Absolutely undecidable sets

Rupert Hölzl
Universität der Bundeswehr München

Joint work with Laurent Bienvenu and Adam R. Day

1 *Reminder:* Turing degrees

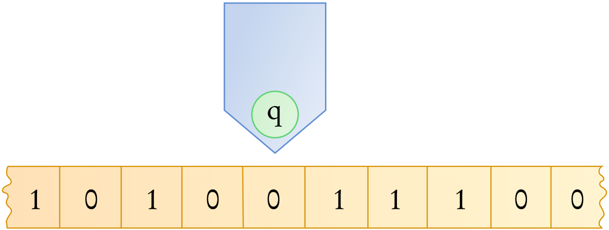
Turing machines

1. Turing machines are a theoretical computation model that can simulate any other classical computation model.

Turing machines

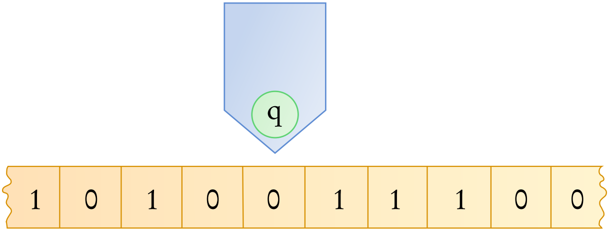
1. Turing machines are a theoretical computation model that can simulate any other classical computation model.
2. Such a machine is in one of *finitely* many internal states q.

Turing machines



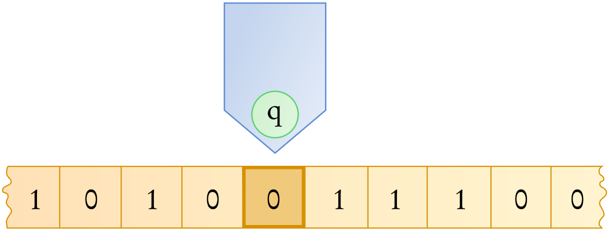
1. Turing machines are a theoretical computation model that can simulate any other classical computation model.
2. Such a machine is in one of *finitely* many internal states q.
3. It has reading and writing heads that move around on one or more *infinite* tapes and read and write symbols.

Turing machines



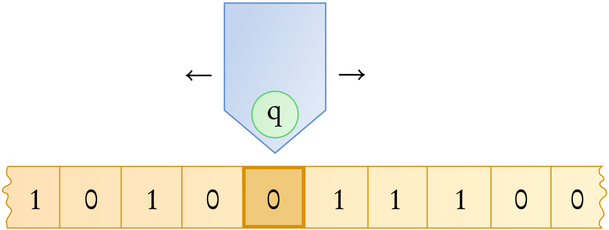
1. Turing machines are a theoretical computation model that can simulate any other classical computation model.
2. Such a machine is in one of *finitely* many internal states q.
3. It has reading and writing heads that move around on one or more *infinite* tapes and read and write symbols.
4. Its internal state and last read symbol determine its next actions:

Turing machines



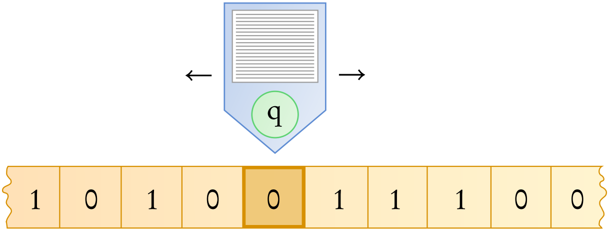
1. Turing machines are a theoretical computation model that can simulate any other classical computation model.
2. Such a machine is in one of *finitely* many internal states q.
3. It has reading and writing heads that move around on one or more *infinite* tapes and read and write symbols.
4. Its internal state and last read symbol determine its next actions:
   - the symbol to write in the current cell and

Turing machines



1. Turing machines are a theoretical computation model that can simulate any other classical computation model.
2. Such a machine is in one of *finitely* many internal states q.
3. It has reading and writing heads that move around on one or more *infinite* tapes and read and write symbols.
4. Its internal state and last read symbol determine its next actions:
   - the symbol to write in the current cell and
   - the next movement.

