ANOTHER NOTE ON THE JOIN PROPERTY

MINGZHONG CAI

(Communicated by Mirna Dzamonja)

Abstract. We first prove two theorems on the low\(_2\) degrees and the join property in the local structure \(\mathcal{D}(\leq 0')\): An r.e. degree is low\(_2\) if and only if it is bounded by an r.e. degree without the join property (in \(\mathcal{D}(\leq 0')\)), and an FPF \(\Delta^0_2\) degree is low\(_2\) if and only if it fails to have the join property. We also study the join property in the global structure and show that for every array recursive degree, there is a degree above it which fails to satisfy the join property.

1. Introduction

Definability is one of the central topics of mathematical logic. For example, in classical recursion theory, we study definability of various notions of Turing degrees. That is, given a degree-invariant notion in second-order arithmetic, we try to find an equivalent definition using the Turing reducibility (order) in the Turing degree structure, i.e., in the structure \((\mathcal{D}, \leq)\). One can also consider restricted domains, and typical examples include \((\mathcal{D}(\leq 0'), \leq)\), the collection of \(\Delta^0_2\) degrees, and \((\mathcal{R}, \leq)\), the collection of r.e. degrees.

In [L], Lewis proved that for every low degree (i.e., \(a' = 0'\)), there is always another low degree above it which does not satisfy the join property (a degree \(d\) satisfies the join property if for every \(b\) strictly between \(d\) and \(0\), there is a \(c < d\) such that \(d = b \lor c\)). In the other direction, the best result so far is the following:

**Theorem 1.1 (DGLM).** Every \(\mathcal{GL}_2\) degree \(d\) (i.e., \(d'' > (d \lor 0')'\)) satisfies the join property.

As a corollary, if we only consider the local structure of the degrees below \(0'\), there is only a small gap left: the degrees that are low\(_2\) (\(a'' = 0''\)) but not low. So naturally Lewis conjectured the following:

**Conjecture 1.2.** In \((\mathcal{D}(\leq 0'), \leq)\), can the low\(_2\) degrees be defined as the ones that are bounded by a degree which does not satisfy the join property?

In the paper, Lewis actually proved slightly stronger theorems:

**Theorem 1.3.** If \(a\) is a low r.e. degree, then there is an r.e. degree \(d \geq a\) which does not satisfy the join property.

**Theorem 1.4.** Every FPF low degree fails to satisfy the join property.
In [L], Lewis used Theorem 1.4 and the fact that every low degree is bounded by a low FPF degree (low basis theorem) to prove the main result. Our first observation is that, this attempt fails for low_2. In fact, if an r.e. degree is low_2 but not low_1, then it is not bounded by any incomplete FPF degree below 0' ([BGKLNT Corollary 1.3]).

In the other direction, we are able to generalize both Theorem 1.3 and 1.4 to the low_2 case:

**Theorem 1.5.** If a is a low_2 r.e. degree, then there is an r.e. degree d ≥ a which does not satisfy the join property (in (D(≤ 0'), ≤)).

**Theorem 1.6.** Every FPF low_2 degree fails to satisfy the join property.

In the proofs of the above two theorems, we modified Lewis’ construction changing it from a finite injury argument to an infinite injury argument. Now together with Theorem 1.1, we have the following characterizations:

**Corollary 1.7.** In the r.e. degrees, the low_2 ones are exactly the ones that are bounded by an r.e. degree without the join property (in (D(≤ 0'), ≤)).

**Corollary 1.8.** An FPF Δ_0^2 degree has the join property if and only if it is non-low_2.

Lewis also conjectured that in the global structure (D, ≤), the Turing jump can be defined similarly by the join property:

**Conjecture 1.9.** Can 0' be defined as the least degree such that every degree above it satisfies the join? Or given an incomplete degree, can we always find a degree above it which fails to satisfy the join?

For example Lewis’ theorem shows that given a low degree, there is always a degree above it which fails to satisfy the join. Our third theorem of the paper is to show that this is also true for array recursive degrees. Recall that a degree is array recursive if every function recursive in the degree is dominated by m_K, the modulus of the halting problem.

**Theorem 1.10.** If a is an array recursive degree, then there is a degree d ≥ a which does not satisfy the join property.

Note that both low_2 (in the local structure) and the jump (in the global structure) have been shown to be definable (see [SS], [S] and [S2]), but the known definitions depend on coding arithmetic into the degree structure and so it is still interesting to find their natural definitions.

Here we also include a few remarks on the notation used in the paper. Given a Δ_0^2 set A, A_s is a fixed Δ_0^2 approximation to A at stage s up to the last bit where it is changed. So each A_s is a finite binary string. In the case when A is r.e., then A_s is a fixed Σ_1^0 approximation. When this A_s is an initial segment of A, we say it is a correct version or correct initial segment of A; otherwise we say it is incorrect.

2. The first theorem: r.e. case

Here we prove the first theorem:

**Theorem 2.1.** Given a low_2 r.e. degree a, there exist r.e. degrees b and c such that a ∨ b ∨ c does not have the join property via b, i.e., no d < a ∨ b ∨ c joins b to a ∨ b ∨ c.
2.1. Requirements and priority tree. Given a low$_2$ r.e. set $A$, we build two r.e. sets $B$ and $C$ such that $A \oplus B \oplus C$ does not satisfy the join and $B$ is the witness that $A \oplus B \oplus C$ fails to satisfy the join property. We try to satisfy the following list of requirements:

- $\mathcal{R}_e : [\Phi_e(A \oplus B \oplus C) = D_e \land \Psi_e(D_e \oplus B) = C] \Rightarrow [\exists \Gamma(\Gamma(D_e) = B)]$,

where $\Phi_e$ and $\Psi_e$ are both the $e$-th Turing functional (and we use Posner’s trick for notational convenience).

Note that this is slightly stronger than what we actually need to satisfy (i.e., as opposed to $\Psi_e(D_e \oplus B) = A \oplus C$). Of course we also want $B$ to be nonrecursive:

- $\mathcal{P}_e : \exists x(B(x) \neq \Theta_e(x))$,

where $\Theta_e$ is the $e$-th partial recursive function.

