Similar to Subcase 2(b), except that in this case the state of $\mathcal{O}$ in $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$ is $\langle V \cup \{r\}, v_0, v, u_0, u, d\rangle$, instead of $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$.

Case 3. *Both operations pending in* $E(I)$ *are* REVEAL: Let $op_0 = \text{REVEAL}(i_0, v, u)$ and $op_1 = \text{REVEAL}(i_1, v', u')$, for some $i_0, i_1, u, u' \in \{0, 1\}$ and $v, v' \in \{0, 1, 2\}$. There are three subcases.

Subcase 3(a). $i_0 = i_1 = 0$: Then $\mathcal{O}$ is upset in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

Subcase 3(b). $i_0 \neq i_1$: Without loss of generality, we may assume that $i_0 = 0$ and $i_1 = 1$. It must be the case that $v = v_0$ and $u_0 = \perp$ (otherwise $\mathcal{O}$ is upset in $E \cdot e_0(I)$, contrary to Lemma 4). Also $v' \in V$ and either $u_1 = \perp$, or $v' = v_1$ and $u' = u_1$ (otherwise $\mathcal{O}$ is upset in $E \cdot e_1(I)$). Thus, the state of $\mathcal{O}$ in $E(I)$ is either $\langle V, v, \perp, \perp, \perp, d\rangle$ or $\langle V, v, v', \perp, u', d\rangle$. In either case, it can be verified from the specification of $\mathsf{booster}_\infty$ that the state of $\mathcal{O}$ in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$ is

$$\begin{cases} \langle V, v, v', u, u', d\rangle & \text{if } v = v' \Leftrightarrow u \neq u' \\ ☹ & \text{otherwise} \end{cases}$$

Subcase 3(c). $i_0 = i_1 = 1$: Since $\mathcal{O}$ is not upset in either one of $E \cdot e_0(I)$ or $E \cdot e_1(I)$, it must be that $v, v' \in V$ and $\mathcal{O}$ is in state $\langle V, v_0, v, u_0, u, d\rangle$ in $E \cdot e_0(I)$ and in state $\langle V, v_0, v', u_0, u', d\rangle$ in $E \cdot e_1(I)$. From the specification of $\mathsf{booster}_\infty$, it can be checked that the state of $\mathcal{O}$ in $E \cdot e_0 \cdot e_1(I)$ is

$$\begin{cases} \langle V, v_0, v', u_0, u', d\rangle & \text{if } v' = v \text{ and } u' = u \\ ☹ & \text{otherwise} \end{cases}$$

Similarly, the state of $\mathcal{O}$ in $E \cdot e_1 \cdot e_0(I)$ is

$$\begin{cases} \langle V, v_0, v, u_0, u, d\rangle & \text{if } v = v' \text{ and } u = u' \\ ☹ & \text{otherwise} \end{cases}$$

Thus, $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$.

This completes the proof of $(*)$, and thereby of the lemma. □

By Lemmata 4 and 6, $\mathcal{O}$ must be fresh in $E(I)$. Furthermore, each P_k must have an ENROLL operation to $\mathcal{O}$ pending in $E(I)$ (otherwise, by the specification of $\mathsf{booster}_\infty$, $\mathcal{O}$ would be upset in $E \cdot e_k(I)$, contrary to Lemma 4).

Lemma 7 *For each $k \in \{0,1\}$, let $j_k \in \{0,1\}$ be the parameter of the* ENROLL *operation of P_k pending in $E(I)$. Then $j_0 \neq j_1$.*

PROOF. Suppose, for contradiction, that $j_0 = j_1$. By Lemma 5, $E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ are nodes in $\mathcal{F}$. We first show that $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. Recall that $j_0, j_1 \in \{0,1\}$. There are two cases:

CASE 1. $j_0 = j_1 = 0$: Then $\mathcal{O}$ is upset in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

CASE 2. $j_0 = j_1 = 1$: Let r_0 and r_1 be the responses of $\mathcal{O}$ returned to the ENROLL operations in e_0 and e_1, respectively. Then $\mathcal{O}$ is in state $\langle \{r_0, r_1\}, \perp, \perp, \perp, \perp, 1 \rangle$, in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