Turing machines



1. Turing machines are a theoretical computation model that can simulate any other classical computation model.
2. Such a machine is in one of *finitely* many internal states q.
3. It has reading and writing heads that move around on one or more *infinite* tapes and read and write symbols.
4. Its internal state and last read symbol determine its next actions:
    - the symbol to write in the current cell and
    - the next movement.
5. The instructions for this are given as a *finite* list, a *programme*.

Turing machine computations

1. Turing machines can compute sets:
   - Designate one internal state as *accepting*, and one as *rejecting*.
   - **Definition.** A set $A \subseteq \mathbb{N}$ is *computably enumerable* if there is a Turing machine M that terminates in the accepting state iff $n \in A$.
   - **Definition.** A set $A \subseteq \mathbb{N}$ is *computable* if there is a Turing machine M that terminates in the accepting state if $n \in A$ and in the rejecting state otherwise.

2. Turing machines can compute functions:
   - Designate one tape as *input tape* and one as *output tape*.
   - Initially, the input tape contains a binary word σ as input.
   - If the machine terminates after it has produced a binary word τ on the output tape, then we write $M(\sigma) = \tau$.
   - **Definition.** A partial function f is *partial computable* if there is a Turing machine M with $M(\sigma) = f(\sigma)$ for all $\sigma \in \mathrm{dom}(f)$.
   - Via binary encoding we can have computable $f : \mathbb{N} \to \mathbb{N}$.

Turing functionals

1. **Intuition.** A *Turing functional* computably converts one *infinite* binary sequence into another.
2. **Definition.** A *Turing functional* $\Phi\colon 2^\omega \to 2^\omega$ is a (partial) function for which there exists a Turing machine M such that

$$\sigma, \sigma' \in \mathrm{dom}(M) \ \wedge\ \sigma \preceq \sigma' \implies M(\sigma) \preceq M(\sigma')$$

For $A \in 2^\omega$ where $|M(A \upharpoonright n)| \to \infty$, let $\Phi(A) = \lim_{n\to\infty} M(A \upharpoonright n)$. Otherwise $\Phi(A)$ is undefined.

3. We write Φ^A for $\Phi(A)$.
4. **In other words:** A Turing functional is a function transforming *infinite* sequences into *infinite* sequences. It is induced by an underlying Turing machine that operates on *finite* sequences.

Turing degrees

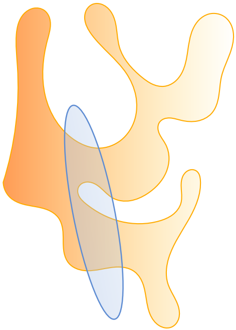
1. **Definition.** A is *Turing reducible* to B, written as $A \leq_T B$, if there is a Turing functional Φ such that $A = \Phi^B$.
2. **Definition.** A is *Turing equivalent* to B, written as $A \equiv_T B$, if both $A \leq_T B$ and $B \leq_T A$.
3. **Definition.** The *Turing degrees* are the equivalence classes induced by $\equiv_T$.
4. **Intuition.** All sets in a Turing degree contain the same information, but represented differently. The representations can be transformed into each other using a Turing functional.
5. **Definition.** A tt-*functional* is a total Turing functional.