We use a priority tree (growing downwards) in the construction. Each $\mathcal{R}_e$ node has infinitely many outcomes, labeled by $(i, j)$ where $i$ ranges in $\omega$ and $j$ ranges in $\{0, 1\}$, ordered lexicographically from left to right (i.e., $(0, 0), (0, 1), (1, 0), (1, 1), \ldots$).

We call these outcomes type 0 or type 1 according to their $j$ values. Each $\mathcal{P}_e$ node has two outcomes $d$ (for diagonalization) and $w$ (for waiting). The nodes are arranged naturally by priority and the highest priority node is at the top.

As usual, at each stage we build an accessible path through the priority tree, and argue in the end that the requirements are satisfied along the left-most accessible path (true path).

2.2. Construction.

2.2.1. $\mathcal{P}_e$ nodes. The construction at a $\mathcal{P}_e$ node is almost the same as that in L: In order to put some diagonalization witness $x$ into $B$, we need to put some (possibly much smaller) trigger $m$ into $C$ to force $D$ to change in order to satisfy higher priority $\mathcal{R}$ requirements.

When we first come to this $\mathcal{P}_e$ node $\beta$ (or after it is initialized), we pick a fresh number $m$, and a number $x_m$ which we require to be greater than all $\psi_e(m)$ uses for all $\mathcal{R}_e$ nodes above $\beta$ which have a type 0 outcome along $\beta$. (Intuitively, type 0 means that we believe the functional $\Psi$ is total and type 1 means that we believe it is nontotal.) So whenever such use $\psi_e(m)$ changes, we repick a fresh, bigger $x_m$, but $m$ remains the same until $\beta$ is initialized.

At $\beta$, we check whether $\Theta_e(x_m)$ converges to 0. If not, then we continue to wait (outcome $w$). If so, then we plan to put $x_m$ into $B$ and put $m$ into $C$ at the same time (and go to the $d$ outcome afterwards).

However, in the construction, such diagonalization plan $(x_m, m)$ might be restrained by a higher priority $\mathcal{R}$ node (see Section 2.2.2). Once we see this, we pick a new $m'$ and start the construction with a new $x_{m'}$, but do not cancel the old pair $(m, x_m)$. So we actually have a list of diagonalization pairs $(x_{m_1}, m_1)$ at $\beta$, and whenever we see a restraint drops (regardless of the current accessible path) we immediately follow the diagonalization plan, if possible, using the least pair $(m_i, x_{m_i})$ which is not restrained at that stage. That is to say, we allow nodes to the left of the accessible path to perform diagonalization.

For convenience, when we perform diagonalization at some $\mathcal{P}_e$ node, we then initialize all lower priority nodes. After such diagonalization, we always follow the $d$-outcome and never perform any action at $\beta$. 

In the verification, we will show that either we have one \( m_i \) with final value \( x_{m_i} \), which witnesses that \( \Theta_e(x_{m_i}) \) does not converge to 0, or eventually we will use some pair \( (m_i, x_{m_i}) \) for diagonalization.

2.2.2. \( R_e \) nodes. The construction at an \( R_e \) node \( \alpha \) is as follows. First, without loss of generality we can assume that \( D_e \) always agrees with \( \Phi_e(A \oplus B \oplus C) \) (at any stage \( s \)), and we only use \( D_e \) up to the length of convergence of the functional \( \Phi_e \) in the computation of \( \Psi_e \). The main question of concern here is whether \( \Psi_e(D_e \oplus B) \) is actually total. If it is total, then lower priority \( \mathcal{P} \) requirements have to make sure that \( x > \psi_e(m) \) as above; if it is not total, then actually the requirement is already satisfied.

By the Recursion Theorem, we can try to ask the following to \( A'' \): Is it the case that for every \( x \) there is a stage \( s \) at which \( \alpha \) is accessible and for every \( y \leq x \), \( \Psi_e(D_e \oplus B; y) \) converges such that the corresponding \( D_e \) use is computed by \( \Phi_e \) using a correct initial segment of \( A \)?

It is easy to see that, if the answer is no, then obviously \( \Psi_e \) is going to be partial (at any \( x \) which witnesses the negation). If the answer is yes, then we basically want to preserve the \( B \) and \( C \) uses of \( \Phi_e \) and subsequently preserve the \( D_e \) use of \( \Psi_e \) (and only allow finitely many injuries to each \( \Psi_e(D_e \oplus B; z) \)) to force \( \Psi_e \) to be total in the end. Note that only \( \mathcal{P} \) requirements are going to change \( B \) and \( C \), and so we give all \( \mathcal{P} \) nodes below \( \alpha \) a natural order (priority number). Along a 0 type outcome, whenever \( \Psi_e(D_e \oplus B; z) \) converges with current versions of \( D_e \) and \( B \), we impose a new restraint on \( B \) and \( C \)'s \( \Phi_e \)-uses for the corresponding \( D_e \) used in the computation and do not remove the restraint until we see that \( A \) up to the \( \Phi_e \)-use is changed, i.e., we see that the computation we tried to preserve uses an incorrect version of \( A \).

For notational simplicity we may omit the oracles and write \( \Psi_e(z) \). We only allow \( \mathcal{P} \) nodes below \( \alpha \) whose priority number is \( z \) or smaller (smaller priority number means higher priority) to injure such restraints on \( B \) and \( C \) (for protecting the use of the corresponding \( \Psi_e(z) \)). This allows at most \( z \) injuries for each \( \Psi_e(z) \) when the true outcome is of type 0; yet each \( \mathcal{P} \) requirement below only needs to respect some finite amount of total (permanent) restraint and so eventually has a chance to act.

Now we use the condition that \( A \) is low, i.e., \( A'' = 0'' \). The \( A'' \) questions we asked can be uniformly computed with oracle \( 0'' \) and so can be uniformly approximated in a standard \( \Delta^0_3 \) way. That is, for an \( R_e \) node \( \alpha \), we can effectively find two \( \Sigma^0_3 \) sentences \( \varphi = \exists x \forall y \exists z R(x, y, z) \) and \( \psi = \exists x' \forall y' \exists z' R'(x', y', z') \) such that the answer to our question at \( \alpha \) is yes if and only if \( \varphi \) is true, and if and only if \( \psi \) is false.

