ON THE EXISTENCE OF A STRONG MINIMAL PAIR

GEORGE BARMPALIAS, MINGZHONG CAI, STEFFEN LEMPP, AND THEODORE A. SLAMAN

ABSTRACT. We show that there is a strong minimal pair in the computably enumerable Turing degrees, i.e., a pair of nonzero c.e. degrees $\mathbf{a}$ and $\mathbf{b}$ such that $\mathbf{a} \cap \mathbf{b} = \mathbf{0}$ and for any nonzero c.e. degree $\mathbf{x} \leq \mathbf{a}$, $\mathbf{b} \cup \mathbf{x} \geq \mathbf{a}$.

1. INTRODUCTION

Much of the work on the degree structure of the computably enumerable (c.e.) Turing degrees has focused on studying its finite substructures and how they can be extended to larger substructures. There are several reasons for this: The partial order of the c.e. degrees is a very complicated algebraic structure, with an undecidable first-order theory, by Harrington and Shelah [HS82]. So, on the one hand, as in classical algebra, a complicated structure is often best understood by studying its finite substructures. On the other hand, the existential fragment of the first-order theory of this degree structure (in the language of partial ordering $<$, least element 0 and greatest element 1) is known to be decidable by Sacks [Sa63], whereas by Lempp, Nies and Slaman [LNS98], the $\exists\forall\exists$-theory of this structure (in the language of partial ordering only) is undecidable. However, the decidability of the $\forall\exists$-theory of this structure has been an open question for a long time; and it is this question which can be rephrased in purely algebraic terms as a question about finite substructures:

Question 1.1 (Extendibility Question). *Let $\mathcal{P}$ and $\mathcal{Q}_i$ (with $i < n$) be finite posets such that for all $i < n$, $\mathcal{P} \subseteq \mathcal{Q}_i$. Under what conditions on $\mathcal{P}$ and the $\mathcal{Q}_i$ can any embedding of $\mathcal{P}$ into the c.e. Turing degrees be extended to an embedding of $\mathcal{Q}_i$ into the c.e. Turing degrees for some i (which may depend on the embedding of $\mathcal{P}$)?*

Call a partially ordered set $\mathcal{P}$ *bounded* if it contains distinguished least and great elements 0 and 1, respectively. We can now formulate the following modified

Question 1.2 (Extendibility Question with 0 and 1). *Let $\mathcal{P}$ and $\mathcal{Q}_i$ (with $i < n$) be finite bounded posets such that for all $i < n$, $\mathcal{P} \subseteq \mathcal{Q}_i$. Under what conditions on $\mathcal{P}$ and the $\mathcal{Q}_i$ can any embedding of $\mathcal{P}$ into the c.e. Turing degrees (preserving 0 and 1) be extended to an embedding of $\mathcal{Q}_i$ into the c.e. Turing degrees (preserving 0 and 1) for some i?*

The answer for $n = 1$ to Question 1.2 was given by the following

Theorem 1.3 (Slaman, Soare [SS99]). *Uniformly in finite bounded posets $\mathcal{P}$ and $\mathcal{Q}$, there is an effective procedure to decide whether any embedding of $\mathcal{P}$ into the c.e. Turing degrees (preserving 0 and 1) be extended to an embedding of $\mathcal{Q}$ into the c.e. Turing degrees (preserving 0 and 1).*

This result of Slaman and Soare built on a long line of research into the algebraic structure of the c.e. degrees, starting with the Sacks Splitting and Density Theorems [Sa63b, Sa64] and the proof of the existence of a minimal pair of c.e. degrees by Lachlan [La66] and Yates [Ya66].

In their proof in [SS99], Slaman and Soare identify two basic obstacles to extending an embedding. The first of these is lattice-theoretic: The c.e. degrees form an upper semilattice in which the meet of some but not all pairs of degrees exists. In fact, the major hurdle toward deciding the $\forall\exists$-theory of this structure has been the long-standing lattice embeddings problem, asking for an (effective) characterization of those finite lattices which can be embedded into the c.e. Turing degrees. (Note that the lattice embeddings problem can be phrased as a subproblem of the Extendibility Question by making all the Q_i one-point extensions of $\mathcal{P}$, each testing the preservation of a particular meet or join in the lattice embedding. Lerman [Le00] gave a noneffective (indeed a Π_2^0-)condition for lattice embeddability; a more recent survey is Lempp, Lerman and Solomon [LLS06].)

The other basic obstacle to extending an embedding identified by Slaman and Soare is a phenomenon some-times called "saturation"; a minimal example of it is given by setting $\mathcal{P} = \{0, a, b, 1\}$ (with incomparable a, b) and $Q = \mathcal{P} \cup \{x, z\}$ (with $0 < x < a, z$ and $b < z < 1$ but $x \not\leq b$ and $a \not\leq z$). In the general case, there may be a number of such elements $x \in Q - \mathcal{P}$, and for each x there will be a non-empty set $Z(x) \subseteq Q - \mathcal{P}$ of such z.

An early example of a specific instance of an answer to the Extendibility Question 1.2 for $n > 1$ was given by Lachlan's Nondiamond Theorem [La66]: No minimal pair of c.e. degrees can cup to $\mathbf{0}'$. (For this, we set $\mathcal{P} = \{0, a, b, 1\}$ (with incomparable a, b), $Q_0 = \mathcal{P} \cup \{x\}$ (with $0 < x < a, b$), and $Q_1 = \mathcal{P} \cup \{y\}$ (with $a, b < y < 1$).) So this is an instance of two lattice-theoretic obstructions which cannot be overcome individually, but can be overcome in combination.

The main theorem of this paper provides an example where a lattice-theoretic obstruction and a "saturation" obstruction cannot each be overcome either individually or in combination:

Main Theorem. *There is a* strong minimal pair *in the c.e. Turing degrees, i.e., there are nonzero c.e. degrees* $\mathbf{a}$ *and* $\mathbf{b}$ *such that* $\mathbf{a} \cap \mathbf{b} = \mathbf{0}$ *and for any nonzero c.e. degree* $\mathbf{x} \leq \mathbf{a}$, $\mathbf{b} \cup \mathbf{x} \geq \mathbf{a}$.

Note that this is an instance of the Extendibility Question 1.2 by setting $\mathcal{P} = \{0, a, b, 1\}$ (with incomparable a, b), $Q_0 = \mathcal{P} \cup \{x\}$ (with $0 < x < a, b$) and $Q_1 = \mathcal{P} \cup \{x, z\}$ (with $0 < x < a, z$ and $b < z < 1$ but $x \not\leq b$ and $a \not\leq z$).

We should mention here that our Main Theorem has a long and twisted history. It was discussed and claimed, in both directions, by a number of researchers over the past 25 years. The only published proof is in Lerman's monograph [Le10], where he attributes the theorem to Slaman (also see the review by Barmpalias [Ba11]). However (per personal communication with Lerman), the proof published by Lerman [Le10] has a gap, which is filled by a feature which we introduce in our proof here.

We would like to state here the following related question, which we leave open:

Question 1.4. *Is there a "two-sided" strong minimal pair; i.e., are there nonzero c.e. degrees* $\mathbf{a}$ *and* $\mathbf{b}$ *such that* $\mathbf{a} \cap \mathbf{b} = \mathbf{0}$, *for any nonzero c.e. degree* $\mathbf{x} \leq \mathbf{a}$, $\mathbf{b} \cup \mathbf{x} \geq \mathbf{a}$, *and for any nonzero c.e. degree* $\mathbf{y} \leq \mathbf{b}$, $\mathbf{a} \cup \mathbf{y} \geq \mathbf{b}$?

This is, of course, an instance of the Extendibility Question 1.2 (with $n = 3$, combining one lattice-theoretic and two "saturation" obstructions, namely, setting $\mathcal{P} = \{0, a, b, 1\}$ (with incomparable a, b), $Q_0 = \mathcal{P} \cup \{w\}$ (with $0 < w < a, b$), $Q_1 = \mathcal{P} \cup \{x, z\}$ (with $0 < x < a, z$ and $b < z < 1$ but $x \not\leq b$ and $a \not\leq z$), and $Q_2 = \mathcal{P} \cup \{x', z'\}$ (with $0 < x' < b, z'$ and $a < z' < 1$ but $x' \not\leq a$ and $b \not\leq z'$). We remark here that our Question 1.4 has a negative answer if we also require the join of (the images of) a and b to be "branching" (i.e., meet-reducible); i.e., any embedding of $\mathcal{P} = \{0, a, b, c, d, e, 1\}$ (with incomparable a, b, incomparable d, e, and $a, b < c < d, e$) extends to an embedding of $Q_0 = \mathcal{P} \cup \{w\}$ (with $0 < w < a, b$), $Q_1 = \mathcal{P} \cup \{x, z\}$ (with $0 < x < a, z$ and $b < z < 1$ but $x \not\leq b$ and $a \not\leq z$), $Q_2 = \mathcal{P} \cup \{x', z'\}$ (with $0 < x' < b, z'$ and $a < z' < 1$ but $x' \not\leq a$ and $b \not\leq z'$), $Q_3 = \mathcal{P} \cup \{y\}$ (with $a, b < y < c$), or $Q_4 = \mathcal{P} \cup \{y'\}$ (with $c < y' < d, e$). This last result was observed by Slaman by combining Theorem 1.3 with the Non-Embeddability Condition (NEC) of Ambos-Spies and Lerman [AL86]. This last result also suggests that the full answer to our Extendibility Questions 1.1 and 1.2 is likely to be very hard.

2. REQUIREMENTS AND PRIORITY TREE

In this section we describe a set of requirements that guarantee our main theorem, and the way these require-ments can be assigned to strategies on a priority tree. This methodology is rather standard for priority arguments of this type, and the reader is referred to the arguments in [FeSo81, SS93] (Harrington's plus-cupping theorem and Slaman's triple) which exhibit certain similarities. Moreover, these ideas are refinements of certain devices

that were used in Lachlan's original $\mathbf{0}'''$-priority argument in [La75]. We will also refer to these constructions in Section 3, in order to explain the origins of the basic strategies for meeting our requirements.