Since $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$ and $\mathcal{O}$ is the only object accessed in steps e_0 and e_1, each process and each object is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. By Lemma 13 of Part I, $E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ have the same valence. This contradicts the fact that $E \cdot e_0 \cdot e_1$ is 0-valent (because $E \cdot e_0$ is), while $E \cdot e_1 \cdot e_0$ is 1-valent (because $E \cdot e_1$ is). $\square$

To summarise: $\mathcal{O}$ is fresh in $E(I)$, each P_k has an ENROLL(j_k) operation to $\mathcal{O}$ pending in $E(I)$, and $j_0 \neq j_1$. By Lemma 21 of Part I, no process can decide unless a REVEAL operation is applied to $\mathcal{O}$.[4] Thus, by the Termination requirement of Equality Negation, there is a schedule S that is applicable to $E(I)$ so that no process has taken a REVEAL step to $\mathcal{O}$ in S, and each process has a REVEAL operation to $\mathcal{O}$ pending in $E \cdot S(I)$. S contains ENROLL steps of both processes because each process has an ENROLL operation pending in $E(I)$ but has a REVEAL operation pending in $E \cdot S(I)$. By Lemma 7, S contains both ENROLL(0) and ENROLL(1) steps. The next lemma states that the application of the two REVEAL operations pending in $E \cdot S(I)$ does not cause $\mathcal{O}$ to become upset.

Lemma 8 *Consider any node $E \cdot S$ in $\mathcal{F}$ such that S contains no* REVEAL *step to $\mathcal{O}$ and both processes have a* REVEAL *operation to $\mathcal{O}$ pending in $E \cdot S(I)$. For each $k \in \{0,1\}$, let d_k be a step of P_k that is applicable to $E \cdot S(I)$, and $c_{\overline{k}}$ be a step of $P_{\overline{k}}$ that is applicable to $E \cdot S \cdot d_k(I)$. (Note that d_k and $c_{\overline{k}}$ are* REVEAL *steps that access $\mathcal{O}$.) Then, $\mathcal{O}$ is not upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$.*

[4]Lemma 21 of Part I, as well as Lemmata 18 and 20 of Part I to which we appeal below, was proved assuming that $\mathbf{A}_{en}$ is an algorithm that solves Equality Negation between two processes and uses only registers and $\mathsf{booster}$ objects. It can be checked that the proofs of these results apply, verbatim, if $\mathbf{A}_{en}$ uses only registers and $\mathsf{booster}_\infty$ objects. Thus, we shall use these results without repeating their proofs. Lemmata 13 and 14 of Part I that were cited earlier in the section apply for *any* algorithm that solves Equality Negation between two processes, regardless of the object types it uses.

PROOF. Suppose, to the contrary, that $\mathcal{O}$ is upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$, for some $k \in \{0,1\}$. Let P_ℓ, $\ell \in \{0,1\}$, be the process that does *not* take the first step in S, and $\hat{S}$ be the schedule obtained from S by moving the first step of P_ℓ to the beginning.[5] By Lemma 20 of Part I, $E \cdot \hat{S}$ is a node in $\mathcal{F}$, and each process and each object except $\mathcal{O}$ is in the same state in $E \cdot S(I)$ as in $E \cdot \hat{S}(I)$. We shall prove that

$$E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}} \text{ is a node in } \mathcal{F}, \text{ and } \mathcal{O} \text{ is upset in } E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}(I). \qquad (**)$$

Given $(**)$, each process and each object (including $\mathcal{O}$) is in the same state in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$ as in $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}(I)$. By Lemma 13 of Part I, $E \cdot S \cdot d_k \cdot c_{\overline{k}}$ and $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}$ have the same valence. This contradicts Lemma 18 of Part I, since P_ℓ takes the first step in $\hat{S}$, while $P_{\overline{\ell}}$ takes the first step in S.

It remains to show $(**)$. There are three cases:

CASE 1. *$\mathcal{O}$ is upset is $E \cdot S(I)$*: Since S contains only ENROLL operations to $\mathcal{O}$, it must contain two ENROLL(0) operations to $\mathcal{O}$ (otherwise $\mathcal{O}$ would not be upset in $E \cdot S(I)$). Then so does $\hat{S}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S}(I)$. It is immediate from the specification of $\mathsf{booster}_\infty$, that d_k is applicable to $E \cdot \hat{S}(I)$, and $c_{\overline{k}}$ is applicable to $E \cdot \hat{S} \cdot d_k(I)$. Therefore, $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}$ is a node in $\mathcal{F}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}(I)$ (because, once a $\mathsf{booster}_\infty$ object becomes upset, it remains upset forever).