Note that the outcomes are labeled by \((i,j)\), where \( j \) corresponds to the \( \Sigma_3 \) approximations of either \( \text{yes}(0) \) or \( \text{no}(1) \) answer and \( i \) corresponds to the \( \Sigma_3 \) witness, i.e., \( x \) or \( x' \) as above. It is easy to see that if \( j_0 \) corresponds to the correct answer and \( i_0 \) is the first \( \Sigma_3 \) witness, then the outcome \((i_0, j_0)\) is the leftmost one visited infinitely often.

Now extending a type 1 outcome, we don’t have to do anything since the requirement is already satisfied. Extending a type 0 outcome, if \( \Psi_e \) finally agrees with \( C \), then we need to define a functional \( \Gamma \) computing \( B \) from \( D_e \). To achieve this, we only need to define \( \Gamma(D_e; x_m) \) for each \( x_m \) that might potentially enter \( B \). We do this naturally by using the \( \Psi_e \)-use of \( D_e \) at \( m \) to compute \( B(x_m) \) when
\[\Psi_e(D_e \oplus B; m) = C(m).\] Otherwise if \(\Psi_e(D_e \oplus B; m) \downarrow \neq C(m)\) then we stop the definition of \(\Gamma\) at \(x_m\) (and so stop the definition of \(\Gamma\), but the requirement is also satisfied). In the verification, we will show that if \(x_m\) enters \(B\), then by \(m\) entering \(C\) we can force a change in \(D_e\) up to the \(\Psi_e\) use, and so the functional \(\Gamma\) is well-defined (see verification for details).

2.3. Verification. The fact that there is a true path (leftmost path accessible infinitely often) follows from the \(\Delta^0_3\) approximation of each \(R_e\) node. So in the verification below, we always assume that we are at a node \(\alpha\) on the true path, and we have passed the stage when all nodes to the left of \(\alpha\) have stopped acting (e.g. performing diagonalization). We use \(s(\alpha)\) to denote that stage.

2.3.1. \(R_e\) requirements. Assume that \(\alpha\) is the \(R_e\) node on the true path. If its true outcome is of type 1, i.e., the answer to the key question is no, then obviously the functional \(\Psi_e\) is partial and the requirement is automatically satisfied. In this case, we also know a number \(x\) which is the first place where \(\Psi_e(x)\) diverges, and any \(B\) and \(C\) restraints for preserving \(\Psi_e(x)\) will eventually drop (see verification below for \(\mathcal{P}_e\)). (This number \(x\) may be less than the least witness of the failure of the \(\Pi^A_2\)-question we asked.)

Now suppose that the true outcome is of type 0, i.e., for each \(x\) there is a stage \(s\) when we see computations \(\Psi_e(y)\) for each \(y \leq x\) with correct \(A\) uses. Since \(A\) is r.e., such \(A\) use will remain correct forever, and so in order to show that \(\Psi_e\) is total in the end, we only need to verify that each \(\Psi_e(z)\) is only injured finitely often. The only ones that causes injury are the \(\mathcal{P}\) nodes below \(\alpha\) whose priority number is less than \(z\), and obviously each one is going to follow the diagonalization plan at most once. Therefore \(\Psi_e(z)\) can only be injured at most \(z\) times, and there is going to be a stage after which \(\Psi_e(z)\)’s corresponding \(A\)-use is correct and \(B\) and \(C\) uses remain unchanged forever. So \(\Psi_e\) is going to be total.

In this case we need to verify that \(\Gamma\) is well-defined (this part is essentially the same as in [L]). We can assume that \(\Psi_e(D_e \oplus B)\) agrees with \(C\) in the end, since otherwise the requirement is satisfied. It suffices to verify that for each \(x_m\) entering \(B\), the corresponding \(D_e\)-use of \(\Gamma(x_m)\), i.e., the \(D_e\)-use of \(\Psi_e(m)\), changes to versions which are different to what we have ever seen before. We prove this by induction on the size of \(x_m\). Let \(s_0\) be the stage when we add \(x_m\) into \(B\) and \(m\) into \(C\). Before stage \(s_0\), all \(D_e\)-uses used in the definition of \(\Gamma(x_m)\) compute \(\Psi_e(D_e \oplus B; m) = 0\). If \(B\) up to the \(\Psi_e(m)\) use remains the same as of stage \(s_0\), then of course \(D_e\) has to change to a different version, as \(C(m) = 1\) after diagonalization. If later at stage \(s_1\), \(B\) up to the \(\Psi_e(m)\) use changes, then it means that there is a smaller \(x_{m'}\) entering \(B\), which implies that a higher priority (than \((m, x_m)\)) \(\mathcal{P}\) node acts at that stage and so \(m' < m\). In addition, such \(x_{m'}\) remains unchanged after stage \(s_0\) (otherwise it will become larger than the use we want to preserve), meaning that the use of \(\Psi_e(D_e \oplus B; m')\) has not yet changed since \(s_0\). Now by the induction hypothesis, \(D_e\) up to such \(\Psi_e(m')\) use has to be different from any versions we have seen before \(s_1\) (and so before \(s_0\)).

Since \(\Psi_e\) is total, the \(\Gamma\) uses eventually stop growing, and it is easy to see that \(\Gamma\) is always correct, i.e., it computes the correct value of \(B\) at each stage. Therefore the requirement is satisfied, as \(\Gamma(D_e) = B\) in the end.
2.3.2. $P_e$ requirements. At a $P_e$ node $\beta$, we want to show that either there is a trigger $m_i$ such that for the final value $x_{m_i}$, $\Theta_e(x_{m_i})$ does not converge to 0, or eventually we diagonalize at $\beta$ (and the requirement is permanently satisfied).