2.1. **List of requirements.** As usual, we construct two c.e. sets A and B such that in the end $\mathbf{a} = \deg(A)$ and $\mathbf{b} = \deg(B)$. We first have the requirements which satisfy that $\mathbf{a}$ and $\mathbf{b}$ form a strong minimal pair:

$$\mathcal{R}_i : \ \Phi_i(A) = W_i \ \Rightarrow [\exists\Gamma(\Gamma(B \oplus W_i) = A) \vee \exists\Delta(\Delta = W_i)] \,.$$

Then we have the diagonalization requirements which guarantee that A is not below B [B not being below A will be guaranteed automatically]:

$$\mathcal{S}_i : \ \Psi_i(B) \neq A.$$

Note that each $\mathcal{S}_i$ states that there exists an x such that $\Psi_i(B; x) \neq A(x)$. In the construction, each $\mathcal{S}_i$-node has subsidiary $\mathcal{S}_{i,j}$-nodes, each using a possibly different *killing point* (to be defined and clarified later) for forcing $\Psi_i(B; x)$ to diverge. We call a node associated with such $\mathcal{S}_i$ a *parent node* and a node associated with $\mathcal{S}_{i,j}$ a *child node*. At each stage, the collection of an $\mathcal{S}_i$-parent node and its previously visited, uncanceled child nodes is called an $\mathcal{S}_i$-*family* (of that stage).

2.2. **Discussion of the requirements in a historical context.** It is worth noting the similarity of the requirements with those of the arguments in [La75, FeSo81, SS93]. Such a discussion may be beneficial to the reader who is familiar with these older and simpler arguments; but it may also be helpful to the reader who is not an expert in $\mathbf{0}'''$-priority arguments and might like to first consult these simpler proofs. In its simple form, Harrington's plus-cupping theorem (presented in [FeSo81] but also in [Sho90]) asserts the existence of a nonzero degree $\mathbf{a}$ such that every noncomputable $\mathbf{w} \leq \mathbf{a}$ cups to $\mathbf{0}'$ (i.e., there exists some $\mathbf{b} < \mathbf{0}'$ such that $\mathbf{0}' \leq \mathbf{a} \cup \mathbf{b}$). The main requirements for this theorem (excluding the noncomputability of A) can be written as

$$\mathcal{R}_i^* : \ \Phi_i(A) = W_i \ \Rightarrow [\exists\Gamma, B_i \, (\Gamma(B_i \oplus W_i) = A \ \wedge \ \mathbf{0}' \not\leq_T B_i) \vee \exists\Delta(\Delta = W_i)] \,.$$

The similarity of the plus-cupping requirements with our requirements of Section 2.1 is clear. The main difference is that in the plus-cupping requirements, for each W_i we can build a different B_i while in our requirements there is a unique B that must accommodate all conditions. Another relevant example is the construction of a so-called 'Slaman triple', i.e., three degrees $\mathbf{a}, \mathbf{b}, \mathbf{c}$ such that $\mathbf{a} > \mathbf{0}$, $\mathbf{c} \not\leq \mathbf{b}$ and for all noncomputable $\mathbf{w} \leq \mathbf{a}$ we have $\mathbf{c} \leq \mathbf{w} \cup \mathbf{b}$. This was published in [SS93] (based on some unpublished notes of Slaman from 1983) and it is clear that if we also require $\mathbf{a} = \mathbf{c}$ then a Slaman triple becomes the strong minimal pair of our main theorem. The requirements for a Slaman triple (excluding the noncomputability of $\mathbf{a}$ and $\mathbf{c} \not\leq \mathbf{b}$, which is similar to our positive $\mathcal{S}_i$-requirements in Section 2.1) are

$$\mathcal{R}_i^{**} : \ \Phi_i(A) = W_i \ \Rightarrow [\exists\Gamma \, (\Gamma(B \oplus W_i) = C) \vee \exists\Delta(\Delta = W_i)] \,.$$

The similarities of $\mathcal{R}_i^{**}$ with our $\mathcal{R}_i$ are also clear. Instead of using the same set for the roles of A and C we use two, therefore relaxing the conflict that is generated between the positive and the negative requirements. On the other hand, we use a single B here, in contrast with $\mathcal{R}_i^*$, where we had a different B_i for each condition $\mathcal{R}_i^*$.

The strategies used in the arguments in [FeSo81, SS93] involve a gap-cogap technique for the construction of the Turing reductions Γ, which originated in [La75] and which will also be used in our argument. In Section 3, we will discuss this technique in detail, as well as the additional difficulties that conditions $\mathcal{R}_i$ present, which are the reason for the more complicated approach we eventually take.

2.3. **Priority tree.** Our priority tree is defined top down, i.e., the top node has the highest priority. Each node has several possible outcomes, prioritized left to right.

Each $\mathcal{R}_i$-node α has two outcomes: i (infinite) and f (finite). Along the i-outcome, we are defining a functional Γ_α for computing A from $B \oplus W_i$. Such a node α is *active* at some β below if there is no g_α-outcome (see below) between β and α and there are no α' and β' with $\alpha' \subset \alpha \subset \beta' \subset \beta$ such that α' and β' form a pair (see definition below).

Each $\mathcal{S}_i$-parent node β has three outcomes: d (diagonalization), g (gap, defined below), and w (wait). The g-outcome stands for an apparent computation $\Psi_i(B; x) = 0$ against which we cannot diagonalize (i.e., put x into A without risking to lose the computation $\Psi_i(B; x) = 0$). We arrange the priority tree in such a way that immediately following the g-outcome of each $\mathcal{S}_i$-parent node, we have its first $\mathcal{S}_{i,0}$-child node.

Each $\mathcal{S}_{i,j}$-child node β is below the g-outcome of its $\mathcal{S}_i$-parent node and has outcomes $g_{\alpha_0}, \ldots, g_{\alpha_k}, c$ (ordered from left to right). Each g_α (which, following convention, again stands for "gap") corresponds to one active $\mathcal{R}$-node α above the $\mathcal{S}_i$-parent node (*not* the $\mathcal{S}_{i,j}$-child node), ordered in such a way that if $\alpha \subset \alpha'$, then g_α is to the left of $g_{\alpha'}$. For the nodes extending the g_α-outcome, we say that α and this child node β form a *pair*. In addition, we also define a computable function Δ along the g_α-outcome for computing the set W corresponding to the requirement at α. Extending a g_α-outcome, we stop adding new $\mathcal{S}_{i,k}$-child nodes (and believe that this requirement has been satisfied forever). There is only one c-outcome (c stands for "claim") to the right of all the g_α-outcomes. Extending such an outcome, we continue to add new $\mathcal{S}_{i,k}$-child nodes. Of course, we arrange the priority tree in a reasonable way such that along every infinite path, each requirement is represented at most once by a strategy (or pair of strategies, in the case of the $\mathcal{R}$-requirements) which is not enclosed by any other pair.

3. Overview of the Strategies and Their Conflicts

In this section we discuss the basic strategies that are a starting point for the more complex strategies that are needed for the satisfaction of the requirements. We start with the standard gap-cogap strategy for the satisfaction for simple combinations of prioritized conditions, and slowly build the ideas needed for the general case. Recall that the tree of strategies grows from the root downwards, so that a strategy node *above* another is of higher priority with respect to the latter one. The main conflict occurs between the 'positive' requirements (or strategies) $\mathcal{S}_j$ (which typically put numbers into A and try to preserve a B-computation by restraining the enumeration of small numbers into B) and the 'negative' requirements $\mathcal{R}_i$ which typically facilitate the enumeration of numbers into B, which are often needed for the rectification of the functional Γ_i that they build. The latter rectification is needed due to the enumeration of numbers into A by some positive strategies. In the preliminary Sections 3.1 and 3.2, we assume that strategy $\mathcal{S}_j$ operates from a single node, instead of being split into a parent node and child nodes as we described in Section 2.3. We do this for simplicity, as these sections only serve as an illustration of the typical gap-cogap strategy, which is sufficient for simple configurations of requirements but not for the full construction.

3.1. **Typical gap-cogap strategy: one $\mathcal{S}$ below one $\mathcal{R}$.** The strategy of an $\mathcal{R}_i$-node is to simply enumerate Γ_i-computations for the reduction $\Gamma_i(B \oplus W_i) = A$, and enumerate a number into B when there is a number k such that $\Gamma_i(B \oplus W_i; k) = A(k)$. In the latter case, this number would typically be the current use of the computation $\Gamma_i(B \oplus W_i; k)$, and its enumeration facilitates the rectification of the reduction. When k is the witness of some positive requirement (or some related parameter, see below) then the use of the rectified computation may need to be increased to a large number (for reasons that will become clear when we discuss the $\mathcal{S}_j$-strategy).

The $\mathcal{S}_j$ typically picks a witness x and waits for the computation $\Psi_j(B; x)$ to converge with output value 0. If and when this happens, a typical diagonalization strategy would prompt for the enumeration of x into A and the preservation of the B-use of the computation $\Psi_j(B; x)$. However, this naive strategy is not successful in the present context, since the higher-priority $\mathcal{R}_i$-strategy may enumerate into B a number that can destroy the computation $\Psi_j(B; x)$. Such an enumeration may be caused due to the enumeration of x into A by $\mathcal{S}_j$, and the instructions of $\mathcal{R}_i$ to maintain the correctness of Γ_i. This is the primary conflict between the requirements, and at this elementary level it can be resolved by a standard gap-cogap strategy on the behalf of $\mathcal{S}_j$ (much like in the arguments in [FeSo81, SS93] which we discussed in Section 2.2).

The gap-cogap strategy for $\mathcal{S}_j$ typically operates in cycles, periodically restraining A or B, thus building a potential computation Δ for the set W_i. Prior to the start of the alternating cycles, it chooses a witness x. The first step in each cycle is:

(w) Wait for the computation $\Psi_j(B; x)$ to converge with output value 0.

If and when this happens, it checks if the B-use of the computation $\Gamma_i(B \oplus W_i)$ is less than the B-use of the computation $\Psi_j(B; x)$. If this is not true, then it can safely enumerate x into A and restrain the B-use of the computation $\Psi_j(B; x)$, thereby securing the disagreement $\Psi_j(B; x) \neq A(x)$. Note that in this case, the Γ_i-rectification that may be prompted by $\mathcal{R}_i$ will not affect this diagonalization. Otherwise, it will consider the W_i-use of $\Gamma_i(B \oplus W_i; x)$, say u_w, and

(a1) drop any restraint on A (thus allowing W_i to change, under the assumption that $\Phi_i(A) = W_i$);
(a2) define $\Delta = W_i$ up to u_w and restrain enumerations into B up to the B-use of $\Psi_j(B; x)$.