CASE 2. *$\mathcal{O}$ is not upset is $E \cdot S(I)$ but it is upset in $E \cdot S \cdot d_k(I)$*: S contains exactly one ENROLL(0) and at least one ENROLL(1) step. (We have already argued, in the paragraph preceding Lemma 8, that S contains both ENROLL(0) and ENROLL(1) steps. It cannot contain two ENROLL(0) steps because, otherwise, $\mathcal{O}$ would be upset in $E \cdot S(I)$, contrary to the hypothesis of this case.) By assumption, S does not contain any REVEAL steps. Since $\mathcal{O}$ is fresh in $E(I)$ and it is not upset in $E \cdot S(I)$, the state of $\mathcal{O}$ in $E \cdot S(I)$ is $\langle V, v_0, \perp, \perp, \perp, j_{\overline{\ell}} \rangle$, where v_0 is the return value of the ENROLL(0) step in S and V is the set of return values of the ENROLL(1) steps in S. (To see why the last component of the state is $j_{\overline{\ell}}$, recall that $P_{\overline{\ell}}$ is the process that takes the first step in S, and that step involves an ENROLL($j_{\overline{\ell}}$) operation to $\mathcal{O}$. Since $\mathcal{O}$ is fresh in $E(I)$, when that ENROLL($j_{\overline{\ell}}$) operation is applied to $\mathcal{O}$, the state of $\mathcal{O}$ is set to a state of the form $\langle *, *, \perp, \perp, \perp, j_{\overline{\ell}} \rangle$. The last component of the state is not changed, once it is set.) Thus, the state of $\mathcal{O}$ in $E \cdot \hat{S}(I)$ is $\langle V, v_0, \perp, \perp, \perp, j_\ell \rangle$ — i.e., the same as in $E \cdot S(I)$, except for the last component.

Since $\mathcal{O}$ is upset in $E \cdot S \cdot d_k(I)$ it follows, from the specification of $\mathsf{booster}_\infty$, that d_k is a REVEAL(i, v, u) step to $\mathcal{O}$ so that one of the following holds:

(a) $i = 0$ and $v \neq v_0$ (i.e., v is not the return value of the ENROLL(0) step in S); or

(b) $i = 1$ and $v \notin V$ (i.e., v is not the return value of some ENROLL(1) step in S).

[5] That is, if $S = S' \cdot e \cdot S''$ for some schedules S', S'' and a step e, such that S' is a solo $P_{\overline{\ell}}$ schedule and e is a step of P_ℓ, then $\hat{S} = e \cdot S' \cdot S''$.

In either case, it is immediate from the specification of $\mathsf{booster}_\infty$ that the application of a $\text{REVEAL}(i, v, u)$ operation to $\mathcal{O}$ in state $E \cdot \hat{S}(I)$ will cause $\mathcal{O}$ to become upset and can return any value in $\{0, 1\}$. Thus, d_k is applicable to $E \cdot \hat{S}(I)$ and $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot d_k(I)$. Hence, $c_{\overline{k}}$ is applicable to $E \cdot \hat{S} \cdot d_k(I)$. This implies that $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}(I)$ is a node in $\mathcal{F}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}(I)$.

CASE 3. *$\mathcal{O}$ is not upset is $E \cdot S \cdot d_k(I)$ but it is upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$:* Let the REVEAL operation applied to $\mathcal{O}$ in d_k be $\text{REVEAL}(i, v, u)$. Since $\mathcal{O}$ is not upset in $E \cdot S \cdot d_k(I)$, and S does not contain any REVEAL steps, we conclude the following about the state of $\mathcal{O}$ in $E \cdot S \cdot d_k(I)$:

- If $i = 0$ then that state is $\langle V, v_0, \perp, u, \perp, j_{\overline{\ell}} \rangle$, where v_0 is the return value of the $\text{ENROLL}(0)$ step in S, $v = v_0$, and V is the set of return values of the $\text{ENROLL}(1)$ steps in S.
- If $i = 1$ then that state is $\langle V, v_0, v, \perp, u, j_{\overline{\ell}} \rangle$, where v_0 is the return value of the $\text{ENROLL}(0)$ step in S, V is the set of return values of the $\text{ENROLL}(1)$ steps in S, and $v \in V$.