First we need to argue that for each $m_i$, eventually $x_{m_i}$ stops changing. This is by our construction that each $x_{m_i}$ only changes when $\Psi_e(m_i)$ uses change for requirements with type 0 outcome above $\beta$. By induction hypothesis (see Section 2.3.1) we know that these $R_e$’s all have total $\Psi_e$. Therefore eventually these uses stop changing and consequently $x_{m_i}$ stabilizes.

According to each $R_e$ node above, this $\beta$ has been assigned priority numbers and let $z$ be the largest such. For every $R_e$ above $\beta$ with type 0 outcome, let $\psi_e(z)$ be the final use of $\Psi_e(z)$; for those with type 1 outcome, let $\psi_e(z)’$ be the final use of $\Psi_e(z)$ if it is convergent, or the final use of the last convergent $\Psi_e(z’)$, if $\Psi_e(z)$ is divergent.

Suppose our claim were false, then we could find a pair $(m,x_{m_i})$ at $\beta$ which requests to perform diagonalization and both are greater than the maximum value of all such $\psi_e(z)$’s (so the permanent restraints will not restrict such pair from performing diagonalization). Now it is easy to see that only these (temporary) restraints imposed by $R_e$’s with type 1 outcome along $\beta$ can prevent $(m,x_{m_i})$ from performing diagonalization. All these temporary restraints correspond to incorrect initial segments of $A$ and so each will eventually be removed. The argument is slightly trickier since we need to show that it is not the case that $(m,x_{m_i})$ is always restrained but by possibly different requirements at each stage.

At $s_0$ be the first stage when all of these permanent restraints $\psi_e(z)$ stop changing and $\Theta_e(x_{m_i})$ has already converged and it equals 0. At $s_0$, the reason we do not perform diagonalization is because of these $R_e$’s with type 1 outcome imposing some temporary restraint. There can be only finitely many such temporary restraints at this stage, and let $A_{s_0} \upharpoonright y$ be the shortest $A$-use corresponding to these restraints. Such initial segment is incorrect, and so at some stage $s_1$, it changes and so all these restraints we saw at $s_0$ drop. Now at $s_1$, the only chance that we still have some temporary restraint for $(m,x_{m_i})$ is that there are some other restraints imposed by higher priority requirements between stages $s_0$ and $s_1$, i.e., some temporary restraint was issued with a shorter $A_{s} \upharpoonright y’$ which remains correct at $s_1$. However, since it is still incorrect, there must be a stage $s_2$ when such initial segment changes and again the restraint is dropped. However, this procedure cannot happen infinitely often, as each time the length of such incorrect initial segment of $A$ drops. So eventually at some stage $s_i$, we will see a change in $A_{s_i} \upharpoonright \hat{y}$, which drops all the temporary restraints which prevent $(m,x_{m_i})$ from performing diagonalization. So at stage $s_i$, we would have performed diagonalization, which contradicts our assumption. This finishes the proof of the theorem.

3. The second theorem: FPF case

Now we turn to our second theorem:

**Theorem 3.1.** Every FPF low$_2$ degree $a \ (\leq 0’)$ fails to have the join property, i.e., there exists a nonrecursive $b < a$ such that no $d < a$ joins $b$ to $a$. In fact, one can choose $b$ to be r.e.

Basically we combine the idea above with the FPF permitting argument introduced by Kucera [K] in the same way as in [L]. By the Recursion Theorem, we can
assume that in the construction we have a recursive list of diagonalization points \( \{n_i : i \in \omega \} \), and we are free to define \( \varphi_{n_i}(n_i) \) along our recursive construction. We fix a DNR function \( f \leq_T a \) and a recursive approximation \( f_s \) for \( f \). So in particular, for each \( i, f(n_i) \neq \varphi_{n_i}(n_i) \). In the construction, this allows us to show that \( f \) computes \( B \) and \( C \) (as in the first construction, we also build an r.e. set \( C \)).

### 3.1. List of requirements and priority tree

The list of requirements is very similar to the one in the first theorem. We have the requirements which ensure that \( B \) cannot be joined with some \( D \) to \( A \) unless \( D \) already computes \( B \) (and \( \Phi_e, \Psi_e \) and \( \Theta_e \) are the same as in the first construction):

- \( R_e : [\Phi_e(A) = D_e \land \Psi_e(D_e \oplus B) = C] \Rightarrow [\exists \Gamma'(\Gamma(D_e) = B)]. \)

Again we want \( B \) to be nonrecursive:

- \( P_e : \exists x(B(x) \neq \Theta_e(x)). \)

The priority tree is actually exactly the same as what we used in the proof of the first theorem, so we omit its description here.

### 3.2. Construction

The construction here resembles the proof of the first theorem. We build an accessible path along the priority tree at each stage, and at the same time allow \( P \) nodes to the left of the accessible path to act if they meet some certain criteria to perform diagonalization.

#### 3.2.1. \( R_e \) nodes

At an \( R_e \) node, we only need to worry about the totality of \( \Psi_e(D_e \oplus B) \). Knowing the index of \( B \) (as a function recursive in \( A \)) in advance by the Recursion Theorem, we can translate it to an \( A'' \) question (in the end we will show that \( B \) is recursive in \( A \)). Since \( A \) is low\(_2\), we can find a \( \Delta_3^0 \) approximation to the totality of \( \Psi_e(D_e \oplus B) \). So again type 0 outcomes mean that \( \Psi_e \) is total and type 1 outcomes mean that it is partial.

Compared to the first theorem, the difference here is that we do not have to actively preserve the witnessed computations of \( \Psi_e \) anymore. \( A \) is a fixed given set and if the \( \Delta_3^0 \) approximation gives a yes answer, then we know that \( \Psi_e \) is total and do not have to put restraints on \( B \) or \( C \). The \( \Gamma \) functional along each type 0 outcome is handled in the same way as in Section 2.3.1

#### 3.2.2. \( P_e \) node

Now suppose that we are at a \( P_e \) node \( \beta \). When we first visit \( \beta \) (or after it has been initialized), we pick a new \( m \) and choose the corresponding \( x_m \) in the similar way as in Section 2.3.2 i.e., \( x_m \) is greater than all \( \Psi(m) \) uses of all \( R \) nodes above \( \beta \) and which have type 0 outcome along \( \beta \). Once any of such uses increase, we change \( x_m \). In addition, we also pick a new point \( n_i \) from the diagonalization points for the permission argument.