2 Motivation

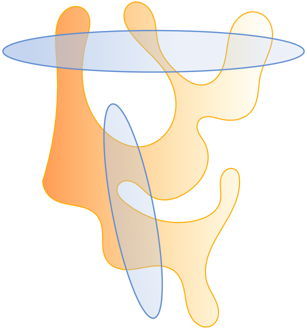
Bi-immunity

Bi-immunity

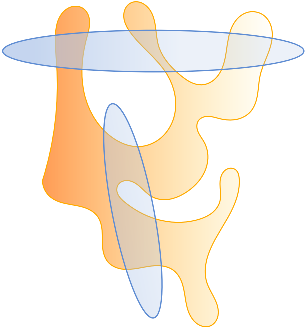
Bi-immunity



Bi-immunity

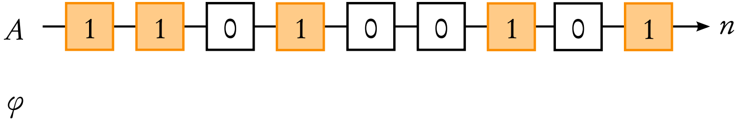


Bi-immunity



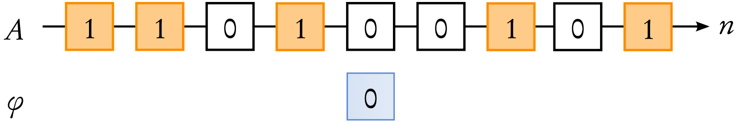
1 **Definition.** A set A is *bi-immune* if neither A nor its complement contain an infinite computably enumerable set.

Bi-immunity



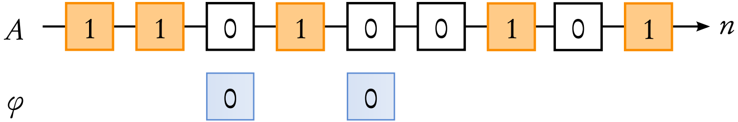
1. Another way of seeing bi-immunity is to say that a partial computable function φ that "predicts" $A(n)$ for infinitely many n must make a mistake somewhere.

Bi-immunity



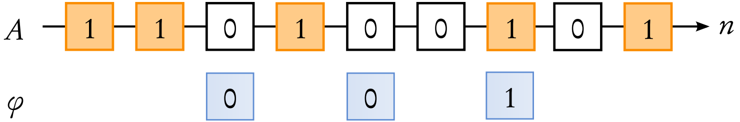
1. Another way of seeing bi-immunity is to say that a partial computable function φ that "predicts" $A(n)$ for infinitely many n must make a mistake somewhere.

Bi-immunity



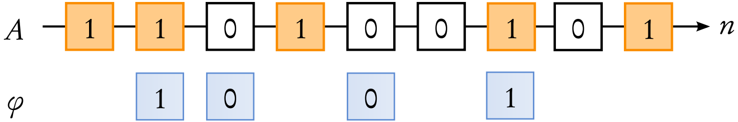
1. Another way of seeing bi-immunity is to say that a partial computable function φ that "predicts" $A(n)$ for infinitely many n must make a mistake somewhere.

Bi-immunity



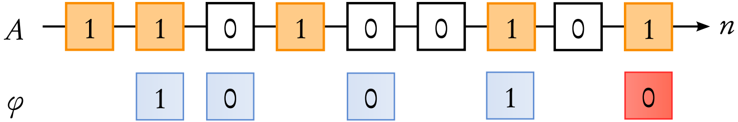
1. Another way of seeing bi-immunity is to say that a partial computable function φ that "predicts" $A(n)$ for infinitely many n must make a mistake somewhere.

Bi-immunity



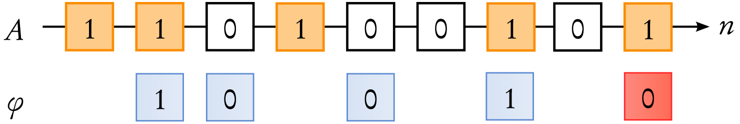
1. Another way of seeing bi-immunity is to say that a partial computable function φ that "predicts" $A(n)$ for infinitely many n must make a mistake somewhere.

Bi-immunity



1. Another way of seeing bi-immunity is to say that a partial computable function φ that "predicts" $A(n)$ for infinitely many n must make a mistake somewhere.

Bi-immunity



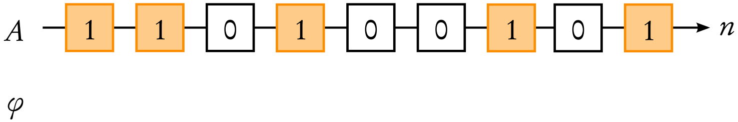
1. Another way of seeing bi-immunity is to say that a partial computable function φ that "predicts" $A(n)$ for infinitely many n must make a mistake somewhere.
2. **Theorem (Jockusch).** There exists a non-computable set A such that there is no bi-immune set B that is Turing-equivalent to A.
3. **Intuition.** Some information just cannot be represented in a bi-immune way.

