JOINING UP TO THE GENERALIZED HIGH DEGREES

PHILIP ELLISON AND ANDREW E.M. LEWIS

Abstract. We show that every generalized high Turing degree is the join of two minimal degrees, thereby settling a conjecture of Posner's from the 70s.

1. Introduction

For Turing degrees below $\mathbf{0}'$, the jump hierarchy provides a way of formalizing the notion of being close to $\mathbf{0}'$ on the one hand, or close to $\mathbf{0}$ on the other. Let $\boldsymbol{a}^{(n)}$ denote the nth jump of $\boldsymbol{a}$. We say that $\boldsymbol{a}$ is *high*$_n$ if $\boldsymbol{a}^{(n)} = \mathbf{0}^{(n+1)}$, and we say that $\boldsymbol{a}$ is *low*$_n$ if $\boldsymbol{a}^{(n)} = \mathbf{0}^{(n)}$. These definitions can also be appropriately generalized in order to deal with degrees which are not necessarily below $\mathbf{0}'$. We say that $\boldsymbol{a}$ is *generalized high*$_n$ if $\boldsymbol{a}^{(n)} = (\boldsymbol{a} \vee \mathbf{0}')^{(n)}$, and we say that $\boldsymbol{a}$ is *generalized low*$_n$ if $\boldsymbol{a}^{(n)} = (\boldsymbol{a} \vee \mathbf{0}')^{(n-1)}$. Note that these hierarchies coincide below $\mathbf{0}'$. We say that a degree is *generalized high* if it is generalized high$_1$.

This paper is part of an ongoing programme to study the relationship between the jump class of a Turing degree and the order theoretic properties that it satisfies, one aim of this programme being to find natural order theoretic conditions which suffice to define the various levels of the jump hierarchy. Along these lines the following are all basic properties that can be isolated as being of specific interest:

(i) **The cupping property.** We say that $\boldsymbol{a}$ satisfies the *cupping property* if it can be cupped up to every degree above it, i.e. for every $\boldsymbol{c} > \boldsymbol{a}$ there exists $\boldsymbol{b} < \boldsymbol{c}$ such that $\boldsymbol{b} \vee \boldsymbol{a} = \boldsymbol{c}$.

(ii) **The join property.** We say that $\boldsymbol{c}$ satisfies the *join property* if every non-zero degree below it can be cupped up to it, i.e. for every non-zero $\boldsymbol{a} < \boldsymbol{c}$ there exists $\boldsymbol{b} < \boldsymbol{c}$ with $\boldsymbol{a} \vee \boldsymbol{b} = \boldsymbol{c}$.

(iii) **The complementation property.** We say that $\boldsymbol{c}$ satisfies the *complementation property* if every non-zero degree below it can be complemented below $\boldsymbol{c}$, i.e. for every non-zero $\boldsymbol{a} < \boldsymbol{c}$ there exists $\boldsymbol{b} < \boldsymbol{c}$ with $\boldsymbol{a} \vee \boldsymbol{b} = \boldsymbol{c}$ and $\boldsymbol{a} \wedge \boldsymbol{b} = \mathbf{0}$.

(iv) **The minimal complementation property.** We say that $\boldsymbol{c}$ satisfies the *minimal complementation property* if every non-zero degree below it can be complemented by a minimal degree below $\boldsymbol{c}$, i.e. for every non-zero $\boldsymbol{a} < \boldsymbol{c}$ there exists a minimal degree $\boldsymbol{b} < \boldsymbol{c}$ with $\boldsymbol{a} \vee \boldsymbol{b} = \boldsymbol{c}$.

Initial segment results (see [ML]) suffice to show that there are degrees which are low (a degree is *low* if it is low$_1$, so any low degree is low$_n$ for all $n \geq 1$) and which fail to satisfy any of the properties (i) to (iv). Low minimal degrees satisfy each of the properties (ii) to (iv). That there exist low degrees satisfying the cupping

2000 *Mathematics Subject Classification.* Primary 03D28, Secondary 03D10.

The first author was supported by an EPSRC research studentship. The second author was supported by a Royal Society University Research Fellowship.

property follows from the fact that all PA degrees satisfy the cupping property [AK]. Similarly it can be seen that being low_2 and not low_1, does not suffice to ensure satisfaction or to ensure failure of any one of these properties. So far then, these results would seem to imply little relation between the jump class of a degree and which of the properties (i)–(iv) it satisfies.

On the other hand, a number of results of a more positive nature have been achieved. Jockusch and Posner showed that all degrees which are not generalized low_2 satisfy the cupping property [JP], and Downey, Greenberg, Lewis and Montalbán [N4] have shown that all such degrees also satisfy the join property. Greenberg, Montalbán and Shore [GMS] have shown that all generalized high degrees satisfy the complementation property. In [BC], Cooper showed that $\mathbf{0}'$ is the join of two minimal degrees and three decades subsequent to this, Lewis, Seetapun and Slaman [AL] proved that $\mathbf{0}'$ satisfies the minimal complementation property. This was extended in [AL2] to show that all degrees above $\mathbf{0}'$ satisfy the minimal complementation property. The result of this paper, then, can be seen as a step towards ruling out the possibility that $\mathbf{0}'$ can be defined as the least degree such that all degrees above it satisfy the minimal complementation property:

Theorem 1.1. *Every generalized high degree is the join of two minimal degrees.*

This result was claimed by Posner in the 70s, but remained unpublished. Subsequent to this, some serious attempts were made to provide a counterexample and disprove Posner's claim, see for example [MG]. Having seen Posner's notes on the question, communicated privately to Cooper, we have been unable to discern whether he had a valid proof in mind, but the proof presented here is anyway substantially different and significantly shorter.