Once we pick such \( m \) and \( n_i \) (at stage \( s_0 \)), we continue the following construction regardless of the accessible path as long as \( \beta \) is not initialized (i.e., we allow left nodes to act). First of all we of course wait for \( \Theta_e(x_m) \) to converge to 0. Meanwhile we check the approximation \( f_1(n_i) \) for \( t \geq s_0 \) (remember \( f \) is \( \Delta_3^0 \) and \( f_1 \) is a recursive approximation). If at some stage \( t \) we see that the value \( f_1(n_i) \neq f_{s_0}(n_i) \) before we see \( \Theta_e(x_m) \downarrow 0 \), then we cancel the pair \( (m, x_m) \) and pick a new pair \( (m', x_{m'}) \), but keep the same \( n_i \) (and now the \( f \) approximation starts from this \( t \)). Once for some current pair \( (m^*, x_{m^*}) \), we see that \( \Theta_e(x_{m^*}) \downarrow 0 \) (say at stage \( s_1 \)) before the \( f(n_i) \) approximation changes, then we add \( m^* \) into \( C \) and add \( x_{m^*} \) into \( B \).

Once we diagonalize (which can happen when the node is not on the accessible path), then we define \( \varphi_{n_i}(n_i) = f_s(n_i) \) where \( s \) is the current stage number. We
also initialize all lower priority nodes. After that, we always go down to the \( d \) outcome since the requirement is now permanently satisfied. If we haven’t performed diagonalization, we always go down to the \( w \) outcome.

3.3. Verification. Following the same argument as in the proof of the first theorem, we know that there is a true path. Along the true path each \( R_e \) requirement is satisfied by essentially the same argument (note that there are only finitely many nodes to the left which are visited, and so there are only finite many actions to the left of the true path which might injure this \( R_e \) node).

There are two facts that remain to be verified. First, each \( P_e \) requirement is satisfied, and second, \( B \oplus C \) is recursive in \( f \). We use the assumption that \( f \) is DNR. In particular for any \( n_i, f(n_i) \neq \varphi_{n_i}(n_i) \).

Suppose we are at a \( P_e \) node \( \beta \) on the true path, and we have passed the stage when all nodes to the left of \( \beta \) stopped acting. Now we have a fixed \( n_i \) at \( \beta \) which will not be canceled by higher priority requirements. For this \( n_i \), the approximation \( f_s(n_i) \) only changes finitely often, therefore there is a last \( (m_j, x_{m_j}) \) which will not be canceled. Either \( \Phi_e(x_{m_j}) \) does not converge to 0, or eventually it does and we put \( m_j \) into \( C \) and \( x_{m_j} \) into \( B \) for diagonalization.

To see that \( B \oplus C \) is recursive in \( A \) (or \( f \)), we observe the following: First of all, only picked pairs \( (m, x_m) \) (associated with \( n_i \)) will potentially enter \( B \) and \( C \), so for all other \( B(x) \) and \( C(x) \), we know their values are 0. Now for a pair \( (m, x_m) \) picked at stage \( s_0 \), we run the approximation of \( f_s(n_i) \) starting from \( s_0 \) until the value agrees with \( f(n_i) \), the final value, say at \( s_1 \). If at \( s_1 \), the pair \( (m, x_m) \) has been canceled (e.g., the node has been initialized or \( f_s \) changes before \( \Theta_e(x_m) \) converges to 0), then of course they will never have chance to enter \( B \) and \( C \). If they have been added in by stage \( s_1 \), then of course \( B(x_m) = C(m) = 1 \). Now if they have not been canceled but also have not been added in, then we claim that they will never be added in later. Note that in this case, \( f_{s_0}(n_i) = f_{s_1}(n_i) = f(n_i) \). Now later if they were added in, then \( \Theta_e(x_m) \) must have converged to 0 (at some stage \( s_2 \)) before \( f_s \) changes, and so at the time, \( f_{s_2}(n_i) \) would agree with \( f_{s_1}(n_i) = f(n_i) \) and we would have defined \( \varphi_{n_i}(n_i) = f(n_i) \). This contradicts our assumption of \( f \) being DNR. Therefore we have a recursive algorithm computing \( B \oplus C \) from \( f \). To be precise, in order to apply the Recursion Theorem, we actually have a uniform algorithm computing \( B \oplus C \) from \( f \), i.e., there is a recursive function, given an index of the construction, gives an index witnessing \( B \oplus C \leq_T f \). This finishes the proof.

4. The third theorem: Array recursive case

In this section, we prove the third theorem. The readers can easily find the idea from the previous two constructions here: Any change in \( B \) has to be paired with a change in \( C \) to force \( D \) to change.

**Theorem 4.1.** For every array recursive degree \( a \), there is a degree \( c \geq a \) which fails to have the join property, i.e., there is a nonrecursive degree \( b < c \) such that there is no \( d < c \) which joins \( b \) to \( c \).

As usual, we assume that we have a set \( A \) whose degree is array recursive, and we plan to construct sets \( B \) and \( C \) such that the corresponding degrees \( b = \deg(B) \) and \( c = \deg(A \oplus B \oplus C) \) have the required property.
In the construction, we use tree systems for approximating such \( B \) and \( C \). Intuitively, such a tree system is a “tree of trees”. In this construction, a tree is a subset of \( 2^{\leq \omega} \) which is closed downwards. A set \( A \) is a path on \( T \) if every initial segment of \( A \) is in \( T \).

A finite tree of height \( n \) is a subset of \( 2^{\leq n} \) which is closed downwards and contains at least one string of length \( n \). Notice that in this definition, every finite tree has a unique height.

A tree system is a pair \((T, S)\) where \( T \) is a tree and \( S \) is a function with the following properties:

(1) the domain of \( S \) is \( T \);
(2) for every \( \tau \) on \( T \), \( S(\tau) \) is a finite tree of height \(|\tau|\) (i.e., length of \( \tau \));
(3) if \( \tau \subseteq \tau' \) on \( T \), then \( S(\tau) = S(\tau') \cap 2^{\leq |\tau'|} \).