Thus, the state of $\mathcal{O}$ in $E \cdot \hat{S} \cdot d_k(I)$ is as above, except that the last component is j_ℓ instead of $j_{\overline{\ell}}$. Since $\mathcal{O}$ is upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$, from the specification of $\mathsf{booster}_\infty$ it follows that $c_{\overline{k}}$ is a $\text{REVEAL}(i', v', u')$ step to $\mathcal{O}$ so that one of the following holds:

(a) $i' = 0$ and $v' \neq v_0$ (i.e., v' is not the return value of the $\text{ENROLL}(0)$ step in S).

(b) $i' = 1$ and $v' \in V$ (i.e., v' is not the return value of some $\text{ENROLL}(1)$ step in S).

(c) $i' = i = 0$.

(d) $i' = i = 1$, and either $v' \neq v$, or $u' \neq u$.

(e) $i' \neq i$ and it is not the case that $v' = v \Leftrightarrow u' \neq u$.

In each of these cases, it can be seen that the application of a $\text{REVEAL}(i', v', u')$ operation to $\mathcal{O}$ when in a state of the form $\langle V, v_0, \perp, u, \perp, j_\ell \rangle$ or $\langle V, v_0, v, \perp, u, j_\ell \rangle$ (the only possible states of $\mathcal{O}$ in $E \cdot \hat{S} \cdot d_k(I)$) will cause $\mathcal{O}$ to become upset and can return any value in $\{0, 1\}$.

Thus, $c_{\overline{k}}$ is applicable to $E \cdot \hat{S} \cdot d_k(I)$. Therefore, $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}$ is a node in $\mathcal{F}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot d_k \cdot c_{\overline{k}}(I)$.

This completes the proof of ($**$), and thereby of the lemma. $\square$

Lemma 9 *Consider any node $E \cdot S$ in $\mathcal{F}$ such that S contains no* REVEAL *step to $\mathcal{O}$ and both processes have a* REVEAL *operation to $\mathcal{O}$ pending in $E \cdot S(I)$. For each $k \in \{0, 1\}$, let i_k, v_k and u_k be the parameters of the* REVEAL *operation to $\mathcal{O}$ of P_k that is pending in $E \cdot S(I)$. Then,*

(a) *For each* $k \in \{0,1\}$, v_k *is the response of an* ENROLL(i_k) *step to* $\mathcal{O}$ *contained in* S.

(b) i_0 *and* i_1 *cannot be both equal to 0.*

(c) *If* $i_0 = i_1 = 1$ *then* $v_0 = v_1$ *and* $u_0 = u_1$.

(d) *If* $i_0 \neq i_1$ *then* $v_0 = v_1 \Leftrightarrow u_0 \neq u_1$.

PROOF. For each $k \in \{0,1\}$, let d_k be a step of P_k that is applicable to $E \cdot S(I)$, and $c_{\overline{k}}$ be a step of $P_{\overline{k}}$ that is applicable to $E \cdot S \cdot d_k(I)$, By the specification of booster$_\infty$, if any of the properties (a)–(d) failed to hold, $\mathcal{O}$ would be upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$, for any $k \in \{0,1\}$. This would contradict Lemma 8. Thus, (a)–(d) must all hold. □

We now describe an algorithm, derived from $\mathbf{A}_{en}$, that solves Equality Negation between two processes Q_0 and Q_1, and uses one booster$_\infty$ object less than $\mathbf{A}_{en}$. The idea is very similar to an analogous algorithm in Part I (cf. Figure 10 of Part I). Roughly speaking, the algorithm works as follows: Each process Q_k simulates the actions of the corresponding process P_k in an execution of $\mathbf{A}_{en}$, and uses this execution to determine what value to decide. In this simulation, the processes are allowed to access all objects used by $\mathbf{A}_{en}$, except $\mathcal{O}$, each of which is initialised to the state it has in configuration $E(I)$.