2. Notation and terminology

The notation and terminology we use will generally be standard. The only prerequisite knowledge is Sacks' oracle construction [GS] of a minimal degree below $\mathbf{0}'$ – for an introduction to minimal degree constructions we refer the reader to [RS]. A *tree* $\mathcal{T}$ is a set of finite binary strings, which is not necessarily downward closed. We say $\tau \in \mathcal{T}$ is a *leaf* if there does not exist $\tau' \supset \tau$ in $\mathcal{T}$. Ψ_i denotes the ith Turing functional, and we say that two strings τ and τ' are Ψ-*splitting* if $\Psi(\tau)$ and $\Psi(\tau')$ are incompatible. We say τ is of *level* n in $\mathcal{T}$ if $\tau \in \mathcal{T}$ and it has precisely n proper initial segments in $\mathcal{T}$, and that $\tau' \in \mathcal{T}$ is a *successor* of τ in $\mathcal{T}$ if $\tau' \supset \tau$ and there exists some n such that τ is of level n in $\mathcal{T}$ and τ' is of level $n+1$ (note that we use the term 'successor', where some authors use the term 'immediate successor'). $\mathcal{T}$ is a *c.e.* Ψ-*splitting tree* if every pair of incompatible strings in $\mathcal{T}$ are Ψ-splitting and $\mathcal{T}$ has a computable enumeration $\{\mathcal{T}_s\}_{s\in\omega}$ such that:

Conv A: $|\mathcal{T}_0| = 1$ and each $\mathcal{T}_{s+1} - \mathcal{T}_s$ is finite and consists only of strings extending leaves of $\mathcal{T}_s$.

Conv B: If any string in $\mathcal{T}$ has successors in $\mathcal{T}$, then it has precisely three. Three strings of this kind, all of which are successors to the same string, will be called a splitting triple.

Corresponding to each Ψ-splitting tree that we consider, we shall assume that there is some fixed computable enumeration satisfying Conv A and Conv B. If $\mathcal{T}$ is any tree which is computably enumerated according to Conv A, and if $\sigma \in \mathcal{T}$, then by the *c.e.* Ψ-*splitting subtree* of $\mathcal{T}$ above σ, we just mean any (canonically chosen)

c.e. Ψ-splitting tree $\mathcal{T}' \subseteq \mathcal{T}$ which has σ as the unique string of level 0 and which is maximal, in the sense that if $\tau \in \mathcal{T}'$ and there exist three extensions of τ in $\mathcal{T}$ every pair of which are a Ψ-splitting, then τ has successors in $\mathcal{T}'$. We assume that any string $\tau \in \mathcal{T}'$ is enumerated into this tree at a stage $s \geq s'$, where s' is the stage at which τ is enumerated into $\mathcal{T}$.

If $\mathcal{T}$ is a c.e. Ψ-splitting tree, then the *thin subtree* of $\mathcal{T}$ above $\tau \in \mathcal{T}$ is the smallest $\mathcal{T}'$ satisfying:

(i) $\tau \in \mathcal{T}'$;
(ii) if $\tau' \in \mathcal{T}'$ and is of even level in $\mathcal{T}'$, then its leftmost successor in $\mathcal{T}$ (if there exists such) is in $\mathcal{T}'$;
(iii) if $\tau' \in \mathcal{T}'$ and is of odd level in $\mathcal{T}'$, then every successor of τ' in $\mathcal{T}$ is in $\mathcal{T}'$.

We assume $\mathcal{T}'$ to be given the computable enumeration in which each string in $\mathcal{T}'$ is enumerated into this tree at the same stage it is enumerated into $\mathcal{T}$.

For any two finite binary strings τ and τ', if there exists some least n such that $\tau(n) \downarrow \neq \tau'(n) \downarrow$, then we say that τ is to the left of τ' (and that τ' is to the right of τ) if $\tau(n) = 0$. If $\mathcal{T}$ is tree and $\tau \in \mathcal{T}$ has three successors in $\mathcal{T}$, then we shall refer to these three successors as the *leftmost*, the *second* and the *rightmost* successor respectively when ordered from leftmost to rightmost. We write λ to denote the string of length 0. By a 3-fold Ψ-splitting, we mean three strings, every pair of which are Ψ-splitting. From this point on, by a splitting tree we shall mean a tree which is a c.e. Ψ-splitting tree for some Ψ, and by a splitting we shall actually mean a 3-fold splitting—but no confusion will result from these abuses of terminology. We assume the full binary tree to be given the enumeration in which all strings of length s are enumerated at stage s.

3. The case for degrees above $\mathbf{0}'$

We consider first the case for the degrees above $\mathbf{0}'$. There are a number of ways we could go about defining a construction sufficient to deal with this case (see for example [DP]). We define the construction here so as to serve as an appropriate introduction to the techniques we shall be using in the proof of Theorem 1.1.

3.1. **The basic idea.** Suppose we are given C of degree above $\mathbf{0}'$, and consider running Sacks' oracle construction of a set A of minimal degree below $\mathbf{0}'$, using an oracle for C. At each stage we are given a sequence of trees $\mathcal{T}_0, \cdots, \mathcal{T}_i$ and a finite binary string a which is the initial segment of A that we have constructed so far. We find the greatest $j \leq i$ such that there exists a splitting triple in $\mathcal{T}_j$ above a, and then redefine a to be the leftmost string in this splitting triple, before defining the sequence of trees to be considered at the next stage of the construction.

Of course, we did not *have* to redefine a to be the leftmost string in the splitting. We could consider trying to code C into A at stage $s+1$ by redefining a to be the second string if $C(s) = 0$ and the rightmost string otherwise. A constructed in this way would fail to compute C *because* it is unable to retrace the construction to see how the sequence of trees is defined at each stage. We shall show how to define two sets A and B in this way, however, so that $A \oplus B$ is able to compute the sequence of trees defined at each stage of the construction, and thereby compute C.