We call \( S \) the system part of the tree system. Intuitively, the system \( S \) uniformly associates a tree to each path \( P \) on \( T \), i.e., \( S(P) = \cup_{n \in \omega} S(P \upharpoonright n) \).

Such a tree system is recursive in \( A \) if both \( T \) and \( S \) are recursive in \( A \). In our construction, we will use \( A \)-recursive tree systems to approximate the sets \( B \) and \( C \). In other words, \( B \) is approximated by \( \Pi_1^{0,A} \) (i.e., \( \Pi_1^0 \) relative to \( A \)) classes, and \( C \) is approximated by \( \Pi_1^{0,A\oplus B} \) classes. For convenience of notation, we restrict the use of Greek letters: \( \tau \) (with possible superscripts and subscripts) only stands for strings on the tree \( T \) and \( \rho \) (as well as other letters such as \( \eta \)) only stands for strings on the system, i.e., on some \( S(\tau) \).

4.1. Initial tree system. In this construction, the initial tree system is a little bit more complicated than usual. We in fact need a recursion-theoretic injury construction just for the initial tree system.

The motivation is the same as the ideas used in the previous two constructions. Recall that we want to make sure that if \( \Psi(D \oplus B) = C \) for some \( D = \Phi(A \oplus B \oplus C) \), then \( D \) can compute \( B \). So whenever we change \( B \), say at \( x \), we want to change \( C \) at some \( m \) such that \( x > \psi(m) \) so that the change of \( C \) automatically yields a difference in \( D \) as \( \Psi(D \oplus B) = C \) where the \( B \)-use for computing \( C(m) \) is unchanged.

The set \( A \) being array recursive gives us a strong way of bounding such uses \( \psi(m) \) and so we can follow the modulus function of \( K \) in the initial tree system to enforce \( x > \psi(m) \) even without knowing the set \( A \).

We assume that the readers know some basic \( \Pi_1^0 \) class constructions, and so it is clear when we say “we terminate a node \( \tau' \)”, we mean that we stop adding extensions for any \( \tau' \supsetneq \tau \).

Without loss of generality, we can arrange the enumeration of \( K \) and modify its modulus (making it strictly increasing) so that the limit computation \( \lambda(x, s) \) to \( m_K(x) \), the modulus of \( K \), has the following properties:

(1) At any stage \( s \), for at most one \( x \), \( \lambda(x, s) \neq \lambda(x, s - 1) \) (i.e., we only enumerate \( K \) at most one element at a time), and if this happens, \( \lambda(x, s) = s \);
(2) \( \lambda(x, s) \) only changes at most \( x \) times (just for notational convenience we assume that \( 0 \notin K \));
(3) \( \lambda(x, s) \neq \lambda(y, t) \) for any \( x \neq y \) (e.g., the approximation \( \lambda(1, s) \) starts with 1 instead of 0 in the usual setting).

For example, at stage 17 we see 5 entering \( K \), and we have not yet started defining \( \lambda(8, 0) \). Then \( \lambda(5, 17) = 17, \lambda(6, 18) = 18, \lambda(7, 19) = 19 \) and \( \lambda(8, 0) = 20 \).
Using this function $\lambda(x, s)$, we construct the initial tree system $(T, S)$ as follows. Intuitively, (the extendable part of) the tree $T$ is very thin, i.e., the nodes with “true branchings”, branchings that will not be terminated, occur very sparsely on the tree. We put down requirements $P_i$ for each $i \in \omega$ saying that $B \neq \varphi_i$ (for the requirement that $B$ is not recursive, as in the previous two constructions).

In the construction, each $P_i$ wants to put true branching nodes of length (i.e., of height) at least $m_K(i)$, above the $P_{i-1}$’s corresponding nodes. For example, if some $\tau$ is of length $m_K(i)$, then $P_i$ requests to make sure both $\tau \ast 0$ and $\tau \ast 1$ are extendable, so later if $\varphi_i(|\tau|)$ converges, then we can freely terminate one of the branches. We say that these nodes are assigned to $P_i$.

Of course we cannot recursively compute $m_K(x)$, and so we run the approximation $\lambda(x, s)$ and use it for the construction of $T$ as in a finite injury argument. At any stage $s + 1$, the tree $T$ has been constructed up to height $s$. We first check if for some $P_i$ assigned to nodes of length $l$, $\varphi_i(l)$ converges. If so, we say $P_i$ acts to terminate all $\tau$’s such that $\tau(l) = \varphi_i(l)$ (after this termination $P_i$ is satisfied and never active again, in addition we never cancel the assignment of $P_i$).

After we finish checking through these $P_i$ for diagonalization, we check whether the $\lambda$ approximation has changed (and if so we need to move the assignment of the corresponding $P$’s).

If we do not see any change in $\lambda$ so far, then we add two immediate extensions for the current height $s$ nodes on the tree, and these extensions are assigned to the first requirement $P_k$ which currently does not have assigned nodes.

If we see a change in $\lambda$, i.e., for some (and only one) $x$, $s + 1 = \lambda(x, s + 1) \neq \lambda(x, s)$. If $P_x$ has acted before, then we do not do anything and continue to extend each node of length $s$ by one node of length $s + 1$. If $P_x$ has not acted, then we cancel all node assignments to $P_x$ or lower requirements which have not acted. For each node $\tau$ assigned with $P_{x-1}$, we fix two length-$s$ extensions of $\tau \ast 0$ and $\tau \ast 1$ respectively, and extend them to one of their immediate successors of length $s + 1$, and then terminate all other nodes extending $\tau$ which are incomparable with the ones we choose. Then we assign all these length-$s + 1$ nodes to the requirement $P_x$.

Note that in such a construction, we always assign a whole level of nodes to one requirement, and at any stage, one requirement is only assigned to one level of nodes.

It is not difficult to verify that we can run this construction recursively, and in the end the true branching nodes are as high as the modulus of $K$ as arranged by the requirements $P_i$. All $P_i$ requirements are satisfied and so all infinite paths on $T$ are nonrecursive.