This action initiates an interval of stages that may be called an 'A-gap', which is characterized by a lack of restraint on A and the enforcement of a restraint on B. During this interval of stages, $\mathcal{R}_i$ receives the instruction to increase the use of $\Gamma_i(B \oplus W_i; x)$ to a large number (larger than the use of $\Psi_j(B; x)$) in the event that W_i changes below u_w. When the strategy is revisited (as in a standard tree of strategies argument),

(b1) if W_i has changed below u_w since the stage the A-gap was opened, it enumerates x into A, while enforcing a permanent restraint on B for the preservation of $\Psi_j(B; x) \neq A(x)$;

(b2) otherwise, it closes the A-gap (thereby reinforcing a restraint on A, equal to the use of the current, possibly new computation $\Phi_i(A) = W_i$ up to u_w), and opens a B-gap by dropping the restraint on B and enumerating the B-use of $\Gamma_i(B \oplus W_i; x)$ into B.

Note that step (b2) is possible since $\mathcal{S}_j$ works under the assumption that the reduction $\Phi_i(A) = W_i$ has infinitely many expansionary stages. Moreover, note that the B-enumeration in step (b2) will destroy the computation $\Psi_j(B; x)$. Now let us review the long-term behavior of the $\mathcal{S}_j$. The routine comes to halt if one of the following cases occurs at some stage s_0:

(1) $\Psi_j(B; x)$ remains undefined or not equal to 0 at all stages larger than s_0; or

(2) x is enumerated into A by $\mathcal{S}_j$.

In the first case, $\mathcal{S}_j$ is clearly satisfied (this can be viewed as a Σ_2^0-outcome). In the second case, according to the strategy, the disagreement $\Psi_j(B; x) \neq A(x)$ will be preserved (since the B-use of $\Gamma_i(B \oplus W_i; x)$ would be larger than the B-use of $\Psi_j(B; x)$). Hence in this case also (assuming that basic priority is respected amongst the requirements) $\mathcal{S}_j$ is met in a Σ_2^0-way. The interesting case is when these events do not occur, in which case the following cycle of 'states' of the $\mathcal{S}_j$-strategy repeats indefinitely:

$$(3.1) \qquad\qquad\qquad (w) \to (a1) \to (a2) \to (b2) \to (w) \to \cdots$$

Under this infinitary Π_2^0-scenario, the witness x remains fixed, while the $\mathcal{S}_j$-strategy alternates between A-gap states (when B-restraint is imposed but not A-restraint) and B-gap states (when A-restraint is imposed but not B-restraint). The A-gap interval consists of the steps (w), (a1), (a2) (where the latter two typically occur at the same stage) while the B-gap interval consists of step (b2). In this case, observe that the $\mathcal{S}_j$-strategy builds a total computable function Δ which correctly computes W_i: new computations are produced at the (a2) steps, and throughout the stages none of these computations are falsified. Indeed, if such a computation were falsified (through a W_i-change below the maximum initial segment of numbers on which Δ is defined) then the strategy would execute step (b1), thus ending the perpetual cycle (3.1) and producing a successful Σ_2^0-outcome for $\mathcal{S}_j$. On the other hand, under this outcome, the use of $\Gamma_i(B \oplus W_i; x)$ is driven to infinity, thereby making Γ_i partial at the chosen number x. This aspect of the strategy is sometimes known as 'capricious destruction' of Γ_i, since our strategy intentionally 'kills' the very reduction that we build at a higher-priority node (but for good reasons, see the next paragraph).

Hence, under this infinitary Π_2^0-outcome of $\mathcal{S}_j$ (often called a 'gap outcome'), the actions of this strategy satisfy the higher-priority $\mathcal{R}_i$, as well as itself since the use of $\Psi_j(B; x)$ is driven to infinity. On the other hand, $\mathcal{S}_j$ can pass the information that Γ_i is partial at x to the lower-priority requirements, so a lower $\mathcal{S}_{j'}$ can successfully implement a standard diagonalization strategy by only considering computations $\Psi_{j'}(B; y)$ which have use B-use below the B-use of $\Gamma_i(B; x)$ (which goes monotonically to infinity). In the next section, we see that this gap-cogap strategy also works in a nested environment, thus satisfying $\mathcal{S}_j$ below any finite number of $\mathcal{R}_i$-strategies.

3.2. Typical gap-cogap strategy: one $\mathcal{S}$ below many $\mathcal{R}$.

When an $\mathcal{S}_j$-strategy works below a finite number of $\mathcal{R}_i$-strategies, it needs to resolve the same issues as the ones discussed in Section 3.1, but this time with respect to each of the higher-priority strategies. More specifically, it may have trouble preserving a diagonalization $\Psi_j(B; x) \neq A(x)$ due to a number of Γ-reductions that are being built with higher priority. In this section, we show that a nested version of the strategy we discussed in Section 3.1 suffices to deal with these conflicts. This nesting approach is also typical in arguments like those in [FeSo81, SS93]. For simplicity, suppose that a node working for $\mathcal{S}_0$ is below a node for $\mathcal{R}_1$, which in turn is below a node working for $\mathcal{R}_0$. The methodology we give below generalizes trivially to the case where we have a node for $\mathcal{S}_0$ below nodes for $\mathcal{R}_k, \ldots, \mathcal{R}_0$. The idea is to implement the gap-cogap strategy for $\mathcal{S}_0$ sequentially, first with respect to $\mathcal{R}_1$ and then with respect to $\mathcal{R}_0$.

Consider the gap-cogap strategy of $\mathcal{S}_0$ with respect to $\mathcal{R}_1$. Under the Π_2^0-outcome of this strategy, W_1 is proven computable while Γ_1 is partial at a specified level (namely the witness x of $\mathcal{S}_0$). In this case, another requirement $\mathcal{S}_1$ can work below $\mathcal{S}_0$, with the additional information that Γ_1 is partial at x. Then a standard gap-cogap strategy for the copy of $\mathcal{S}_1$ against $\mathcal{R}_0$ alone can successfully work for satisfaction of both requirements (as in Section 3.1).

On the other hand, there is a possibility that this gap-cogap routine of $\mathcal{S}_0$ against $\mathcal{R}_1$ ends up having a Σ_2^0-outcome. In this case, the strategy would typically go to step (b1). However, at such a stage, $\mathcal{S}_0$ can no longer proceed directly with the diagonalization $\Psi_j(B; x) \neq A(x)$. Indeed, the higher-priority $\mathcal{R}_0$ could potentially destroy such a disagreement (in a way that we have already discussed: through a rectification of its Γ_0 reduction). In this case, $\mathcal{S}_0$ needs to start a new gap-cogap cycle with respect to $\mathcal{R}_0$. If this nested cycle repeats indefinitely, it provides a computation Δ_0 for W_0 while making both Γ_0 and Γ_1 partial at x. In this case, the highest priority $\mathcal{R}_0$ is met, at the expense of $\mathcal{R}_1$ and $\mathcal{S}_0$ which are 'injured' and need to be satisfied by means of additional copies of their strategies under the information that Γ_0 is partial at x. This is certainly possible, as it reduces to the cases we have already discussed. If, on the other hand, the second (nested) gap-cogap cycle of $\mathcal{S}_0$ reaches step (b1), then it can diagonalize, thereby producing the disagreement $\Psi_j(B; x) \neq A(x)$ and preserving it indefinitely (since the relevant Γ_0- and Γ_1-uses are sufficiently large, due to the W_0- and W_1-changes that occurred, respectively).

We may sum up the nesting of the gap-cogap strategies as follows. Strategy $\mathcal{S}_0$ first attempts to 'clear' the computation $\Psi_j(B; x)$ from the Γ_1-use on x. If and when it achieves this (through a W_1-change) it proceeds to clear this computation from the Γ_0-use on x. If and when this is achieved, it can successfully diagonalize. In any other case (except the trivial case when $\Psi_j(B; x)$ remains undefined or not equal to 0), it produces a Π_2^0-outcome that enables copies of the existing strategies to satisfy their corresponding requirements at nodes of lower priority. It is important to note that in the above scenario, after the computation $\Psi_j(B; x)$ is cleared from the Γ_1-use on x, the strategy has one chance to clear it from the Γ_0-use on x (namely, in the next cycle when the A-restraints drop). If this fails, the strategy starts the module anew, waiting again for the convergence of $\Psi_j(B; x)$.

Note that here we have two different Π_2^0-outcomes corresponding to the following cases:

(1) we never clear the Γ_1-use;
(2) we clear the Γ_1-use infinitely often but we never clear the Γ_0-use.

Also note that we only attempt to clear the Γ_0-use when we have already cleared the Γ_1-use. In this sense, we say that $\mathcal{S}_0$ first opens a gap for $\mathcal{R}_1$ and then for $\mathcal{R}_0$.

These nested gap-cogap strategies are sufficient for dealing with one $\mathcal{S}$-strategy below any finite number of $\mathcal{R}$-strategies. When we consider multiple $\mathcal{S}$-strategies below a number of $\mathcal{R}$-strategies, new conflicts occur, which we discuss in the following sections.

Now in our formal construction (see Section 4.1), we instead handle the gap-cogap requirements from different notes by alternating global A-stages and B-stages in the background. During A-stages, we are allowed to change A but not B; during B-stages, we are allowed to change B but not A. Later in the discussion, we will use A-stages and B-stages instead of the gap-cogap terminology.

In particular, in the above construction, we do not need to make enumerations immediately but can wait for an appropriate stage to perform the action. For example, after we enumerated a diagonalization witness x into A during an A-stage, we cannot simultaneously enumerate the Γ-use (for the correction of the Γ functional computing A) into B, but we can do this later when we next time visited the corresponding $\mathcal{R}$ node.

3.3. **A minimal new example: two $\mathcal{S}$ below two $\mathcal{R}$.** Here, we illustrate the idea by a minimal example where we see a conflict which needs some new strategy, and we will briefly explain how to handle the conflict. (See Figure 1.)