The basic idea is very simple. We shall construct a sequence of trees $\mathcal{T}_0, \mathcal{T}_1, \cdots$ such that A is a path through $\mathcal{T}_j$ when j is even, and B is a path through $\mathcal{T}_j$ when

j is odd. For each j we shall also consider a tree $\mathcal{S}_j$ which will be a thin subtree of $\mathcal{T}_j$. Suppose for a moment that j is even—the following discussion will also hold for odd j, but with B in place of A. Initially we try to construct A lying on $\mathcal{S}_j$. $\mathcal{T}_{j+2}$, then, is constructed as a splitting subtree of $\mathcal{S}_j$. When we find at some stage that we must redefine $\mathcal{T}_{j+1}$, however, we code this fact by forcing A to "step off" $\mathcal{S}_j$ (i.e. we decide that A should be a path through $\mathcal{T}_j$ which is not a path through $\mathcal{S}_j$). If this happens at stage s and $A \oplus B$ has already been able to compute the set of trees that we are working with at stage s, then it will be able to see that A has stepped off $\mathcal{S}_j$, and so will be able to discern how we redefined the trees at this stage of the construction.

Along these lines, the following terminology will be useful. Suppose that τ is a string of even level in $\mathcal{S}_j$. Then we call the splitting triple in $\mathcal{T}_j$ whose strings are successors of τ (should such a triple exist), a *coding opportunity*. The second and rightmost strings of the coding opportunity are referred to as the *coding* strings. We shall define a_s and b_s at each stage s, and ultimately we define $A = \bigcup_s a_s$ and $B = \bigcup_s b_s$. At every stage $s > 0$ of the construction we shall find a coding opportunity in some $\mathcal{T}_j$ and define either a_s or b_s (depending on whether j is even or odd) to be one of the strings in this coding opportunity. Note that the set of coding opportunities in $\mathcal{T}_j$ depends also on $\mathcal{S}_j$, which may be redefined at stages of the construction when $\mathcal{T}_j$ is not.

We start, then, with the following sort of procedure in mind. At the beginning of stage $s+1$ we are given $\mathcal{T}_0, \cdots, \mathcal{T}_i$, $\mathcal{S}_0, \cdots, \mathcal{S}_i$, a_s and b_s. Initially we define $a = a_s$ and $b = b_s$. These values a and b may be redefined a finite number of times during stage $s+1$, to be extensions of their previous values. At the end of the stage we will define $a_{s+1} = a$ and $b_{s+1} = b$. At stage $s+1$ we perform a finite iteration which moves down through the sequence of trees, starting with $\mathcal{T}_i$ and continuing until we find some $\mathcal{T}_j$ with a coding opportunity that can be used at this stage. Suppose for now that i is even.

We ask, does there exist a splitting triple above a in $\mathcal{T}_i$? If so, there are now two subcases. If this splitting triple is not a coding opportunity in $\mathcal{T}_i$, then we redefine a to be the leftmost string in the triple and go back to asking whether there exists a splitting triple in $\mathcal{T}_i$ above a. Otherwise, redefine a to be the second string in the coding opportunity if $C(s) = 0$ and the rightmost string otherwise, before redefining $\mathcal{S}_i, \mathcal{T}_{i+1}, \mathcal{S}_{i+1}$ and terminating stage $s+1$ of the construction. The use of this coding opportunity codes the fact that we did not have to redefine any of $\mathcal{T}_0, \cdots, \mathcal{T}_i$ at this stage.

If not, then let $j \leq i$ be the least which is even and such that there doesn't exist a splitting triple above a in $\mathcal{T}_j$. We now try to code the fact that $\mathcal{T}_j$ must be redefined. In order to do so, we begin the iteration again but with $\mathcal{T}_{j-1}$ and b in place of $\mathcal{T}_i$ and a (and with "odd" in place of "even").

3.2. **Further considerations.** When $A \oplus B$ retraces the construction at stage $s+1$, having already computed the set of trees that we are working with at this stage together with a_s and b_s which are the initial segments of A and B which have been decided by the end of stage s, it will continue to enumerate the trees until it sees either A or B step off one of the $\mathcal{S}_j$. We must ensure, however, that the first coding opportunity which appears in this way, with either A or B extending one of its coding strings as appropriate, is actually the coding opportunity that is used

at stage $s+1$ and not one that is used at some subsequent stage. If we proceed simply as described in 3.1 it remains possible that at stage $s+1$ we use a coding opportunity in $\mathcal{T}_i$, which is enumerated into this tree at stage t (say), and that at the next stage we use a coding opportunity in $\mathcal{T}_{i-1}$ which is enumerated into this tree at a stage $t' < t$. In this case $A \oplus B$ will retrace the construction incorrectly. For this reason we define a value t_s at each stage s, which is the stage at which the coding opportunity we use at stage s is enumerated into the respective tree. At stage $s+1$ we are restricted to using coding opportunities which are enumerated into the relevant tree at a stage $> t_s$. The following terminology is useful in this context. If $\mathcal{T}$ is a tree with a given computable enumeration, then for any $\tau \in \mathcal{T}$, $t(\mathcal{T}, \tau)$ is the stage at which τ is enumerated into $\mathcal{T}$. We also use the variable ρ to range over the set of splitting triples, and let $t(\mathcal{T}, \rho)$ be the stage at which ρ is enumerated into $\mathcal{T}$.

Perhaps some words of caution are useful here. In the above and in what follows, we often use computer science type conventions of choosing variables, and then allowing these variables to be redefined multiple times without any corresponding indication in the notation. The motivation here is that, in doing so, the argument should become much easier to follow – the alternative, indexing variables in terms of the stages and ultimately the sub-stages of the construction, would result in heavy notation. The danger is that ambiguity might result from referring to variables that take multiple values at various stages of the construction. During the construction there will be no ambiguity, because when we refer to the value of any given variable, we simply mean the value as defined *at that point in the construction*. During the verification, ambiguity will not result so long as we are very clear about exactly which point of the construction (and not just the stage, but which point of the stage) is being referred to.