Weakening bi-immunity

1. The notion of bi-immunity can be weakened by replacing "φ's that make infinitely many predictions" by a smaller class.
2. **Definition.** The *(upper) density of* $D \subseteq \mathbb{N}$ is

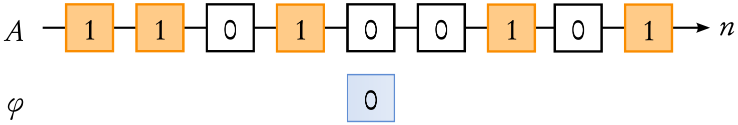
$$\rho(D) := \limsup_{n\to\infty} \frac{|D \cap \{0,\ldots n-1\}|}{n}.$$

Absolute undecidability



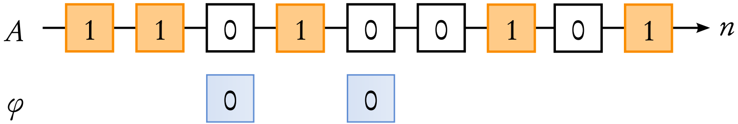
1 **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi : \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.

Absolute undecidability



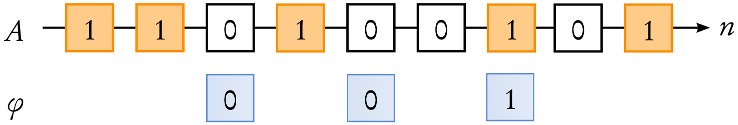
1 **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi\colon \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.

Absolute undecidability



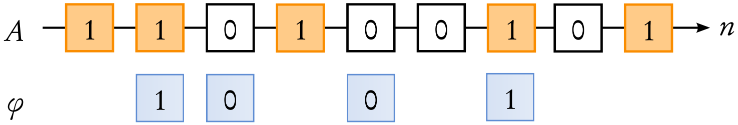
1 **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi\colon \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.

Absolute undecidability



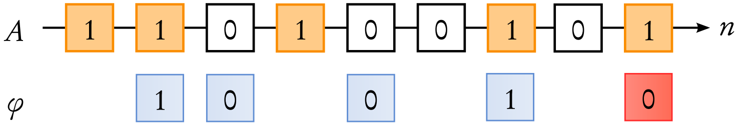
1 **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi : \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.

Absolute undecidability



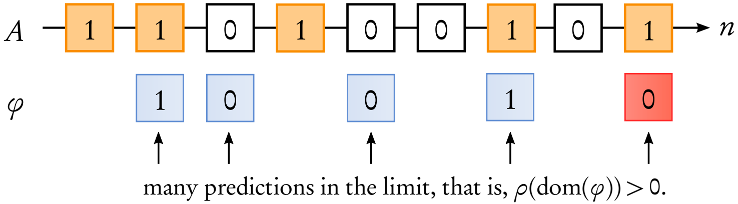
1 **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi : \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.

Absolute undecidability



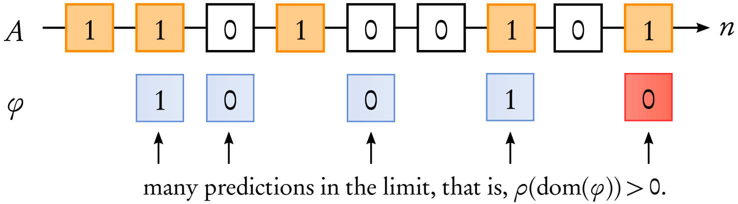
1 **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi : \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.

Absolute undecidability



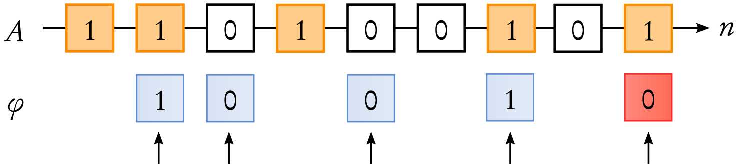
1 **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi\colon \mathbb{N} \to \{0,1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.