First of all, for later purposes, we want to separate a parent node $\mathcal{S}_0$ and its child nodes $\mathcal{S}_{0,j}$. Roughly each child node is taking care of the old strategies which selects the $\mathcal{R}$-requirement above to pair with and defines the corresponding function Δ. The first child node $\mathcal{S}_{0,0}$ is always immediately following its parent node's g-outcome.
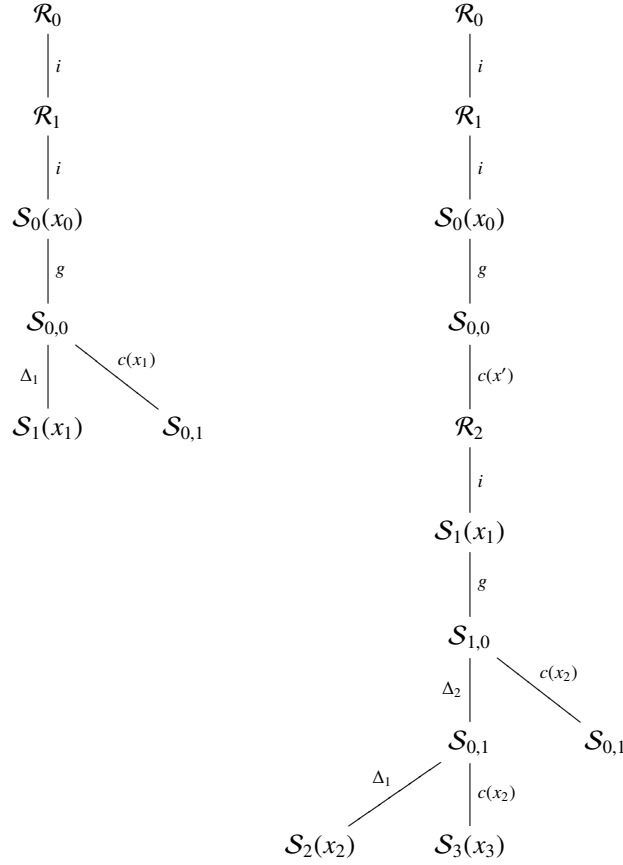
Figure 1. A minimal example (left) and a complete example (right)

Let $\mathcal{R}_0$ and $\mathcal{R}_1$ be two consecutive $\mathcal{R}$-requirements, and let the $\mathcal{R}_1$-node be extending the $\mathcal{R}_0$-node's i-outcome. Consider an $\mathcal{S}_0$-node extending the $\mathcal{R}_1$-node's i-outcome. Now, at the $\mathcal{S}_0$-node, as in a usual construction of this type, we may have a diagonalization witness x_0, but the use $\psi_0(x_0)$ may always be too large (say, $\geq \gamma_1(x_0)$), and so we go to the g-outcome. At the first $\mathcal{S}_{0,0}$-child node, we use $\gamma_1(x_0)$ to kill the computation $\Psi_0(B; x_0)$ infinitely often, say. At the same time, the $\mathcal{S}_{0,0}$-child node will build a function Δ_1 to correctly compute W_1 (for the $\mathcal{R}_1$-node).

Now, to make sure that Δ_1 is always correct, the $\mathcal{S}_{0,0}$-child node has to set up some mechanism to prevent injury. In the construction, we implement an alternating A-stage/B-stage approach, so that at each stage, at most one A or B can change. There are now two cases here. During a B-stage, A does not change, and so $W_1 = \Phi_1(A)$ (up to the length of agreement) will not change, either, since otherwise, we will not visit the $\mathcal{S}_0$-node again. During an A-stage, A can change but B does not. If now W_1 changes, then we can increase the Γ_1-use while preserving the $\Psi_0(B; x_0)$-computation. Then we observe that $\gamma_1(x_0) > \psi_0(x_0)$, and so we will switch to the left of the outcome associated with Δ_1. In this process, unless we move to the left of the outcome associated with Δ_1, we see that the $\Psi_0(B; x_0)$-computation is used to protect Δ_1 during A-stages, since only a W_1-change without a B-change guarantees that we can move to the left of the outcome associated with Δ_1; so, in the argument, it is crucial that we can preserve the use of $\Psi_0(B; x_0)$.

Now, say, extending the Δ_1-outcome, we have another $\mathcal{S}_1$-node with a witness x_1. During an A-stage s_0, it might want to enumerate x_1 into A for its own diagonalization (and so A would be changed). By the observation above, we have to protect the use of $\Psi_0(B; x_0)$ at the same time. However, if we implement the diagonalization procedure for $\mathcal{S}_1$ here, then later at $s_1 > s_0$ the $\mathcal{R}_0$-node's Γ_0-functional, after observing a change at x_1 in A, will inevitably add $\gamma_0(x_1)$ into B for Γ_0-correction (unless W_0 has changed from s_0 to s_1, but this is not in our control). At s_1, however, there is no guarantee that $\gamma_0(x_1) > \psi_0(x_0)$.

The solution is thus briefly as follows: In such a situation at stage s_0, we instead go to a different outcome to the right of the Δ_1-outcome, which we call the c-outcome. We stay at this c outcome as long as $\gamma_0(x_1) \leq \psi_0(x_0)$ (since otherwise, there is no problem). So at a following stage $s_1 > s_0$, as long as $\gamma_0(x_1) \leq \psi_0(x_0)$ is still true, instead of using $\gamma_0(x_0)$ to kill the $\Psi_0(B; x_0)$-computation, we can use $\gamma_0(x_1)$. We say that x_1 is the *claim point* for this c-outcome at this stage s_0. We count this as a small step toward success. Later at the next $\mathcal{S}_{0,1}$-child node, we have a similar scenario for which we may go to the c-outcome with a larger claim point, etc. If this happens infinitely often along the true path (i.e., there are infinitely many $\mathcal{S}_{0,j}$-nodes with a c-outcome along the true path), then we are using larger and larger numbers to push $\psi_0(x_0)$ to infinity, and so the $\mathcal{S}_0$-requirement will be satisfied in a Π_3-way; on the other hand, Γ_1 is still active (since it is only injured finitely often at each argument), so we do not have to build Δ_1 for it.

From a local viewpoint, the conflict happens when we see a computation at $\mathcal{S}_1$ which we want to use to diagonalize, and some higher requirements $(\mathcal{R}_1 - \mathcal{S}_0)$ put some restraint on the diagonalization. So the c-outcome with a larger claim point x_1 essentially allows us to freeze the computation at $\mathcal{S}_1$ and at the same time allow $\mathcal{R}_1$ and $\mathcal{S}_0$ to continue working towards success by switching the killing point from x_0 to x_1.

From a global viewpoint, while other outcomes are standard in this type of gap-cogap construction, each such c-outcome is a Σ_2-type of outcome, which states that in the construction, there is a stage with a claim point such that we will keep this claim point (stay in the c-outcome) forever in the following construction.

Node	Symbol	Access	Action	Sub-action	Outcomes	Type
$\mathcal{R}$	α	normal	defines Γ	B-enumerations	i, f	Π_2^0 / Σ_2^0
$\mathcal{S}$-parent	β	normal or child-link	clearing/claim	A-enumeration	d, g, w	$\Sigma_1^0 / \Pi_2^0 / \Sigma_2^0$
$\mathcal{S}$-child	β_j	n. or own-parent-link	defines Δ	B-enumerations	g_{α_t}, c	Π_2^0 / Σ_2^0

Table 1. Nodes on the priority tree, their main actions and their outcomes

3.4. **The new idea: c-outcomes.** The use of the c-outcomes is new and in fact crucial to our construction, so it is important to explain its use and address the differences between a c-outcome and a standard g-outcome (gap outcome) for example as in Section 3.1.

As we have mentioned above, the c-outcome in our minimal example essentially allows us to freeze the computation at the $\mathcal{S}_1$ node (see Figure 1) as well as Δ_1 to the left of the c-outcome, while waiting for a later stage when diagonalization is safe to perform (i.e., the Γ_1-use is large enough). It is important that here we do not perform any enumeration at this c-outcome. A natural attempt, which actually fails to work, would be to perform the same gap-cogap operation with the new witness x_1 at the c-outcome. The reason is that, it is possible that such a witness x_0 or x_1 at which the Γ_1-use is used to push $\Psi_0(x_0)$-use may change and possibly go to infinity. All the lower priority nodes, for a successful construction, need to guess at the outcome correctly. However, with only one (or even infinitely many) c-outcome where the gap-cogap strategy is performed, it is not possible for the lower priority nodes to know whether the witness will stop increase or go to infinity.

In general, a c-outcome is an outcome of a $\mathcal{S}$-child node, but unlike a g-outcome (of the same child node) it does not enumerate any elements (into B). Instead such enumeration is delayed to g-outcomes of other $\mathcal{S}$-children nodes below this c-outcome.

Along the c-outcome, the $\mathcal{S}$ requirement at the parent node is satisfied as we push its use $\Psi(x)$ to infinity for a fixed x (the diagonalization witness at the parent node). This is the same as along the g-outcomes to the left of the c-outcome. What makes a c-outcome different is that the delay of B-enumerations allows us to satisfy the requirement $\mathcal{R}$ by keeping the corresponding Γ total. Note that g-outcomes always kill such Γ functional and so they need to build Δ's in order to satisfy $\mathcal{R}$.

In addition, such delay also allows us to work on other (lower priority) requirements between the $\mathcal{S}$-children nodes. That is, if a lower priority node is only below a c-outcome of the "$\mathcal{S}$-family" but not any of the g-outcomes of the children nodes, then it believes that Γ is still total and so active. As a result, this also requires

some minor adjustments to the different numbers used in our standard gap-cogap construction, which we will describe in the complete example below.

3.5. A complete example: clearing point, killing point and claim point.

Now we complete the minimal example above to add in all the features of the construction. In particular, we will explain various numbers used during the construction. (See again Figure 1.) Table 1 can be helpful as a guide on the general structure of the argument and the complexity of the outcomes of the strategies.

Suppose we have two consecutive $\mathcal{R}$-requirements $\mathcal{R}_0$ and $\mathcal{R}_1$, and the $\mathcal{R}_1$-node is extending the $\mathcal{R}_0$-node's i-outcome. Extending the $\mathcal{R}_1$-node's i-outcome, we again have an $\mathcal{S}_0$-node followed by its first $\mathcal{S}_{0,0}$-child node. Now, extending the c-outcome of the $\mathcal{S}_{0,0}$-node (with a claim point x'), we have an $\mathcal{R}_2$-node followed (along its i-outcome) by an $\mathcal{S}_1$-node.