Now let us focus on the system part. For each $\tau$ on the tree $T$, we need to specify a finite tree of height $|\tau|$ as $S(\tau)$. First let $h(x) = \sum_{i=0}^{x-1}(i + 1)$ (and $h(0) = 0$). Each requirement $P_i$ (as above) is also assigned the “space” between $h(i)$ and $h(i + 1) - 1$. For example, $P_0$ is assigned with interval $[0, 0]$, $P_1$ has $[1, 2]$ and $P_2$ has $[3, 5]$. In particular, each $P_i$ is assigned with a closed interval of $i + 1$ numbers, which will be used for handling the changes in the approximation of $\lambda(i, s)_{s \in \omega}$.

Roughly speaking, the nodes on the tree where $P_i$ is assigned (and where true branching occurs) correspond to changes in $B$, and these intervals here are for corresponding change in $C$ (which will later be used to force $D$ to change). Since

---

1. Alternatively, if later the approximation $\lambda(i, s)$ changes, we could assign higher nodes to $P_i$ and start diagonalization again, but it is unnecessary.
\( \lambda \) value may change, we reserve some extra positions for each \( P_i \) so that we can always pick a new position when the old \( \lambda \) value changes.

On the system part, the construction is by default adding two immediate extensions of a node unless some requirement \( P_i \) prevents such extension. If \( P_i \) acts in the tree construction (for diagonalization), then it stops its action in the system part (but still all nodes that have been terminated remain terminated).

For example, if \( \tau \) on \( T \) is at first assigned with \( P_1 \), then extending \( \tau \ast 0 \), \( P_1 \) terminates all nodes \( \rho \) (on the system part) with \( \rho(1) = 1 \), and similarly extending \( \tau \ast 1 \), \( P_1 \) terminates all nodes \( \rho \) with \( \rho(1) = 0 \). If \( 1 \notin K \) (i.e., \( m_k(1) \) has reached its final value), then that is all; if \( 1 \in K \) then according to the tree construction above, there are higher nodes assigned with \( P_1 \), and for these higher \( \tau' \) assigned with \( P_1 \), \( P_1 \) in addition requests to terminate all \( \rho \) with \( \rho(2) = 1 \) on \( \tau' \ast 0 \) side and with \( \rho(2) = 0 \) on \( \tau' \ast 1 \) side. Intuitively, each change on the tree (in \( B \)) corresponds to a change on the system (in \( C \)).

Inductively (following the same construction as the tree part), at some height \( s + 1 \), if some \( P_i \) acts in the tree construction, then it stops its action in the system part. Then we check whether some \( P_x \) changes its nodes (some \( \lambda(x, s) \) changes value). If not, then the next \( P_k \) (as in the tree construction) puts requirements saying that for all \( \tau \) newly added in the tree (all of which are assigned with \( P_k \)), for any extension of \( \tau \ast 0 \) (resp. \( \tau \ast 1 \)), all \( \rho \) with \( \rho(h(k)) = 1 \) (resp. \( = 0 \)) are terminated. If some \( P_x \) changes its nodes, then say it is the \( c \)-th time \( P_x \) changes its nodes, we add in requests saying that for all \( \tau \) at height \( s + 1 \) (which are all assigned to \( P_x \) at the end of the tree construction), for extensions of \( \tau \ast 0 \) (resp. \( \tau \ast 1 \)), all \( \rho \) with \( \rho(h(x) + c) = 1 \) (resp. \( = 0 \)) are terminated.

This finishes the construction of the initial tree system (which we later denote as \( (T_0, S_0) \)). Notice that the matching between \( B \) and \( C \) is not exactly what we planned, but the difference is a recursive function \( h(x) \), which can be absorbed easily in the verification.

4.2. Construction. The inductive construction at each step is actually much easier compared to the construction of the initial tree system. It is very similar to the “forcing totality” construction in the hyperimmune-free basis theorem.

By induction, we assume that we have \( (T_e, S_e) \) which is recursive in \( A \). The next requirement in concern is \( R_e \), which says:

\[
\text{If } \Phi_e(A \oplus B \oplus C) = D_e \text{ and } \Psi_e(D_e \oplus B) = C, \text{ then there is } \Gamma \text{ such that } \Gamma(D_e) = B.
\]

In the construction, we will terminate some nodes on the system part, i.e., given \( \rho \) on \( S(\tau) \), we stop any extension of \( \rho \) on any \( \tau' \supset \tau \). Notice that this termination automatically induced termination on the tree part, i.e., if we see that all nodes on \( S(\tau) \) are terminated, then we also terminate \( \tau \) on the tree.

Now we ask, is there an \( x \) such that if we terminate all \( \rho \)'s on \( S(\tau) \) such that \( \Psi_e(\Phi_e(A \oplus \tau \oplus \rho) \oplus \tau; x) \downarrow \), and correspondingly terminate some \( \tau \)'s as described above, we still have an infinite tree part?

If the answer is yes with a witness \( x \), then we take \( (T_{e+1}, S_{e+1}) \) to be the new tree system after applying termination with respect to \( x \). If the answer is no, then we know that we have forced totality of \( \Psi_e \) along every path of the tree system. In particular, it is easy to see (as in some standard \( \Pi^0_1 \) class argument), there is a
function $f \leq_T A$ which dominates every possible use function $\psi_e$ (see verification below for details).

In this case, we let $(T_{e+1}, S_{e+1}) = (T_e, S_e)$, and we will see in the verification why such $f$ can be used to help us computing $B$ from $D_e$ in the end. This finishes the inductive step of the construction.

4.3. Verification. Since the requirement that $B$ is nonrecursive has been satisfied by the initial tree system, here we only need to satisfy each $R_e$. In the construction above, if we can force $\Psi_e(\Phi_e(A \oplus B \oplus C) \oplus B; x)$ to be partial at some $x$, then obviously we have satisfied the requirement. So we only need to discuss the case when $\Phi_e$ and $\Psi_e$ are total, and we let $D_e = \Phi_e(A \oplus B \oplus C)$.