It is also worth pointing out that the stages of the construction and the stages in the enumeration of the trees that are considered at any point of the construction, are treated entirely separately. Thus, at stage 5 we might enumerate $\mathcal{T}_2$ until we see that, at stage 117 in the enumeration of this tree, a splitting triple ρ appears which we can use as a coding opportunity. If we use this coding opportunity at stage 5, then $t_5 = 117$, and, at the end of this stage, if we remark that $t(\mathcal{T}_2, \rho) = 117$, then the value $\mathcal{T}_2$ being referred to here is the value at the end of stage 5.

There is another (fairly trivial) way in which our construction will deviate from the Sacks construction. In that construction, when we find at stage $s+1$ that j is the least such that there do not exist any proper extensions of a_s in $\mathcal{T}_j$, we then redefine this tree to be the *full subtree* of $\mathcal{T}_{j-1}$ above a_{s+1} (i.e. the set of all strings in $\mathcal{T}_{j-1}$ which extend a_{s+1}). The redefined tree $\mathcal{T}_j$, however, does not subsequently play any vital role in the construction. If $\mathcal{T}_j$ was defined to be a Ψ_k-splitting tree at the end of stage s, then at stage $s+1$ we could alternatively just redefine $\mathcal{T}_j$ to be the Ψ_{k+1}-splitting subtree of $\mathcal{T}_{j-1}$ above a_{s+1}. It will be technically much more convenient to follow this approach here, and so we shall do so (although now that we are constructing *two* sets and since we are using thin subtrees, $\mathcal{T}_j$ will be defined to be a splitting subtree of $\mathcal{S}_{j-2}$ rather than $\mathcal{T}_{j-1}$).

Before describing the construction precisely, let us describe in outline how $A \oplus B$ will be able to retrace it. So suppose that $A \oplus B$ has already been able to decide a_s, b_s, t_s and the sequence of trees $\mathcal{T}_0, \cdots, \mathcal{T}_i$ which are defined at the end of stage s, together with each $\mathcal{S}_j$ for $j \leq i$. In order to compute these values for stage $s+1$, $A \oplus B$ will then enumerate these trees until it sees an initial segment of A which is compatible with a coding string in some $\mathcal{T}_j$ such that j is even, or an initial segment of B which is compatible with a coding string in some $\mathcal{T}_j$ such that j is odd (and then it will consider the first such coding string which appears). Suppose for now, that the first such coding string τ is a coding string in $\mathcal{T}_j$, and that j is even. Then a_{s+1} is that coding string, and we define the construction so as to ensure that b_{s+1} is the longest initial segment of B which is enumerated into $\mathcal{T}_{j-1}$ at a stage $\leq t(\mathcal{T}_j, \tau)$.

3.3. **The construction.** We proceed in stages as follows.
Stage 0. It is convenient to assume that Ψ_0 is the identity functional $2^{<\omega} \to 2^{<\omega}$. Define $\mathcal{T}_0$ and $\mathcal{T}_1$ to be the Ψ_0-splitting subtree of the full binary tree above λ, and define $\mathcal{S}_0$ and $\mathcal{S}_1$ to be the thin subtree of $\mathcal{T}_0$ above λ. Define $a_0 = b_0 = \lambda$ and $t_0 = 0$.

Stage $s+1$. Let i be the greatest such that $\mathcal{T}_i$ is defined. We run a finite iteration which will terminate at the first step $n+1$ at which we find an appropriate opportunity to code $C(s)$. At step n in this iteration we define a value j_n such that $j_0 = i$ and each $j_{n+1} \leq j_n$. Initially we have $a = a_s$ and $b = b_s$ and we may redefine these values to be extensions of their previous values at various stages in the iteration.

Step 0. Define $j_0 = i$, $a = a_s$ and $b = b_s$.
Step $n+1$. We ask, does there exist a splitting triple in $\mathcal{T}_{j_n}$, above a if j_n is even and above b if j_n is odd? If not, then let j be the least $\leq j_n$, which is even if j_n is even and odd if j_n is odd, such that there does not exist such a splitting triple in $\mathcal{T}_j$ (the way in which we define $\mathcal{T}_0$ and $\mathcal{T}_1$ means that $j \geq 2$). Put $j_{n+1} = j - 1$ and perform step $n+2$.

Otherwise, consider the first such splitting triple ρ enumerated into $\mathcal{T}_{j_n}$. Now there are two cases to consider.

Case (a): ρ is not a coding opportunity or else $t(\mathcal{T}_{j_n}, \rho) \leq \max\{t_s, x\}$, where $x = t(\mathcal{T}_{j_n - 1}, a)$ if j_n is odd and $x = t(\mathcal{T}_{j_n - 1}, b)$ otherwise. Then redefine a to be the leftmost string in the splitting triple if j_n is even, and redefine b to be the leftmost string in the splitting triple if j_n is odd. Put $j_{n+1} = j_n$ and perform step $n+2$.

Case (b): otherwise. Define $t_{s+1} = t(\mathcal{T}_{j_n}, \rho)$. First we code $C(s)$.

If j_n is even then define a_{s+1} to be the second string in the splitting triple if $C(s) = 0$ and the rightmost string otherwise. In this case, define b_{s+1} to be the longest of all those leftmost extensions of b in $\mathcal{S}_{j_n - 1}$ which are enumerated into this tree at a stage $\leq t_{s+1}$. If j_n is odd then define b_{s+1} to be the second string in the splitting triple if $C(s) = 0$ and the rightmost string otherwise. In this case, define a_{s+1} to be the longest of all those leftmost extensions of a in $\mathcal{S}_{j_n - 1}$ which are enumerated into this tree at a stage $\leq t_{s+1}$.