Let x_1 be the diagonalization witness for $\mathcal{S}_1$. When we try to diagonalize against Ψ_1 at stage s_0 when we see a convergent computation $\Psi_1(x_1)$, we first need to make sure that $\psi_1(x_1) < \gamma_i(x_1)$ for $i = 0, 1, 2$; in addition, notice that the $\mathcal{S}_0$-family currently has a c-outcome, which means that extending any outcome of the $\mathcal{S}_1$-node (e.g., the d-outcome), there will be more $\mathcal{S}_{0,j}$-child nodes, and later at any stage $s > s_0$ they will possibly enumerate $\gamma_0(x')$ or $\gamma_1(x')$ into B in order to push $\psi_0(B; x_0)$ to infinity. This means that, for successful diagonalization against Ψ_1, we also need to care about possible Γ-use enumerations at x' (which is $< x_1$). So here at stage s_0 we call such a number x' the *clearing point* at $\mathcal{S}_1$ and use it to clear the computation: For clearance, we require $\psi_1(x_1) < \gamma_i(x')$ for $i = 0, 1, 2$. If this is not true, then we use x' (instead of x_1) to push $\psi_1(B; x_1)$ at the first $\mathcal{S}_{1,0}$-child node.

Say, at the $\mathcal{S}_{1,0}$-node, we choose to go along the g_{α_2}-outcome building Δ_2 (since $\psi_1(x_1) \geq \gamma_2(x')$). Now extending this g_{α_2}-outcome, say, we first have an $\mathcal{S}_{0,1}$-child node. As required by the $\mathcal{S}_{0,0}$-node, the $\mathcal{S}_{0,1}$-node uses x' to push $\psi_0(B; x_0)$ to infinity. We call such a number x' the *killing point* at $\mathcal{S}_{0,1}$. Say after stage $s_1 > s_0$, such a child node also has a c-outcome (whose claim point x_2 comes from some $\mathcal{S}_2$-node extending one of its g-outcomes, as in the minimal example). Extending such a c-outcome, we have an $\mathcal{S}_3$-parent node, say with diagonalization witness x_3.

From the $\mathcal{S}_3$-node's point of view, $\mathcal{R}_2$ has been satisfied (by the $\mathcal{S}_{1,0}$-node), and $\mathcal{R}_0$ and $\mathcal{R}_1$ are still active. The clearing point at the $\mathcal{S}_3$-node is x_2, because it believes that the new $\mathcal{S}_0$-family members will use x_2 instead of x' as the killing point. So the $\mathcal{S}_3$-node checks whether $\psi_3(x_3) < \gamma_i(x_2)$ for $i = 0, 1$.

Now suppose this is true, i.e., we have a cleared computation, say at stage $s_2 > s_1$. Then, according to the minimal example above, we next want to make sure that Δ_2 is preserved, and we try to clear the $\Psi_1(B; x_1)$-computation by going to the c-outcome of the $\mathcal{S}_{1,0}$-node.

The tricky part is that, this time at stage s_1, for successful clearance, we actually want $\psi_1(x_1) < \gamma_i(x_2)$ (for $i = 0, 1$) (instead of $\psi_1(x_1) < \gamma_i(x_3)$): The reason here is that, to the right of this Δ_2, later at any stage $s > s_2$ it is possible that a new $\mathcal{S}_{0,j}$-child node will use x_2 as the killing point and enumerate $\gamma_0(x_2)$ or $\gamma_1(x_2)$ into B, and we do not want these numbers to injure $\Psi_1(B; x_1)$, which we use to protect Δ_2. We say that x_2 is the *claim point* of this c-outcome at stage s_2 (later this claim point is used as the killing point for new $\mathcal{S}_{1,k}$-child nodes). When we go to the c-outcome, i.e., the $\Psi_1(B; x_1)$-computation is not cleared, then the associated *claim* here is that after this stage, it is always the case that we do not get a clearance, i.e., it is always the case that $\psi_1(x_1) \geq \gamma_i(x_2)$ for $i = 0$ or 1.

Point	$\mathcal{S}$-node	Outcome	Complexity
Witness	Parent	All	Σ_1^0
Clearing	Parent	All	Σ_1^0
Claim	Child	c	Π_2^0
Killing	Child	c	Π_2^0

Table 2. Parameters of the $\mathcal{S}$-nodes (parents and children), associated outcomes and their complexity modulo initialization.

3.6. **Overview of the $\mathcal{S}$-strategies.** Table 2 summarizes the parameters we have introduced for the $\mathcal{S}$-nodes (parents and children). In this section, we summarize their dynamics and basic features, in a top-down description (as opposed to the bottom-up motivational discussion of Section 3.5). The diagonalization is done at the parent node, with a witness which is fixed, as long as the parent node is not injured. The same is true of the *clearing point*, which is another parameter of the parent node. The clearing point is always less than or equal to the witness. In the simple case that we described in Section 3.3, we use the witness as a clearing point, but in the presence of more requirements, we need to differentiate between the two. The clearing point is the number on which we may force the associated Γ-functional to be partial.

Associated with the c-outcome of each $\mathcal{S}_{ij}$-child node is the *claim point* of the node. Each time that the c-outcome is activated, it may have a different claim point. Each $\mathcal{S}_{ij}$-child node also has a *killing point*, which is calculated from the claim points of the higher-priority child nodes. In this way, the killing points of child nodes are raised according to the claim points of the higher-priority child nodes with c-outcomes. The c-outcome of a child node β_j is initiated by a parent node below β_j (not its own parent).

Satisfaction of $\mathcal{S}$	Main outcome	Outcome	Complexity
$\Psi(B; x) \uparrow$ co-finitely	wait outcome (parent)	Γ total	Σ_2^0
$\Psi(B; x) \downarrow \neq A(x)$ co-finitely	diagonalization (parent)	Γ total	Σ_2^0
$\Psi(B; x) \uparrow$ infinitely often	gap outcome (child)	Γ partial	Σ_3^0
$\Psi(B; x) \uparrow$ infinitely often	all children true c-outcomes	Γ total	Π_3^0

Table 3. Four different ways that requirement $\mathcal{S}$ with witness x may be satisfied, and their complexity relative to the corresponding parent node.

Along with the c-outcome, an $\mathcal{S}_{ij}$-child node implements a gap-cogap strategy, sequentially with respect to the Δ-functionals of higher-priority child nodes. This gap module looks for appropriate changes in the approximation to the corresponding sets W, starting from the closest and moving monotonically toward the root of the tree. The usual gap-cogap operation of a child node may be interrupted by its c-outcome infinitely often. Infinitely many c-outcomes along the child nodes of a parent node (in the 'true path') means that the functional we try to diagonalize against is partial. Table 3 displays all the different ways that requirement $\mathcal{S}$ can be satisfied. The first three ways displayed are typical to a gap-cogap argument. However, the last case is special and corresponds to the case when all children fail to succeed with their gap-cogap strategy. In that case, $\Psi(B; x)$ becomes partial due to the enumeration of Γ-uses on larger and larger arguments. Table 3 also displays the effect that the outcomes have on the functional Γ that we build for $\mathcal{S}$. Note that in the context of the global construction, where many requirements are present, the global outcomes are slightly more complex (e.g. a Γ-functional that is left intact by some child node may end up partial due to a child of another parent).

4. Construction

4.1. **Accessible path, stage dichotomy, accessible nodes and visited nodes.** In the construction, each stage is either an A-stage or a B-stage. We can arrange that all even stages are A-stages and all odd stages are B-stages. During A-stages, we are allowed to change A but not B; during B-stages, we are allowed to change B but not A. Each node first ignores the stage setting and follows the construction. When the node wants to change A or B, it checks whether the current stage setting allows this action. If so, it changes A or B as planned; if not, it terminates the stage and waits.

In addition, each node must try to pass down alternating A-stages and B-stages along its (believed) true outcome. If the stage setting is not the one expected, the node needs to wait for another stage to go to the outcome we want. For instance, if a node needs to go to an outcome, and at the last stage that outcome was accessible was an A-stage, then we are expecting a B-stage this time. If this is a B-stage, then there is no problem; if this is an A-stage, then we terminate the stage.

Now, in these two cases when we terminate the stage (since the stage is not the one we wanted), at the very next stage (notice that the stage has changed from A to B or from B to A), we first check whether any W

has changed (from the previous stage) for those W's along the accessible path, up to the previous length of agreement. If so, then for the highest one, we switch to the f-outcome if the length of agreement has decreased (and it is easy to see that then we have a permanent win unless the node is initialized), or to the i-outcome if the length of agreement increased (and so we switch to the left if we went to the f-outcome at the previous stage). Otherwise (if there is no W-change, or the length of agreement does not change, or the length of agreement has increased and we went to the i-outcome at the previous stage), then we directly go through the same accessible path and continue the construction at the node where we terminated the stage.[1] So either we can change A or B as planned, or we can go to the outcome we wanted. In other words, at each node, if the last stage was a terminated stage and there is no W-change, then we continue to the same outcome without any extra action.

As in a usual priority tree construction, at each stage s, we inductively construct an *accessible* path (up to length s) on the priority tree. At each node along the accessible path, we try to decide the outcome at stage s and whether we want to change A or B. Whenever A or B is changed, we terminate the current stage and go to the next stage. We keep the nodes that are to the left of, or compatible with, the accessible path and initialize the nodes that are to the right. Note that we may build a link in the construction and skip some nodes along the accessible path (without going through the construction for them at that stage). So we shall distinguish between notions of a node being visited and being accessible. Being *visited* means that we allow this node to act according to the construction below; and being *accessible* only means that the node is on the accessible path, which does not necessarily mean that the node itself is visited but possibly only some extension of it is.

In the following subsections, we always assume that we are at a visited node at stage s.

4.2. $\mathcal{R}$-node.

Consider an $\mathcal{R}$-node α and note that if the last stage was a terminated stage and W has not changed, then we continue to the same outcome without any action. Otherwise, we check whether the *length of agreement* has increased since the last stage t when we visited this node and the i-outcome was accessible (or if such a stage t does not exist, then we check whether the length of agreement is positive). If not, then we go to the f-outcome. If so, then we go to the i-outcome.