Q_k starts its simulation of P_k by pretending that P_k is in the state it has in configuration $E(I)$. It then simulates each step of P_k that accesses an object other than $\mathcal{O}$ by applying the appropriate operation to that object. To simulate an ENROLL step accessing $\mathcal{O}$, Q_k pretends that it applied this operation to $\mathcal{O}$ and received its own initial value (for which it must solve Equality Negation) as the object's response. By Lemma 21 of Part I, if $\mathbf{A}_{en}$ reaches the configuration $E(I)$, P_0 and P_1 cannot decide unless a REVEAL operation is applied to $\mathcal{O}$. Thus, eventually Q_k will have to simulate a REVEAL step accessing $\mathcal{O}$. When Q_k is about to simulate a REVEAL to $\mathcal{O}$, it does not actually simulate this step. Instead, it decides a value based on the parameters of the REVEAL operation that it is about to simulate and the parameter of the first ENROLL operation to $\mathcal{O}$ that it previously simulated. The algorithm is shown in Figure 4. We use the same notation and conventions as in the algorithm in Figure 10 of Part I.

Lemma 10 *If* $\mathbf{A}_{en}$ *is an Equality Negation algorithm for two processes, then the algorithm in Figure 4 also solves Equality Negation for two processes, using one fewer* negation$_\infty$ *object.*

PROOF. It is obvious from Figure 4 that the algorithm uses one booster$_\infty$ object less than $\mathbf{A}_{en}$. In the remainder of this proof we show that the algorithm solves Equality Negation.

Consider any execution of the algorithm. Let S_i be the value of the auxiliary variable S when the schedule it contains consists of exactly i steps. A straightforward induction on i shows that for all $i \geq 0$ such that S_i is defined,

Shared: every object other than $\mathcal{O}$ used in algorithm $\mathbf{A}_{en}$,
each initialised to the state it has in $E(I)$
S : auxiliary variable : schedule of $\mathbf{A}_{en}$ applicable to $E(I)$, initially empty

For process Q_k, $k \in \{0, 1\}$
$\quad$ *init* := initial value of Q_k
$\quad$ *state* := state of P_k in $E(I)$
$\quad$ j := parameter of the ENROLL operation to $\mathcal{O}$ by P_k pending in $E(I)$
$\quad$ **while** Q_k has not decided **do**
$\quad\quad$ $\langle op, O\rangle := \text{NextOp}(P_k, state)$
$\quad\quad$ **if** $O \neq \mathcal{O}$ **then**
$\quad\quad\quad$ $[\![\ r := \text{Apply}(P_k, op, O);\ \ S := S \cdot (P_k, op, O, r)\ ]\!]$
$\quad\quad\quad$ $state := \text{NextState}(P_k, state, r)$
$\quad\quad$ **else** $(*\ O = \mathcal{O}\ *)$
$\quad\quad\quad$ **if** $op = \text{ENROLL}(i)$ **then**
$\quad\quad\quad\quad$ $S := S \cdot (P_k, \text{ENROLL}(i), \mathcal{O}, init)$
$\quad\quad\quad\quad$ $state := \text{NextState}(P_k, state, init)$
$\quad\quad\quad$ **else**
$\quad\quad\quad\quad$ let i, v and u be such that $op = \text{REVEAL}(i, v, u)$
$\quad\quad\quad\quad$ **if** $i \neq j$ **then**
$\quad\quad\quad\quad\quad$ **if** $v = init$ **then decide** $\overline{u}$
$\quad\quad\quad\quad\quad$ **else decide** u
$\quad\quad\quad\quad$ **else**
$\quad\quad\quad\quad\quad$ **if** $i = 0$ **then** $(*\ i = j = 0\ *)$
$\quad\quad\quad\quad\quad\quad$ **decide** u
$\quad\quad\quad\quad\quad$ **else** $(*\ i = j = 1\ *)$
$\quad\quad\quad\quad\quad\quad$ **if** $v = init$ **then decide** u
$\quad\quad\quad\quad\quad\quad$ **else decide** $\overline{u}$

Figure 4: Solving Equality Negation for two processes using $\mathbf{A}_{en}$ with one fewer $\mathsf{negation}_\infty$ object.

(a) S_i is a schedule of $\mathbf{A}_{en}$ that is applicable to $E(I)$ and contains no REVEAL step to $\mathcal{O}$, and

(b) Let Q_k be the process that assigns S_i to S, $state_k$ be the value that Q_k sets its local variable *state* after assigning S_i to S, and S_{i+} be the value that Q_k assigns to S next (if any). Then $state_k$ is the state of P_k in configuration $E \cdot S_j(I)$ of $\mathbf{A}_{en}$, for any j, $i \leq j < i+$.