Next we must redefine the trees. Let $j = j_n$.

- If $j = i$, with i defined as at the beginning of stage $s+1$, then suppose that $\mathcal{T}_{i-1}$ is presently defined to be a Ψ_k-splitting tree. Define $\mathcal{T}_{i+1}$ to be the Ψ_{k+1}-splitting subtree of $\mathcal{S}_{i-1}$ above b_{s+1} if i is even, and define $\mathcal{T}_{i+1}$ to be the Ψ_{k+1}-splitting subtree of $\mathcal{S}_{i-1}$ above a_{s+1} if i is odd.

- If $j < i$ then suppose that $\mathcal{T}_{j+1}$ was defined to be a Ψ_k-splitting tree at the end of stage s. Redefine $\mathcal{T}_{j+1}$ to be the Ψ_{k+1}-splitting subtree of $\mathcal{S}_{j-1}$, above b_{s+1} if j is even or above a_{s+1} if j is odd.
- Redefine $\mathcal{S}_j$ to be the thin subtree of $\mathcal{T}_j$, above b_{s+1} if j is odd or above a_{s+1} if j is even. Define $\mathcal{S}_{j+1}$ to be the thin subtree of $\mathcal{T}_{j+1}$, above b_{s+1} if j is even or above a_{s+1} if j is odd. Make $\mathcal{T}_{i'}$ and $\mathcal{S}_{i'}$ undefined for all $i' > j+1$.
- Terminate the iteration and stage $s+1$ of the construction.

It is not difficult to show that $A = \bigcup_s a_s$ and $B = \bigcup_s b_s$ are total and are of minimal degree – since this is not our final version of the construction, we omit the details here. Now suppose that the oracle $A \oplus B$ has already been able to compute a_s, b_s, t_s and the sequence of trees $\mathcal{T}_0, \cdots, \mathcal{T}_i$ which are defined at the end of stage s, together with each $\mathcal{S}_j$ for $j \leq i$. In order to conclude that $A \oplus B$ can compute these values for stage $s+1$, it suffices to show that the coding opportunity we use at this stage is the first enumerated into any $\mathcal{T}_j \in \{\mathcal{T}_0, \cdots, \mathcal{T}_i\}$ such that either A extends one of the coding strings and j is even or B extends one of the coding strings and j is odd. In order to see this, suppose that the iteration terminates at step $n+1$ at stage $s+1$ and let $j = j_n$, as defined at that stage. We consider each of the trees $\mathcal{T}_k$ such that $k \neq j$.

First consider $k > j$. Suppose that k is even, a similar argument will apply when k is odd. Let m be the greatest such that $j_m \geq k$, and let $\sigma = a$ as defined at the beginning of step $m+1$ (so that σ is a string in $\mathcal{S}_k$). If j_m is even then our action at step $m+1$ determines that there is no splitting triple in $\mathcal{T}_k$ above σ. So suppose otherwise. Then by induction on the steps after m, at the beginning of step $n+1$, for all strings $\tau \in \mathcal{T}_k$ extending σ, a is either an initial segment of τ or lies to the left of τ. If j is even, then at step $n+1$ we define a_{s+1} to be incompatible with any coding strings in $\mathcal{T}_k$. If j is odd, then at step $n+1$ we define a_{s+1} to lie to the left of any coding strings in $\mathcal{T}_k$ above σ which are enumerated into this tree at a stage $\leq t_{s+1}$.

Next consider $k < j$. If there does not exist any subsequent stage at which we elect to use a coding opportunity in some $\mathcal{T}_{k'}$ for $k' \leq k$ then A lies on $\mathcal{S}_k$ if k is even and B lies on $\mathcal{S}_k$ if k is odd. So suppose otherwise and consider the first such stage s'. If $k' = k$ then the coding opportunity ρ in $\mathcal{T}_k$ which we use at stage s', has $t(\mathcal{T}_k, \rho) > t_{s+1}$. If $k' < k$ then precisely the argument we used above for the case $k > j$, suffices to show that A does not extend any coding strings in $\mathcal{T}_k$ which are enumerated into this tree at a stage $\leq t_{s'}$ if k is even, and B does not extend any coding strings in $\mathcal{T}_k$ which are enumerated into this tree at a stage $\leq t_{s'}$ if k is odd. Since $t_{s'} > t_{s+1}$, the claim follows.

4. Proving the full theorem

If we are given C which is of generalized high degree then we can no longer assume that C is able to compute whether or not a splitting exists above any given string in any given tree. Since we are constructing A and B using an oracle for C, however, it follows by the recursion theorem that we may suppose that we have a function f which is computable in C and such that, given $\mathcal{S}_j$ as defined at any particular point of the construction, $\lim_t f(A, \mathcal{S}_j, k, t)$ is equal to 1 if there exists a Ψ_k-splitting in $\mathcal{S}_j$ above every initial segment of A, and is equal to 0 otherwise (and similarly with B in place of A). Now we can use this function f in order to

decide how long we search for splittings in any given tree before we begin searching for splittings in previous trees. Our use of the recursion theorem here, is the same as that used by Jockusch [CJ] in showing that all generalized high degrees bound a minimal. An oracle for $(C \oplus \emptyset')'$ can compute, given any k, i, and an enumeration of any c.e. tree $\mathcal{T}$, whether or not there exists a Ψ_k-splitting in $\mathcal{T}$ above every initial segment of $\Psi_i(C)$ (irrespective of whether $\Psi_i(C)$ is partial). Since C is generalized high it can therefore uniformly approximate the answers to questions of this kind. By the recursion theorem, we may suppose given i such that $\Psi_i(C) = A \oplus B$, and therefore also the function f as required.