The $\mathcal{R}$-node α also defines a functional Γ along the i-outcome. We make sure that Γ is *well-defined*, i.e., we will not enumerate axioms that use the same oracle but give different outputs. In particular, we may have some *requests* to add some numbers into B here which were assigned by nodes below. What we do is simply put these numbers into B as planned if the corresponding W has not yet changed (see Section 4.3.1).

For convenience, we allow the W-use and B-use for the same x to be different (so we formally write $\gamma(W; x)$ and $\gamma(B; x)$ to denote these uses, but later, when it is clear from the context that we are talking about the B-use, we will simply write $\gamma(x)$). Since all the sets we consider are c.e., at each stage we only need to keep one axiom $\Gamma(B \oplus W; x)$ for a fixed x. We have two cases in which we increase the use. The first case is that some node below puts $\gamma(B; x)$ into B but $A(x) = 0$; in this case, we increase the B-use to be large and fresh, and increase the W-use to be the length of agreement between $\Phi(A)$ and W at this stage. The second case is when the W-use changes; then we increase the B-use to be large and fresh and keep the W-use the same. In all other cases, we do not increase the uses but simply update the axiom with the current oracle.

Of course, we obey the usual monotonicity rules of axioms, that is, whenever we change the uses for some x, we automatically make $\Gamma(B \oplus W; y)$ undefined for all $y > x$. In any case, we will ensure that $\Gamma(B \oplus W; x) = A(x)$ for all $x \le$ the current length of agreement between $\Phi(A)$ and W at this stage; if a use for $\Gamma(B \oplus W; x)$ had never been picked before, then we pick the B-use large and fresh, and the W-use to be the current length of agreement between W and $\Phi(A)$; otherwise, the use is specified as above.

4.3. $\mathcal{S}$-parent node.

At an $\mathcal{S}_i$-node β, if this is the first time at which we visit this node, then we pick a fresh diagonalization witness x for it. Now if we already have a diagonalization witness x, then we check whether $\Psi_i(B; x)$ converges to 0 with a *believable* computation. Here, and in the following, a computation $\Psi_i(B; x)[s] \downarrow$ is believable when there are no numbers below the use of this computation that may enter B at a later stage, by the nodes above β (such are uses of Γ-functionals above β that are partial from the point of view of β). If not, then we go to the w-outcome and continue to the next node. If we find out that earlier we have already visited the d-outcome (i.e., we have already performed diagonalization at this node and $A(x) = 1$). and β has not been initialized since, then we continue to go to the d-outcome.

---

[1] The intuition is that, since no one has changed A or B from the last stage, and the W's have not changed, either, unless we can diagonalize, all the uses of computations remain the same. (See Lemma 5.1 for the full proof later on.)

If there is such a believable computation $\Psi_i(B; x) \downarrow= 0$ (where, when we see a believable such computation, we immediately initialize every node extending the w-outcome) but we have not yet performed diagonalization (i.e., enumerated x into A), then we perform the following construction. We first check whether we can perform diagonalization (see below in Section 4.3.1) and if so, follow the instructions; if not, then we go to the g-outcome (or some other outcomes according to Section 4.3.2 below) and continue to the next node.

4.3.1. *Diagonalization, setting clearing and claim points.* At β, we consider those $\mathcal{S}_{i'}$-requirements which have g-outcome along β and none of whose child nodes has a g-outcome along β. We think of the $\mathcal{S}_{i'}$-family as a whole as announcing the current *killing point for the requirement* $\mathcal{S}_{i'}$, which is defined as the greatest number among all claim points of all $\mathcal{S}_{i'}$-child nodes above or to the left of β as well as the clearing point at $\mathcal{S}_{i'}$. Then we let the *clearing point y at β* be the least of these killing points announced by the $\mathcal{S}_{i'}$-families from above as well as x (if there is no such higher-priority $\mathcal{S}_{i'}$).[2]

We check whether $\gamma_k(y) > \psi_i(x)$ (for the clearing point y defined above) for each active $\mathcal{R}_k$ above. If not, then we go down to the g-outcome here (see Section 4.3.2) and, at the first $\mathcal{S}_i$-child node $\mathcal{S}_{i,0}$, we will go to the corresponding g_{α_k}-outcome defining a function Δ and add $\gamma_k(y)$ into B there (for the greatest such k, see details below in Section 4.4). If $\gamma_k(y) > \psi_i(x)$, then we proceed to the following check.[3]

Here, it is possible that for some other Δ' defined at an $\mathcal{S}_{i'}$-child node β' above β (along the same path), we use the corresponding $\Psi_{i'}(B; x')$-computation to protect Δ', yet some $\gamma_k(x)$ entering B for Γ_k above this $\mathcal{S}_{i'}$-node may cause injury, i.e., $\gamma_k(x) \leq \psi_{i'}(x')$.

If there is no such β', i.e., for every β' along β, we have $\gamma_k(x) > \psi_{i'}(x')$ as above, then we can put x into A and go to the d-outcome of β. While doing that, we issue requests at each active $\mathcal{R}$-node above β to add $\gamma(x)$ into B as follows: Later when we visit $\mathcal{R}$'s i-outcome, if the corresponding W-use (for $\Gamma(B \oplus W; x)$) has changed, then we do not add $\gamma(x)$ into B, but otherwise, we add $\gamma(x)$ into B.

If we see such β', then fix the lowest (i.e., we process these nodes from the bottom up) such β' for which $\gamma_k(x) \leq \psi_{i'}(x')$, we consider all $\mathcal{S}_{i''}$-nodes above β' which have a g-outcome along β' but such that no child node has a g-type outcome along β' (i.e., the $\mathcal{S}_{i''}$-requirements that are still active at β'). For each such $\mathcal{S}_{i''}$-node, we only look at its child nodes below β' (the $\mathcal{S}_{i''}$-*family below β'*). These child nodes define a current killing point, i.e., the maximum claim point (if such $\mathcal{S}_{i''}$-family below β' is empty, then let this current killing point be infinity). Then we let the *claim point z of β* be the minimum number among all these killing points of $\mathcal{S}_{i''}$-families below β' (for all such β'), as well as x, the diagonalization witness at β. So automatically z is less than or equal to x.[4]

This c-outcome at β' is now associated with the *claim* that "after this stage s, it is always the case that $\psi'(x')$ is greater than or equal to $\gamma_k(z)$ for some active Γ_k above the $\mathcal{S}_{i'}$-parent node". (For convenience we denote this claim by $C(\beta', z, s)$.) In addition, this c-outcome announces that z is the new killing point for lower-priority $\mathcal{S}_{i'}$-child nodes, overwriting the old announcements made by higher-priority child nodes for the same $\mathcal{S}_{i'}$. That is, $\mathcal{S}_{i'}$, as a whole requirement, now switches the killing point to z. In this case, we say that β *initiates* the c-outcome at β'.[5] We go to the c-outcome of β' and continue to the next node along that path.

4.3.2. *Possible link to child.* Now, at this time, if we do not have a chance to diagonalize, there might be some $\mathcal{S}_{i'}$-child nodes below, whose c-outcome has been initiated with a claim about the size of $\psi_{i'}(x')$ and some Γ-uses of possibly larger x'' (see above). We check if any of these claims turn out to be false. For those corresponding c-outcomes whose claims turn out to be false, we initialize everything below the c-outcome of these child nodes and everything to the right of them.

---

[2]Since x is a fresh number when it is picked, this y is always less than or equal to x (Lemma 5.8). Roughly speaking, this y is going to be the least killing point when we go to the right of β, and so for successful diagonalization, we want to make sure that β's computation is protected when we switch to the right of it. In the complete example in Section 3.5, our x here is x_3 there, and our y here is x_2 there.

[3]If so, note that $y \leq x$, so it is automatic that $\gamma_k(x) \geq \gamma_k(y) > \psi_i(x)$ and it seems that we are safe to put x into A.

[4]Later we will see that it is automatically greater than the killing point at β' (Lemma 5.9). In the complete example in Section 3.5, our z here happens to be x_2 there as well, just like our y here is x_2 there, but this need not be true in general.

[5]Later, when we reach the parent node for β', we can check whether the condition $\gamma_k(z) \leq \psi'(x')$ is still true, i.e., whether this claim is still true; if not, then we will initialize everything extending the c-outcome at β' and declare that this node β' now gives permission for diagonalization at β.

In addition, we check whether there is an $S_{i,j}$-child node such that the last time it was visited we went to one of its g-outcomes, and now with the current conditions we see that we can switch to the left to that g-outcome. If there is such a child node, then we build a link directly from the S_i-parent node β to that child node, skipping every node between them. Otherwise, we stay at the S_i-parent node β and proceed to the next node along the g-outcome.

4.4. S-child node.

When we reach an $S_{i,j}$-child node β_j of an S_i-node β, the construction proceeds as follows. First, as we have mentioned above, β_j checks whether the c-outcome was accessible at the last stage t when we visited β_j. If so, we check if the associated claim $C(\beta_j, z, t)$ is still true. In that case, we go down to that outcome without doing anything here. If the claim is false, then we have already initialized everything extending the c-outcome of β_j when we reach β. In that case, there must be some S-parent note β'' below some g_α-outcome of β_j which initiated the c-outcome of β_j here. If this node β'' has not been initialized since, then we directly link to this β'', allowing it to finish trying its diagonalization (without visiting the nodes between β_j and β''). If this β'' has already been initialized, then we proceed as in the following paragraph.

Otherwise, i.e., if we didn't visit the c-outcome the last time we visited β_j, then we have a *killing point y* here decided by higher-priority $S_{i,j'}$-child nodes $\beta_{j'}$ above or to the left of β_j (or by β itself if there is no such $\beta_{j'}$): y is the largest of all the claim points of these $\beta_{j'}$ as well as the clearing point at β. We also know that $\gamma(y) \leq \psi_i(x)$ for some functional Γ by some active $\mathcal{R}$-node above β; let α be the lowest-priority such $\mathcal{R}$-node. Now we go to the g_α-outcome. If this is a B-stage, we also add $\gamma(y)$ into B. For the functional Δ associated with the g_α-outcome, we extend Δ up to the W-use $\gamma(W; y)$. Then we continue to the next node, this finishes the inductive step of the accessible path construction.