Directly from the $A$-recursive tree system $(T_e, S_e)$, it is easy to see that there is a function $f \leq_T A$ which dominates the final use function $\psi_e(D_e \oplus B)$: Fixing $x$, from the negative answer to the question we asked in the construction at stage $e$, we know that if we terminate all $\rho$’s on $S(\tau)$ such that $\Psi_e(\Phi_e(A \oplus \tau \oplus \rho) \oplus \tau; x) \downarrow$, and terminate $\tau$ if all branches of $S_e(\tau)$ are terminated, then the tree part is finite. So we know that there is a finite level $l_x$ on the tree where all nodes are terminated. Now the final computation $\Psi_e(\Phi_e(A \oplus B \oplus C) \oplus B; x)$ is one of the computations $\Psi_e(\Phi_e(A \oplus \tau \oplus \rho) \oplus \tau; x)$ for some $\tau$ at this level $l_x$ and some $\rho$ on the corresponding system $S_e(\tau)$. It is easy to see that $A$ can recursively find all possible such $\tau$’s and $\rho$’s and so compute a function which dominates the final use function $\psi_e(D_e \oplus B)$.

As $A$ is array recursive, such $f(x)$ is dominated by the $m_K(x)$ and so by the limit of $\lambda(x, s)$.

Actually we need to argue with the function $\psi_e(h(x + 1))$ (recall that $h$ is the recursive function used in the construction of the initial tree system for the “space” between two requirements). Since such $h$ is recursive, we know $\psi_e(h(x + 1))$ is also dominated by limit of $\lambda(x, s)$.

This means that, there is an $x_0$ such that for every $x > x_0$, eventually the $P_x$ requirement (in the initial tree system construction) either has acted (for diagonalization), or is assigned to some node $\tau$ higher than $\psi_e(h(x + 1))$. Let $\beta_0$ be the initial segment of $B$ of length $m_K(x_0)$ and let $s_0$ be the stage when all $P_y$ requirements for $y \leq x_0$ settle down in the construction of the initial tree system.

Now suppose that in the end $D_e = \Phi_e(A \oplus B \oplus C)$ and $\Psi_e(D_e \oplus B) = C$, then we compute $B$ from $D_e$ as follows: To find an initial segment of $B$ of length $l$, we search on the initial tree system $(T_0, S_0)$ (recall that the initial tree system is recursive) for a pair $(\tau, \rho)$ (i.e., $\tau$ is on $T$ and $\rho$ is on $S_0(\tau)$) and a stage $s > s_0$ with the following properties:

1. $\tau$ and $\rho$ are still extendable at stage $s$;
2. $\tau$ extends $\beta_0$ and $|\tau| \geq l$;
3. for every $x$, if $\Psi_e(D_e \oplus \tau; x)$ converges, then it equals $\rho(x)$;
4. for every $P_x$ assigned to some node $\tau^*$ between $\beta_0$ and $\tau$ at stage $s$, either $|\tau^*| > \psi_e(D_e \oplus \tau; h(x + 1))$, or $P_x$ has acted before stage $s$.

The second condition says that we only search for strings above $\beta_0$ (where only large enough $P_x$ matters); the third one says that our $(\tau, \rho)$ need to satisfy $\Psi_e(D_e \oplus B) = C$; the last (and the most important) condition says, for every $x$ which matters ($x > x_0$), either $P_x$ has acted and so we know which way it goes (on the tree $T_0$), or $|\tau^*| > \psi_e(D_e \oplus \tau; h(x + 1))$, i.e., the length of $\tau^*$ (as a possible initial segment of $B$) where $P_x$ is assigned is greater than the use of $\psi_e(h(x + 1))$.
(and so greater than the uses from $\psi_e(h(x))$ to $\psi_e(h(x+1) - 1)$ by monotonicity of the use function).

We claim that for every $l$, we can always find such $(\tau, \rho)$ and $s$ satisfying the conditions, and for any such triple we find, $\tau$ is in fact an initial segment of $B$. These two claims suffice to finish the proof.

It is not difficult to see that for every $l$, there always exist such $(\tau, \rho)$ and $s$ since the true $B$ and $C$ are respectively on the tree and the corresponding system, and a large enough $s$ suffices to find out that all relevant $P_x$ (with respect to the long enough $\tau \subset B$) have acted or been assigned high enough (by the assumption that $A$ is array recursive).

The hard part is to show that for any such $(\tau, \rho)$ and $s$ satisfying the above conditions, $\tau$ is always an initial segment of $B$. Suppose not, then such $\tau$ disagrees with $B$ first at some $t$, i.e., $\tau(t) \neq B(t)$, and say $\tau' = \tau \upharpoonright t = B \upharpoonright t$. Such a splitting at $t$ must be a place where some $P_x$ is assigned at stage $s$. Without loss of generality, we can assume that $\tau \supset \tau' \ast 0$ and $B \supset \tau' \ast 1$. According to the construction of the initial tree system, we have some $y \in [h(x), h(x+1) - 1]$ such that on the $\tau$ side, $\eta(y) = 0$ for all extendable $\eta$ on the system (including $\rho$); and extending the other side ($B$ side), $\eta(y) = 1$ for all extendable $\eta$ on the system. By conditions of $(\tau, \rho)$ and $s$ as above, $|\tau'| > \psi_e(D_e \oplus \tau; h(x+1)) \geq \psi_e(D_e \oplus \tau; y)$ and so $\Psi_e(D_e \oplus \tau; y) \downarrow = 0$, but then along $B$ we also have $\Psi_e(D_e \oplus \tau'; y) \downarrow = 0$, which leads to a contradiction since along $B$ side, every possible extendable $\eta$ (including initial segments of $C$) has $\eta(y) = 1$ (i.e., $C(y) = 1$ but $\Psi_e(y) = 0$).

ACKNOWLEDGMENTS

The author thanks Joseph S. Miller for very helpful discussions.

REFERENCES


Department of Mathematics, University of Wisconsin, Madison, Wisconsin 53706
Current address: Department of Mathematics, Dartmouth College, Hanover, NH 03755
E-mail address: mingzhong.cai@dartmouth.edu