Suppose that j is even and $\mathcal{T}_{j+1}$ is presently defined to be a Ψ_k-splitting tree. At stage $s+1$ we begin searching for splittings in $\mathcal{T}_{j-1}$ when we find some $t > t_s$ such that $f(B, \mathcal{S}_{j-1}, k, t) = 0$, i.e. *when f indicates that $\mathcal{T}_{j+1}$ may be partial.* Any point at which we are allowed to search for splittings in $\mathcal{T}_j$ we are also allowed to search for splittings in any $\mathcal{T}_{j'}$ for $j' > j$. Let $\mathcal{T}_0, \cdots, \mathcal{T}_i$ and $\mathcal{S}_0, \cdots, \mathcal{S}_i$ be the set of trees defined at the beginning of the iteration at stage $s+1$. At any point during the iteration we say that $\mathcal{T}_j$ is *eligible* at $(s+1, t)$ if either $j = i$ or else for some $j' \leq j+2$ there exists t' with $t_s < t' \leq t$ for which:

- $f(A, \mathcal{S}_{j'-2}, k, t') = 0$ if j' is even and $\mathcal{T}_{j'}$ is presently defined to be a Ψ_k-splitting tree;
- $f(B, \mathcal{S}_{j'-2}, k, t') = 0$ if j' is odd and $\mathcal{T}_{j'}$ is presently defined to be a Ψ_k-splitting tree.

The eligible trees, then, are those in which f dictates that we are allowed to search for splittings. Roughly speaking, what we have said above, is that during the iteration at stage $s+1$ (during which time we do not redefine any of the trees), a tree $\mathcal{T}_{j-1}$ becomes eligible as soon as f indicates that one of the trees $\mathcal{T}_{j'}$, such that $j' \leq j+1$, may be partial.

A slight complication is that those t at which f does not take its limit value, may cause us to give up too soon on the idea of finding Ψ_k-splittings in $\mathcal{T}_{j+1}$, and so end up using a coding opportunity in $\mathcal{T}_j$ and redefining $\mathcal{T}_{j+1}$ at a finite number of stages $s+1$ when we should not have done so. At any subsequent stage s', we therefore say that $(j+1, k)$ *requires attention* if $\mathcal{S}_{j-1}$ has not been redefined at any stage $\geq s+1$ and we find a Ψ_k-splitting in $\mathcal{S}_{j-1}$, above b_{s+1} if j is even and above a_{s+1} if j is odd, within s' steps of some fixed exhaustive search procedure. These pairs of the form $(j+1, k)$ are considered to be ordered lexicographically.

With these changes in place we are ready to define a modified version of the construction, sufficient to prove Theorem 1.1.

4.1. **The construction.** We proceed in stages as follows.
Stage 0. Define $\mathcal{T}_0$ and $\mathcal{T}_1$ to be the Ψ_0-splitting subtree of the full binary tree above λ (assuming the same convention as regards Ψ_0 as previously), and define $\mathcal{S}_0$ and $\mathcal{S}_1$ to be the thin subtree of $\mathcal{T}_0$ above λ. Define $a_0 = b_0 = \lambda$ and $t_0 = 0$.

Stage $s+1$. First we must check to see whether there is some k such that we previously thought we had forced $\Psi_k(A)$ or $\Psi_k(B)$ to be computable, but such that since then, further Ψ_k-splittings have been found which mean we must examine the case for Ψ_k again. If there exists some least pair $(j+1, k)$ which requires attention

(those pairs which satisfy this condition at stage $s+1$ are specified by instructions given at earlier stages, see case (c) later), then:

- Redefine $\mathcal{T}_{j+1}$ to be the Ψ_k-splitting subtree of $\mathcal{S}_{j-1}$, above b_s if j is even and above a_s if j is odd. Define $\mathcal{S}_{j+1}$ to be the thin subtree of $\mathcal{T}_{j+1}$, above b_s if j is even and above a_s if j is odd.
- Make $\mathcal{T}_i$ and $\mathcal{S}_i$ undefined for all $i > j+1$.

We do not need to code this redefinition of the trees, since no oracle was used at this point in the construction in order to decide whether and how these redefinitions should be made. Previous instructions given, as regards when pairs $(j',k') \geq (j+1,k)$ require attention, are now cancelled.

Let i be the greatest such that $\mathcal{T}_i$ is defined. As before, we run a finite iteration which will terminate at the first step $n+1$ at which we find an appropriate opportunity to code $C(s)$.

The iteration.
Step 0. Define $j_0 = i$, $a = a_s$ and $b = b_s$.
Step $n+1$. Find the least t for which there exists $j \leq j_n$, which is even if j_n is even and odd if j_n is odd, such that $\mathcal{T}_j$ is eligible at $(s+1,t)$ and there exists a splitting triple ρ in this tree with $t(\mathcal{T}_j,\rho) \leq t$, whose strings extend a if j is even or b if j is odd. Given this t, let j be the greatest satisfying the conditions above and then consider the first splitting triple ρ enumerated into $\mathcal{T}_j$ satisfying these conditions. Now there are three cases to consider.

Case (a): $j < j_n$. Then define $j_{n+1} = j+1$ and perform step $n+2$. (The corresponding instruction in section 3 has $j-1$ in place of $j+1$. This is because there we look for the *least* j for which we don't see a splitting, while here we take the *greatest* such that we do.)

Case (b): $j = j_n$, but ρ is not a coding opportunity or else $t(\mathcal{T}_{j_n},\rho) \leq \max\{t_s, x\}$, where $x = t(\mathcal{T}_{j_n-1}, a)$ if j_n is odd and $x = t(\mathcal{T}_{j_n-1}, b)$ otherwise. Then redefine a to be the leftmost string in the splitting triple if j_n is even, and redefine b to be the leftmost string in the splitting triple if j_n is odd. Put $j_{n+1} = j_n$ and perform step $n+2$.