5. Verification

We start with a few technical lemmas, then we can show that there is a leftmost path accessible infinitely often (the *true path*) and every node on the true path has a true outcome. We then show that all the functionals Γ (unless killed) and all functions Δ built along the true path are well-defined. This allows us to show that all requirements are satisfied.

5.1. Technical lemmas.

First of all, in our construction, we separated the stages into A-stages and B-stages, and only allowed changes in A or B at A-stages or B-stages, respectively. Sometimes, we may encounter the situation that the algorithm wants to change A but the current stage is a B-stage, or vice versa, and so in the construction, we simply terminate the stage and immediately try the next stage. (See Section 4.1 for details.) We start with a lemma proving that in this case, either we will change the accessible path due to a W-change (which will cause either initialization of the node that wanted to enumerate, or the permanent satisfaction of the requirement of a higher-priority node), or we can perform the desired B- or A-enumeration at the next stage.

Lemma 5.1 (Accessibility of A/B-stages). *Suppose at stage s, we terminated the stage because the stage was not of the type we wanted. Then at the next stage $s + 1$, either some W changes and we switch to the left or right of the accessible path at stage s, or we can perform the enumeration we wanted to perform at stage s.*

Proof. According to the construction, assume that some W along the accessible path (of stage s) changes at stage $s + 1$ by x entering W: If this change decreases the length of agreement between W and $\Phi(A)$ and switches the outcome of a strategy along the accessible path at stage s from an i-outcome to an f-outcome, then we have permanent satisfaction of an $\mathcal{R}$-requirement (unless some higher-priority node acts), since $W(x) = 1$ and we have a computation $\Phi(A; x) = 0$. If this change increases the length of agreement or does not change it, then actually it will not affect any of the Δ's previously defined below the i-outcome (since we only define Δ up to the length of agreement). Now, if we do not switch the accessible path between stages s and $s + 1$, then obviously, since we have not changed A or B from stage s to stage $s + 1$, all criteria required for action remain the same, and we can perform the action (go to a certain outcome or change A or B) as at the previous stage s. $\square$

Usually, in a priority tree argument, one can simply see by inspection that, for any computation (e.g., of Ψ, Φ) witnessed at a node, the use cannot be changed by any node to the right of it (by the choice of sufficiently large witnesses). However, in our construction, this is not true. The problem is that, along a c-outcome of an S-child node, the killing point z is determined by some node extending a g_α-outcome of the S-child node,

i.e., to the left of its c-outcome. Therefore, potentially any B-change up to $\gamma(z)$ via at a node extending the c-outcome might injure some Ψ-computations to the left of it. So we need a lemma stating that, in certain cases, such injury cannot happen.

Lemma 5.2 (Link to a parent node). *In the construction, if we see that a claim for a c-outcome at some $\mathcal{S}_{i,j}$-node β becomes false and build a link to an $\mathcal{S}_{i'}$-node β' along a g_α-outcome (which initiated the c-outcome), then at that time, the computation at β' is still the same as when β' initiated the c-outcome.*

Proof. Say, at stage s_0, β' initiated the c-outcome and by the criterion in the construction, we know that the use $\psi(x)$ at β' (for the diagonalization witness x at β') is $\leq \gamma(y)$ for the least possible killing point y that can be used to the right of β'. If such y in the definition decreases (i.e., some node to the right uses a smaller number as the killing point), then we would have initialized β' and would not build a link from β. This means that when we build a link back to β', its computation is preserved.   □

Lemma 5.3 (Diagonalization of parent preserved). *If an $\mathcal{S}$-node has performed diagonalization, then unless it is initialized, its computation $\Psi(B;x)$ is always preserved.*

Proof. The argument is almost the same as the previous lemma. If a killing point y had decreased, then it would mean that the node had been initialized. If the killing point has not decreased, then by our criterion, the computation is preserved.   □

5.2. **True path lemmas.** Since our tree is finitely branching, there clearly is a leftmost path accessible infinitely often (which we call the *true path*). The slightly tricky problem is that in the construction, there are two cases when we build a link between two nodes and skip nodes in between: The *first* case is when an $\mathcal{S}_i$-node sees that an $\mathcal{S}_{i,j}$-child node can now switch to the left; the *second* is from a c-outcome of an $\mathcal{S}_{i,j}$-node to an $\mathcal{S}_{i'}$-node below one of its g_α-outcomes. It is conceivable that some node on the true path is skipped infinitely often but not visited infinitely often, or its outcome is along the true path but is actually not the *true outcome* (the leftmost outcome we choose infinitely often when visiting the node). The following few lemmas show that this case cannot happen. The idea to prove this is as follows: Each time we skip over a node β, we always "blame" a node below it and make sure that such a node can only do this finitely often before β is visited again.

Lemma 5.4 (First case skip). *If a node β is skipped via the first case, then some node below it switches left. In addition, if β is never visited again and never skipped by the second case, then the skip for the first case can only happen finitely often, and each time we will go strictly to the left of the previous visit.*

Proof. The first claim follows by inspection of the construction. For the second claim, note that for every such link which skips β, β must be between an $\mathcal{S}$-node and one of its child nodes. A somewhat tricky situation may arise that during such a stage when β is skipped, we may add new nodes below it which may cause extra links. But observe that such a new link must be associated with an $\mathcal{S}'$-parent node of higher priority than the $\mathcal{S}$-node which causes the skip at the current stage, so by induction on the number of $\mathcal{S}$-parent nodes above β, one can see that, if β is never visited again, such a skip (for the first case) can only happen finitely often. More precisely, we associate each skip to a combination of $\mathcal{S}$- and $\mathcal{R}$-nodes of higher priority than β, and assign a natural priority on these combinations. It is then easy to check that each time we go to the left, such a combination increases in priority, and so this cannot happen forever.   □

Lemma 5.5 (Second case skip). *At any stage, for any given β, there can be at most one node β' below β which has initiated a c-outcome at a node above β such that the associated claim is still true. That is, during any fixed stage, there can be at most one node which makes us skip β for the second case.*

Proof. Suppose s_0 is the first stage such that the c-outcome of β' is initiated. Then, of course, at stage s_0, there is only one such node (we jump to the c-outcome at s_0). After that, either β' is initialized; or the associated claim never becomes false, and so the claim of the lemma remains true; or later the claim becomes false at stage s_1 and we build a link directly to β' skipping β. At that stage, we note that the computation at β' is still the same as that at stage s_0 (by Lemma 5.2). So at stage s_1, either β' again initiates another c-outcome even higher, or it follows diagonalization and now there are no nodes which make us skip β (for the second case). The same situation happens at every stage afterwards, and so the lemma follows.   □

Lemma 5.6 (True path). *Along the true path, every node is visited infinitely often, therefore all outcomes along the true path are true outcomes.*

Proof. This follows essentially by combining Lemmas 5.4 and 5.5. Suppose some β on the true path is never visited again. Whenever we skip β via the second case, then some node below performs diagonalization, which means that any nodes extending the d-outcome will be fresh at that stage. At that moment, the only reason we can skip β is the first case, and so the next time we skip over β, we must travel to the left of the current visit. It then follows that below any of these diagonalization outcomes d, we will not have new nodes added which request diagonalization, since each such new $\mathcal{S}$-node is visited only once.

Therefore we eventually switch to the left of this diagonalization outcome, and by the same argument as in Lemma 5.4 above, such skips cannot happen infinitely often. So one can only skip over β finitely often, and the lemma follows. $\square$

In addition, we need to show that every node along the true path "passes down" infinitely many A-stages and B-stages (in fact, in alternating order), so every node has the chance to perform the action it wants to eventually.

Lemma 5.7 (Alternating stages on true path). *In the construction, every node on the true path is visited infinitely often at A-stages and at B-stages, respectively.*

Proof. This is because in the construction, we require that when we pass to an outcome, we require a different type of stage (A-stage or B-stage) than the one when we last time went to that outcome (otherwise, we wait and do nothing). Along the true path, as we proved above, every node is actually visited infinitely often, and so by this criterion, every node is visited at alternating A-stages and B-stages. $\square$

Now in the following arguments, we always assume that we have a node ξ on the true path and we have passed the stage when all nodes to the left stop acting. Here, action include being visited or accessible, or c-outcome initiation. Since there is finite injury along the true path, we also assume that ξ is the last node along the true path for its requirement, and we only consider stages when it is visited.

5.3. **Witnesses and functionals.** First, we prove two lemmas about the witnesses and various other points we use in the construction.

Lemma 5.8 (Clearing point and witness of parent node). *Given an $\mathcal{S}$-node with diagonalization witness x, the clearing point y (as in the construction) is always less than or equal to x, and such y is stable if no node to the left acts again.*

Proof. This is by inspection of our construction. $\square$

Lemma 5.9 (Claim and killing point of child node under c-outcome). *Given an $\mathcal{S}_{i,j}$-child node, when its c-outcome is initiated (by β, say), the corresponding claim point z (as in the construction) is always strictly larger than its killing point, and is always less than or equal to the diagonalization witness at β.*

Proof. The second claim is by inspection of the definition of such z. The first claim follows from the fact (proved by induction) that such z is always a diagonalization witness below an $\mathcal{S}_{i,j}$-child node's g_α-outcome (for some α), and so larger than the killing point (whenever it changes, every node below is initialized automatically). $\square$

Next, we show that along the true path, every functional is correct on its domain (modulo finite incorrectness for the Δ's). It follows that the functional computes the set we want if it has total domain.

Lemma 5.10 (Γ-functionals). *Every functional Γ is correct on its domain.*

Proof. This is basically by inspection of the construction that when we add any number x into A, we always make sure to issue requests to add the corresponding $\gamma(x)$-uses into B at Γ. It may be the case that later when we visit Γ, the corresponding W has changed up to the use, and since W is c.e., such a change automatically makes the functional undefined and so there is no problem in not adding $\gamma(x)$ into B in this case. If W has not changed, then, of course, by the construction, we will add $\gamma(x)$ into B so that we can correct the axiom. $\square$

The next lemma is going to be the most crucial and most complicated lemma in the proof. Let us first sketch the argument: To show that $\Delta = W$, it suffices to show that whenever we define some Δ as an initial segment of W, then this initial segment of W is not going to change in the construction later. Now at B-stages, this is obvious since $W = \Phi(A)$ where A does not change. At A-stages, the argument is much trickier, but is very similar to the standard argument used in the style of Lachlan's gap-cogap construction. Basically, we have a computation $\Psi(B; x)$ to protect an initial segment of W in such a way that if it changed (after we changed A) then we would switch to the left of the Δ-outcome. The difficult part is to show that after A changes, the B-use of $\Psi(x)$ is always protected. This is usually true since we have only been to the right of such Δ, but remember that in our construction, actions to the right may injure computations to the left.