By (a), the algorithm in Figure 4 is wait-free and satisfies Termination. Otherwise, for some $k \in \{0, 1\}$, the while loop of Q_k does not terminate, and hence, S_i is defined for all integers $i \geq 0$. But then there is an infinite schedule S^* of $\mathbf{A}_{en}$ that is applicable to $E(I)$; this contradicts the fact that $\mathbf{A}_{en}$ is wait-free. Also, since only two processes execute the algorithm, Agreement is trivially satisfied.

It remains to show that the algorithm in Figure 4 satisfies Negation. Consider any schedule of the algorithm in which Q_0 and Q_1 both decide. For each $k \in \{0, 1\}$, let $init_k$ be the initial value of process Q_k (i.e., the value assigned to local variable *init* of Q_k in line 1); $state_k$ be the value of local variable *state* of process Q_k when Q_k decides; and j_k be the value assigned to local variable j of process Q_k in line 3. Let S^* be the value of S when Q_0 and Q_1 have both decided (i.e., S^* is the final value of the auxiliary variable S). By the algorithm, the step that P_k executes next when in $state_k$ is to apply a REVEAL operation to $\mathcal{O}$; let i_k, v_k and u_k be the parameters of that REVEAL operation. From (b), it follows that the operation of P_k pending in $E \cdot S^*(I)$ is a REVEAL(i_k, v_k, u_k) to $\mathcal{O}$. By (a), $E \cdot S^*$ is a node in $\mathcal{F}$, and S contains no REVEAL operations to $\mathcal{O}$. Thus, the hypothesis of Lemma 9 holds; we will make use of parts (a)–(d) of this lemma below. By line 11 of the algorithm, the only return values of ENROLL operations to $\mathcal{O}$ that appear in S^* are $init_0$ and $init_1$. Thus, by Lemma 9(a), $v_0, v_1 \in \{init_0, init_1\}$. There are three cases:

CASE 1. *For both* $k \in \{0, 1\}$, $i_k \neq j_k$: In this case, each Q_k decides by line 16 or 17 (depending on whether $v_k = init_k$). Also, $i_0 \neq i_1$ (because all of $i_0, i_1, j_0, j_1 \in \{0, 1\}$, and, by Lemma 7, $j_0 \neq j_1$). Thus, by Lemma 9(d), $u_0 = u_1$ if and only if $v_0 \neq v_1$.

By a case analysis of the four possible combinations of lines by which Q_0 and Q_1 can decide, and keeping in mind that $u_0 = u_1$ if and only if $v_0 \neq v_1$, we have that Q_0 and Q_1 decide the same value if and only if

$$(v_0 = init_0 \textbf{ and } v_1 = init_1 \textbf{ and } v_0 \neq v_1) \textbf{ or } (v_0 = init_0 \textbf{ and } v_1 \neq init_1 \textbf{ and } v_0 = v_1) \textbf{ or}$$
$$(v_0 \neq init_0 \textbf{ and } v_1 = init_1 \textbf{ and } v_0 = v_1) \textbf{ or } (v_0 \neq init_0 \textbf{ and } v_1 \neq init_1 \textbf{ and } v_0 \neq v_1).$$

As argued above, $v_0, v_1 \in \{init_0, init_1\}$. Thus, the above disjunction is equivalent to $init_0 \neq init_1$. In other words, Q_0 and Q_1 decide the same value if and only if they have different initial values.

CASE 2. *For both* $k \in \{0, 1\}$, $i_k = j_k$: By Lemma 7, $j_0 \neq j_1$. Thus, for some $k \in \{0, 1\}$, $i_k = j_k = 0$, and $i_{\overline{k}} = j_{\overline{k}} = 1$. Then Q_k decides by line 20, and $Q_{\overline{k}}$ decides by line 22 or 23 (depending on whether $v_{\overline{k}} = init_{\overline{k}}$). So we have that Q_0 and Q_1 decide the same value if and only if

$$(v_{\overline{k}} = init_{\overline{k}} \textbf{ and } u_k = u_{\overline{k}}) \textbf{ or } (v_{\overline{k}} \neq init_{\overline{k}} \textbf{ and } u_k = \overline{u_{\overline{k}}}).$$

In this case, $i_k \neq i_{\overline{k}}$. By Lemma 9(d), $u_k = u_{\overline{k}}$ if and only if $v_k \neq v_{\overline{k}}$. Thus, the above expression is equivalent to

$$(v_{\overline{k}} = init_{\overline{k}} \textbf{ and } v_k \neq v_{\overline{k}}) \textbf{ or } (v_{\overline{k}} \neq init_{\overline{k}} \textbf{ and } v_k = v_{\overline{k}}).$$