Case (c): otherwise. Then we terminate the iteration and make the following definitions.
Definitions for the end of stage $s+1$**.** Define $t_{s+1} = t(\mathcal{T}_{j_n},\rho)$.

First we code $C(s)$.

If j is even then define a_{s+1} to be the second string in ρ if $C(s) = 0$ and the rightmost string otherwise. In this case, define b_{s+1} to be the longest of all those leftmost extensions of b in $\mathcal{S}_{j-1}$ which are enumerated into this tree at a stage $\leq t_{s+1}$. If j is odd then define b_{s+1} to be the second string in ρ if $C(s) = 0$ and the rightmost string otherwise. In this case, define a_{s+1} to be the longest of all those leftmost extensions of a in $\mathcal{S}_{j-1}$ which are enumerated into this tree at a stage $\leq t_{s+1}$.

Next we must redefine the trees.

- If $j = i$ then suppose that $\mathcal{T}_{i-1}$ is presently defined to be a Ψ_k-splitting tree. Define $\mathcal{T}_{i+1}$ to be the Ψ_{k+1}-splitting subtree of $\mathcal{S}_{i-1}$ above b_{s+1} if i is even, and define $\mathcal{T}_{i+1}$ to be the Ψ_{k+1}-splitting subtree of $\mathcal{S}_{i-1}$ above a_{s+1} if i is odd.

- If $j < i$ then suppose that $\mathcal{T}_{j+1}$ is presently defined to be a Ψ_k-splitting tree. Redefine $\mathcal{T}_{j+1}$ to be the Ψ_{k+1}-splitting subtree of $\mathcal{S}_{j-1}$, above b_{s+1} if j is even or above a_{s+1} if j is odd.

  At any subsequent stage s' we say that $(j+1,k)$ *requires attention* if $\mathcal{S}_{j-1}$ has not been redefined subsequent to stage $s+1$, and we find a Ψ_k-splitting in $\mathcal{S}_{j-1}$ within s' steps of some fixed exhaustive search procedure, above b_{s+1} if j is even and above a_{s+1} if j is odd.
- Redefine $\mathcal{S}_j$ to be the thin subtree of $\mathcal{T}_j$, above b_{s+1} if j is odd or above a_{s+1} if j is even. Define $\mathcal{S}_{j+1}$ to be the thin subtree of $\mathcal{T}_{j+1}$, above b_{s+1} if j is even or above a_{s+1} if j is odd. Make $\mathcal{T}_{i'}$ and $\mathcal{S}_{i'}$ undefined for all $i' > j+1$.
- Terminate stage $s+1$ of the construction.

4.2. **The verification.** First, note that $\mathcal{S}_i$ is never redefined at any stage s such that $\mathcal{T}_{i+1}$ is already defined at the beginning of stage s and is not redefined or made undefined at any point during this stage. Therefore, in order to show that, for each i, there is a stage after which $\mathcal{S}_i$ is always defined and never redefined, it suffices to show this for the trees $\mathcal{T}_i$. We wish to show that the following hold for all i:

(a) There is a stage at which $\mathcal{T}_i$ is defined and never subsequently redefined or made undefined.
(b) Suppose $i \geq 2$, that $\mathcal{T}_{i-2}$ is ultimately defined to be a Ψ_{k_0}-splitting tree and that $\mathcal{T}_i$ is ultimately defined to be a Ψ_{k_1}-splitting tree. Then, letting $\mathcal{S}_{i-2}$ take its final value:
   (i) if i is even then $\lim_t f(A, \mathcal{S}_{i-2}, k_1, t) = 1$;
   (ii) if i is even then $\lim_t f(A, \mathcal{S}_{i-2}, k_2, t) = 0$ for all k_2 with $k_0 < k_2 < k_1$;
   (iii) if i is odd then $\lim_t f(B, \mathcal{S}_{i-2}, k_1, t) = 1$;
   (iv) if i is odd then $\lim_t f(B, \mathcal{S}_{i-2}, k_2, t) = 0$ for all k_2 with $k_0 < k_2 < k_1$.

The result is clear for $i = 0$ and $i = 1$, so suppose $i \geq 1$ and the claim holds for all $j \leq i$. We consider the case that i is odd, a similar argument will apply when i is even. Let s_0 be large enough such that $\mathcal{T}_0, \cdots, \mathcal{T}_i$ are never redefined at any stage $\geq s_0$. Then we also have that $\mathcal{S}_0, \cdots, \mathcal{S}_{i-1}$ have already taken their final values at stage s_0 and will not subsequently be redefined. Let $s_1 > s_0$ be such that for each $j \leq i$ with $j \geq 2$ and for all $t \geq t_{s_1}$, $f(A, \mathcal{S}_{j-2}, k, t) = 1$ if $\mathcal{T}_j$ is ultimately defined to be a Ψ_k-splitting tree and j is even, and $f(B, \mathcal{S}_{j-2}, k, t) = 1$ if $\mathcal{T}_j$ is ultimately defined to be a Ψ_k-splitting tree and j is odd. Suppose that $\mathcal{T}_{i-1}$ is ultimately defined to be a Ψ_{k_0}-splitting tree. Let $k_1 > k_0$ be the least such that $\lim_t f(A, \mathcal{S}_{i-1}, k_1, t) = 1$ (the existence of infinitely many k such that Ψ_k is the identity function $2^\omega \to 2^\omega$ ensures that there exists such). Let $s_2 > s_1$ be large enough that $\mathcal{T}_{i+1}$ is never defined to be a Ψ_{k_2}-splitting tree for $k_0 < k_2 < k_1$ at any stage $\geq s_2$. The fact that there exists such a stage s_2 holds, since if $\lim_t f(A, \mathcal{S}_{i-1}, k_2, t) = 0$ then there exists $s_3 > s_1$ such that there are no Ψ_{k_2}-splittings in $\mathcal{S}_{i-1}$ above a_{s_3} and such that $(i+1, k_2)$ never requires attention at a stage $\geq s_3$. Let $s_4 > s_2$ be such that $f(A, \mathcal{S}_{i-1}, k_1, t) = 1$ for all $t \geq t_{s_4}$. If $\mathcal{T}_{i+1}$ is defined to be a Ψ_{k_1}-splitting tree at stage $s_4 + 1$ then it will never subsequently be redefined. If not, then there will be a least stage after $s_4 + 1$ at which $(i+1, k_1)$ requires attention and $\mathcal{T}_{i+1}$ will never be redefined subsequent to that stage. This completes the induction.