Lemma 5.11 (Δ-functionals). *Every function Δ is correct on its domain (modulo a fixed finite amount of injury). More precisely, for every such Δ, there is a stage after which Δ is not going to be injured again.*

Proof. Say, such Δ is defined along a g_α-outcome (with killing point x') of β_i, which is a child node for β (where β has diagonalization witness x).

In addition, we know that, for each parent node β' above β and active at β, every child node of this β' along the true path has true c-outcome. Now we have to wait for a stage s_0 such that every such β' has a child node below β (on the true path) with a true c-outcome initiated (i.e., a c-outcome that will not be initialized later).

We claim that after stage s_0, the Δ-axioms are always correct, i.e., compute $W = \Phi(A)$. If A does not change, then, of course, W cannot change. So we only need to consider the case when A changes, in particular, below β_i's $g_{\alpha'}$-outcomes, since otherwise, such an A-change must be to the right and cannot change the initial segment of W witnessed at β'.

Suppose that at some later stage s_1, some node $\bar{\beta}$ below β_i's g_α-outcome performs diagonalization (most likely via a link under the second case). According to the construction, such a node $\bar{\beta}$ must receive permission from every child node with $g_{\alpha'}$-outcome above it. In particular, β_i needs to give permission that $\gamma(z) > \psi(x)$, where z is the associated claim point at β_i, and the Γ-uses range over all Γ's active above β.

By the definition of stage s_0, such $\gamma(z)$'s are going to be the least possible numbers entering B when we switch to the left of β; and by Lemma 5.9, z is less than or equal to the diagonalization witness added into A. In addition, by inspection of the construction, we know that at stage s_1, the computation $\Psi(B; x)$ converges. (Otherwise, the permission criterion $\gamma(z) > \psi(x)$ is always false.)

So we know that, after we add the diagonalization witness into A at stage s_1, and before we come back to β, the computation $\Psi(B; x)$ at β is always preserved. Now it suffices to show that $W = \Phi(A)$ up to $\gamma(x')$ is preserved (recall that x' is the killing point at Δ and we always define W up to $\gamma(x')$).

Otherwise, when we reach the $\mathcal{R}$-node and go to its i-outcome, we would see that the use $\gamma(W; x')$ has changed, and so according to the construction, we will increase its B-use without changing B here. In particular, we know that when we reach β for the first time after s_1, $\gamma(x') > \psi(x)$, and according to the construction at $\mathcal{S}$-nodes, we would immediately build a link to this β_i and switch to the left of the outcome where Δ is defined, and this, of course, contradicts the assumption. $\qquad\square$

5.4. **Final verification.** We are now ready to prove the satisfaction of all requirements. The following two lemmas complete the verification of the construction of Section 4 and the proof of our main theorem.

Lemma 5.12 ($\mathcal{S}_i$-requirements). *Every $\mathcal{S}_i$-requirement is satisfied.*

Proof. Let β be the last $\mathcal{S}_i$-parent node along the true path. It is easy to check that, once we perform diagonalization, then the $\Psi_i(B; x)$-use is going to be preserved (as we choose the killing point y to be the least one such that some $\gamma(y)$ may enter B later in the construction). So we only need to consider the case when we infinitely often see a believable computation $\Psi_i(B; x)$ but we cannot perform diagonalization.

Our argument now splits into two cases. One is that there is an $\mathcal{S}_i$-child node below β on the true path which has true g_α-outcome (we call this case the Σ_3-outcome for β, i.e., the requirement is satisfied in a Σ_3-fashion). The other is that every $\mathcal{S}_i$-child node below β on the true path has true c-outcome (similarly, we say β has true Π_3-outcome).

In the first case, obviously according to the criterion at β, $\psi_i(x) \geq \gamma(x')$ for the killing point x' at β, and the latter goes to infinity by our construction. So $\Psi_i(B; x)$ diverges and our requirement is satisfied.

In the second case, by our criterion for going to c-outcomes, $\psi_i(x)$ is going to be greater than or equal to $\gamma(z)$ for arbitrary large z, and this also implies that $\Psi_i(B; x)$ diverges.

In addition, in the second case, it is easy to see that, for each claim point z, all $\mathcal{S}_i$-child nodes eventually give up using z and start using the next z' as a killing point (later this will allow us to show that the "impact" of this action on each higher-priority Γ is finite). □

Lemma 5.13 ($\mathcal{R}_i$-requirements). *Every $\mathcal{R}_i$-requirement is satisfied.*

Proof. We let α be the last $\mathcal{R}_i$-node along the true path. Of course, we only need to consider the case that $W = \Phi(A)$ is total, and so we go to the i-outcome of α infinitely often, building Γ. Now if there is an $\mathcal{S}_i$-child node along the true path with true g-outcome associated with α, then by Lemma 5.11, the function Δ built there is going to correctly compute W, and so the $\mathcal{R}_i$-requirement is satisfied.

If there is no such $\mathcal{S}$-child node along the true path, then we need to argue that for each fixed x, $\gamma(x)$ only changes finitely often, and so by Lemma 5.10, Γ is going to be a functional computing A from $B \oplus W$, and our Φ-requirement is also satisfied.

So fix an x. We can assume that $A(x) = 0$ in the end, since otherwise, after x enters A, the Γ-use is going to change for the last time and then settle down forever. By our construction, if W changes, we only increase the B-use without changing the W-use, and so the only case in which we may increase the Γ-use forever is that it happens infinitely often that some $\mathcal{S}$-child node below α has outcomes associated with α and puts $\gamma(y)$ for $y \leq x$ into B during B-stages (where y is the killing point). By induction hypothesis, we can assume that $\Gamma(B \oplus W_i; x')$ has settled down for every $x' < x$. Obviously, only finitely many $\mathcal{S}$-requirements can use x as a killing point. Now by the last paragraph of the proof of the previous lemma and by our assumption, all such child nodes which use x as its killing point will eventually give up using x, and so eventually each $\Gamma(B \oplus W_i; x)$-use settles down. □

REFERENCES

[AL86] Ambos-Spies, Klaus and Lerman, Manuel, Lattice embeddings into the recursively enumerable degrees, J. Symbolic Logic **51** (1986), 257–272.

[Ba11] Barmpalias, George, *Review of Lerman's "A framework for priority arguments" (Lecture Notes in Logic, vol. 34. Cambridge University Press, NY, 2010, xvi + 176 pp.)* Bull. Symbolic Logic, **17** (2011), 464–467.

[FeSo81] P. A. Fejer and R. Soare, *The Plus-Cupping Theorem for the Recursively Enumerable Degrees*, in: "Logic Year 1979–80: University of Connecticut",1981, pp. 49–62.

[HS82] Harrington, Leo and Shelah, Saharon *The undecidability of the recursively enumerable degrees*, Bull. AMS (N.S.) **6** (1982), 79–80.

[La66] Lachlan, Alistair H., *Lower bounds for pairs of recursively enumerable degrees*, Proc. London Math. Soc. (3) **16** (1966), 537–569.

[La75] Lachlan, Alistair H., *A recursively enumerable degree which will not split over all lesser ones*, Ann. Math. Logic **9** (1975), 307–365.

[LLS06] Lempp, Steffen; Lerman, Manuel; and Solomon, D. Reed *Embedding finite lattices into the computably enumerable degrees – a status survey*, in: "Logic Colloquium '02", Lect. Notes Log. 27, Assoc. Symbolic Logic, La Jolla, CA, 2006, pp. 206–29.

[LNS98] Lempp, Steffen; Nies, André and Slaman, Theodore A. *The Π_3-theory of the computably enumerable Turing degrees is undecidable*, Trans. Amer. Math. Soc. **350** (1998), 2719–2736.

[Le00] Lerman, Manuel, *A necessary and sufficient condition for embedding principally decomposable finite lattices into the computably enumerable degrees*, Ann. Pure Appl. Logic **101**-(2000), 275–297.

[Le10] Lerman, Manuel, *A Framework for Priority Arguments*, Lecture Notes in Logic, Cambridge University Press, 2010.

[Sa63] Sacks, Gerald E., *Degrees of unsolvability* Ann. Math. Studies No. 55, Princeton University Press, Princeton, N.J., 1963.

[Sa63b] Sacks, Gerald E., On the degrees less than $\mathbf{0}'$, Ann. of Math. (2) **77** (1963), 211–231.

[Sa64] Sacks, Gerald E., The recursively enumerable degrees are dense, Ann. of Math. (2) **80** 1964, 300–312.

[Sho90] Shoenfield, J.R., Non-bounding constructions, Ann. Pure and Applied Logic **50** 1990, 191–205.

[SS93] Shore, R. and Slaman, Theodore A., Working below a high recursively enumerable degree, J. Symb. Logic **58** 1993, 824–859.

[SS99] Slaman, Theodore A., and Soare, Robert I., *Extension of embeddings in the computably enumerable degrees* Ann. Math. (2) **154** (2001), 1–43.

[So87] Soare, Robert I., *Recursively enumerable sets and degrees*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1987.

[Ya66] Yates, C. E. M., *A minimal pair of recursively enumerable degrees*, J. Symbolic Logic **31** (1966), 159–168.

George Barmpalias: (1) State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China and (2) School of Mathematics, Statistics and Operations Research, Victoria University, Wellington, New Zealand
E-mail address: barmpalias@gmail.com
URL: http://www.barmpalias.net

Mingzhong Cai: Department of Mathematics, Dartmouth College, Hanover, NH 03755, USA
E-mail address: Mingzhong.Cai@dartmouth.edu
URL: http://math.dartmouth.edu/~cai

Steffen Lempp: Department of Mathematics, University of Wisconsin, Madison, WI 53706, USA
E-mail address: lempp@math.wisc.edu
URL: http://www.math.wisc.edu/~lempp

Theodore A. Slaman: Department of Mathematics, University of California, Berkeley, CA 94720, USA
E-mail address: slaman@math.berkeley.edu
URL: http://www.math.berkeley.edu/~slaman