By Lemma 9(a) and the fact that $i_k = 0$, v_k is the response of an ENROLL(0) step to $\mathcal{O}$ that is contained in S^*. S^* cannot contain two ENROLL(0) steps (otherwise $\mathcal{O}$ would be upset in $E \cdot S^*(I)$ — and therefore still upset after the application of the two REVEAL operations pending in $E \cdot S^*(I)$ — contrary to Lemma 8). Since $j_k = i_k = 0$, P_k has an ENROLL(0) to $\mathcal{O}$ pending in $E(I)$. Hence, the unique ENROLL(0) to $\mathcal{O}$ in S^* is a step of P_k. By the algorithm, and specifically line 11, the response to that unique ENROLL(0) step to $\mathcal{O}$ in S^* is the initial value of P_k. Therefore, $v_k = init_k$. Thus, the above expression is equivalent to

$$(v_{\overline{k}} = init_{\overline{k}} \textbf{ and } init_k \neq v_{\overline{k}}) \textbf{ or } (v_{\overline{k}} \neq init_{\overline{k}} \textbf{ and } init_k = v_{\overline{k}}).$$

As argued earlier, $v_{\overline{k}} \in \{init_0, init_1\}$; so the above expression is equivalent to $init_k \neq init_{\overline{k}}$. In other words, Q_0 and Q_1 decide the same value if and only if they have different initial values.

CASE 3. *For some $k \in \{0,1\}$, $i_k \neq j_k$, and $i_{\overline{k}} = j_{\overline{k}}$*: By Lemma 7, $j_k \neq j_{\overline{k}}$, and by Lemma 9(b), i_k and $i_{\overline{k}}$ cannot both be 0. Hence, in this case we must have $i_k = 1$, $j_k = 0$ and $i_{\overline{k}} = j_{\overline{k}} = 1$. Therefore, Q_k decides by line 16 or 17 (depending on whether $v_k = init_k$), and $Q_{\overline{k}}$ decides by line 22 or 23 (depending on whether $v_{\overline{k}} = init_{\overline{k}}$). Also, in this case $i_k = i_{\overline{k}} = 1$ and thus, by Lemma 9(c), $v_0 = v_1$ and $u_0 = u_1$. Keeping in mind that $u_0 = u_1$, we have that Q_0 and Q_1 decide the same value if and only if

$$(v_k = init_k \textbf{ and } v_{\overline{k}} \neq init_{\overline{k}}) \textbf{ or } (v_k \neq init_k \textbf{ and } v_{\overline{k}} = init_{\overline{k}}).$$

Since $v_0 = v_1$ and, as argued earlier, $v_0, v_1 \in \{init_0, init_1\}$, the above expression is equivalent to $init_k \neq init_{\overline{k}}$. Hence, Q_0 and Q_1 decide the same value if and only if they have different initial values.

In all cases the Negation property is satisfied. □

We now have all the pieces needed to prove the main result of this section.

Theorem 3 *Type* $\mathsf{booster}_\infty$ *has Consensus number one.*

Proof. Suppose, to the contrary, that $\mathsf{booster}_\infty$ has Consensus number at least two. Hence, we can solve Consensus for two processes using only $\mathsf{booster}_\infty$ objects and registers. By Corollary 1 of Part I, we can solve Equality Negation for two processes using only $\mathsf{booster}_\infty$ objects and registers. Let $\mathbf{A}_{en}$ be an Equality Negation algorithm for two processes that uses the minimum possible number of $\mathsf{booster}_\infty$ objects (and any number of registers). By Theorem 1 of Part I, $\mathbf{A}_{en}$ uses at least one $\mathsf{booster}_\infty$ object. By Lemma 10, we can solve Equality Negation for two processes using one $\mathsf{booster}_\infty$ object less than $\mathbf{A}_{en}$. This contradicts the definition of $\mathbf{A}_{en}$. □

References

[1] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 11(1):124–149, January 1991.

[2] Prasad Jayanti. On the robustness of Herlihy's hierarchy. In *Proceedings of the 12th Annual ACM Symposium of Principles of Distributed Computing*, pages 145–157, 1993.