Now consider the point of the construction at which $\mathcal{T}_i$ is (re)defined for the last time. This could either be at the end of some stage s or at the beginning of some stage $s+1$. Then a_{s+1} properly extends a_s if i is even and b_{s+1} properly extends b_s if i is odd. Therefore $A = \bigcup_s a_s$ and $B = \bigcup_s b_s$ are total and are of minimal degree so long as they are non-computable. Non-computability can be seen to follow either through an application of Posner's lemma for minimal degree constructions (see for example [RS]), or else from the fact, observed below, that $A \oplus B$ computes C.

The last part of the verification goes through in almost exactly the same way as in section 3. So suppose that the oracle $A \oplus B$ has already been able to compute a_s, b_s, t_s and the sequences of trees which are defined at the end of stage s. Then it is also able to compute the sequence of trees $\mathcal{T}_0, \cdots, \mathcal{T}_i$ which is defined before we carry out the iteration at stage $s+1$, together with each $\mathcal{S}_j$ for $j \leq i$. In order to conclude that $A \oplus B$ can compute these values for stage $s+1$, it suffices to show that the coding opportunity we use at this stage is the first enumerated into any $\mathcal{T}_j \in \{\mathcal{T}_0, \cdots, \mathcal{T}_i\}$ such that either A extends one of the coding strings and j is even or B extends one of the coding strings and j is odd. In order to see this, suppose that the iteration terminates at step $n+1$ at stage $s+1$ and let $j = j_n$, as defined at that stage. We consider each of the trees $\mathcal{T}_k$ such that $k \neq j$.

First consider $k > j$. Suppose that k is even, a similar argument will apply when k is odd. Let m be the greatest such that $j_m \geq k$, and let $\sigma = a$ as defined at the beginning of step $m+1$ (so that σ is a string in $\mathcal{S}_k$). By induction on the steps after m, at the beginning of step $n+1$, for all strings $\tau \in \mathcal{T}_k$ extending σ, a is either an initial segment of τ or lies to the left of τ. If j is even, then at step $n+1$ we define a_{s+1} to be incompatible with any coding strings in $\mathcal{T}_k$. If j is odd, then at step $n+1$ we define a_{s+1} to lie to the left of any coding strings in $\mathcal{T}_k$ above σ which are enumerated into this tree at a stage $\leq t_{s+1}$.

Finally, we must consider $k < j$. This case, however, goes through exactly as written in section 3.

References

[BC] S.B. Cooper, Degrees of unsolvability complementary between recusively enumerable degrees, *Annals of Mathematical Logic*, 4 (1), 31-74, (1972).

[BC2] S.B. Cooper, Minimal degrees and the jump operator, *Journal of Symbolic Logic* 38 (2), 249-271, (1973).

[N4] R. Downey, N. Greenberg, A.E.M. Lewis, A. Montalbán, Extensions of uppersemilattice embeddings below computably enumerable degrees, in preparation.

[MG] M. Giorgi, PhD thesis, Leeds University.

[GMS] N. Greenberg, A. Montalbán, R. Shore, Generalized high degrees have the complementation property, *Journal of Symbolic Logic*, 69 (2004), 1200-1220.

[CJ] C. Jockusch, Simple proofs of some theorems on high degrees of unsolvability, *Canadian Journal of Mathematics*, 29, (1977), 1072-1080.

[JP] C. Jockusch, D. Posner, Double jumps of minimal degrees, *Journal of Symbolic Logic*, 43 (4), 715-724, 1978.

[AK] A. Kucera, Measure, Π^0_1-classes and complete extensions of PA, *Recursion Theory Week* (Oberwolfach, 1984), volume 1141 of Lecture Notes in Math. 245-259, Springer, Berlin.

[ML] M. Lerman, Degrees of Unsolvability, Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1983.

[AL] A.E.M. Lewis, Minimal complements for degrees below $\mathbf{0'}$, *Journal of Symbolic Logic*, 69 (4), 937-966.

[AL2] A.E.M. Lewis, The minimal complementation property above $\mathbf{0'}$, *Mathematical Logic Quarterly*, 51 (2), 470-492.

[DP] D. Posner, A survey of the non-r.e. degrees $\leq \mathbf{0}'$, *London Mathematical Society Lecture Note Series* 45, Recursion Theory: its Generalisations and Applications, Proceedings of the Logic Colloquium '79, Leeds, August 1979, edited by F.R. Drake and S.S.Wainer.

[GS] G. Sacks, A minimal degree less than $\mathbf{0}'$, *Bulletin of the American Mathematical Society*, vol 67, 416-419, 1961.

[RS] R.I. Soare, *Recursively enumerable sets and degrees*, Springer, New York, (1987).

Pure Mathematics, University of Leeds, England, LS29JT
E-mail address: andy@aemlewis.co.uk, phil.j.ellison@googlemail.com