[3] Wai-Kau Lo and Vassos Hadzilacos. All of us are smarter than any of us: more on the robustness of the Consensus hierarchy (Part I). Technical Report CSRI-348, University of Toronto, Toronto, Ontario, Canada, June 1996.

[4] Serge Plotkin. Sticky bits and universality of consensus. In *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing*, pages 159–175, August 1989.

A Results proved in Part I and used in Section 5

Lemma 5 of Part I *Let S be any node in the full computation tree $\mathcal{F}$. Then*

(a) S is not a node in $\mathcal{L}$ if and only if S is a descendent of an avoidable 1-valent node in $\mathcal{F}$. (By convention, a node is a descendent of itself.)

(b) If S is 1-valent in $\mathcal{F}$ and is also in $\mathcal{L}$, then S is 1-valent in $\mathcal{L}$.

(c) If S is 0-valent in $\mathcal{F}$, then it is in $\mathcal{L}$ and is 0-valent in $\mathcal{L}$.

(d) If S is in $\mathcal{L}$ and $children(P, S, \mathcal{F}) = \{S'\}$ (N.B.: a singleton), then S' is in $\mathcal{L}$.

(e) If S is in $\mathcal{L}$ and is 1-valent in $\mathcal{L}$, then $subtree(S, \mathcal{L}) = subtree(S, \mathcal{F})$.

Lemma 6 of Part I *Let S be any node in $\mathcal{L}$ and P be any process such that $|children(P,S,\mathcal{F})| = 2$, each node in $children(P,S,\mathcal{L})$ is univalent and one of them is 1-valent in $\mathcal{L}$. Then $children(P,S,\mathcal{L}) = children(P,S,\mathcal{F})$ and both nodes in $children(P,S,\mathcal{L})$ are 1-valent in $\mathcal{L}$.*

Lemma 7 in Part I *Let S_0 and S_1 be two univalent nodes in $\mathcal{L}$ such that the state of some process and each object is the same in $S_0(I)$ as in $S_1(I)$. Then, S_0 and S_1 are either both 0-valent or both 1-valent in $\mathcal{L}$.*

Lemma 8 of Part I *The root of $\mathcal{L}$ is bivalent.*

Lemma 9 of Part I *Steps e_0 and e_1 access the same object $\mathcal{O}$.*

Lemma 10 of Part I *$\mathcal{O}$, the object accessed in e_0 and e_1, is not a register.*

B Results proved in Part I and used in Section 6

Lemma 13 of Part I *Let S_0 and S_1 be two univalent nodes in $\mathcal{F}$ such that the state of some process and each object is the same in $S_0(I)$ as in $S_1(I)$. Then, S_0 and S_1 are either both 0-valent or both 1-valent in $\mathcal{F}$.*

Lemma 14 of Part I *For some initial configuration I of algorithm $\mathbf{A}_{en}$, the full computation tree $\mathcal{F}$ of $\mathbf{A}_{en}$ from I has a bivalent root.*

Lemma 18 of Part I *For any node $E \cdot S$ in $\mathcal{F}$, if P_k is the process that takes the first step in S, then $E \cdot S$ is k-valent.*

Lemma 20 of Part I *Let $E \cdot S$ be any node in $\mathcal{F}$ such that S contains no* REVEAL *step to $\mathcal{O}$, and d_k be the first step of P_k in S, for any $k \in \{0,1\}$. Let $\hat{S}$ be the schedule obtained from S by moving d_k to the beginning (i.e., if $S = S' \cdot d_k \cdot S''$ for some S' and S'', then $\hat{S} = d_k \cdot S' \cdot S''$). Then, $E \cdot \hat{S}$ is a node in $\mathcal{F}$, and the state of each process and each object other than $\mathcal{O}$ is the same in $E \cdot \hat{S}(I)$ as in $E \cdot S(I)$.*

Lemma 21 of Part I *Let $E \cdot S$ be any node in $\mathcal{F}$ such that some process has decided in $E \cdot S(I)$. Then, S has a* REVEAL *step to $\mathcal{O}$.*

Corollary 1 of Part I *Let $\mathcal{S}$ be a set of object types that includes* register*. If Consensus for two processes is solvable using $\mathcal{S}$, then Equality Negation for two processes is also solvable using $\mathcal{S}$.*

Theorem 1 of Part I *There is no algorithm that solves Equality Negation for two processes using only registers.*