Absolute undecidability



1. **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi : \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.
2. **Intuition.** Like bi-immunity, except that only those φ's that make "many" predictions are required to make mistakes.

Absolute undecidability



many predictions in the limit, that is, $\rho(\mathrm{dom}(\varphi)) > 0$.

1. **Definition (Myasnikov, Rybalov).** A is *absolutely undecidable* if there is no partial computable function $\varphi\colon \mathbb{N} \to \{0, 1\}$ with $\rho(\mathrm{dom}(\varphi)) > 0$ and $\varphi(n) = A(n)$ for $n \in \mathrm{dom}(\varphi)$.
2. **Intuition.** Like bi-immunity, except that only those φ's that make "many" predictions are required to make mistakes.
3. **Intuition.** There is no Turing machine generating non-negligible positive or negative information about A.

In the Turing degrees

1. **Theorem (Jockusch), restated.** There exists a non-computable Turing degree such that none of its elements are bi-immune.
2. **Question (Downey, Jockusch, Schupp).** Does there exist a non-computable Turing degree such that none of its elements are absolutely undecidable?

In the Turing degrees

1. **Theorem (Jockusch), restated.** There exists a non-computable Turing degree such that none of its elements are bi-immune.
2. **Question (Downey, Jockusch, Schupp).** Does there exist a non-computable Turing degree such that none of its elements are absolutely undecidable?
3. We will show that the answer is **“no”** — bi-immunity and absolute undecidability behave differently in this regard.

3 The main result

Main result

1. **Theorem.** There exists a *tt*-functional Γ such that for non-computable A, Γ^A is absolutely undecidable and $\Gamma^A \equiv_T A$.
2. **Corollary.** There is an absolutely undecidable set in every non-computable Turing degree.

General proof idea

1. The functional Γ will code any set A in way that is so redundant, that from any non-negligible fraction of that code Γ^A the whole set A can be recovered.
2. Assume for contradiction that Γ^A is *not* absolutely undecidable. Then there is a φ as above.
3. Since φ is partial computable, we could then use φ to generate such a non-negligible fraction, and then recover A.
4. Then A would be computable, contradiction.
5. So Γ^A must have been absolutely undecidable. $\square$

Walsh-Hadamard codes

1. For $x, y \in \{0,1\}^n$ let $x \odot y = \sum_{i=1}^{n} x_i y_i \bmod 2$.
2. Then the Walsh-Hadamard code of a word $x \in \{0,1\}^n$ is

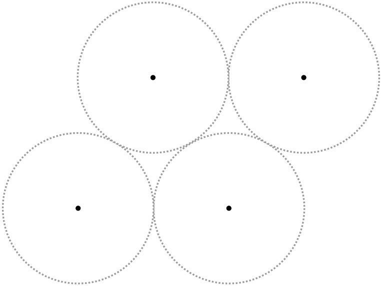
$$WH(x) := x \odot 0^n \;\circ\; x \odot 0^{n-1}1 \;\circ\; x \odot 0^{n-2}10 \;\circ\; \ldots \;\circ\; x \odot 1^n,$$

where $\circ$ denotes concatenation.

List decoding

1. **Hamming distance.** Define $d(x,y) := \#\{i \mid x(i) \neq y(i)\}/n$. The *distance* of a coding scheme E is $\min\{d(E(x),E(y)) \mid x \neq y\}$.

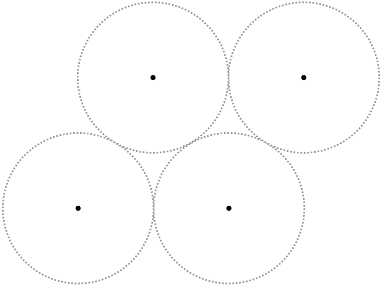
List decoding



(objects in drawing are higher-dimensional than they appear)

1. **Hamming distance.** Define $d(x,y) := \#\{i \mid x(i) \neq y(i)\}/n$. The *distance* of a coding scheme E is $\min\{d(E(x),E(y)) \mid x \neq y\}$.

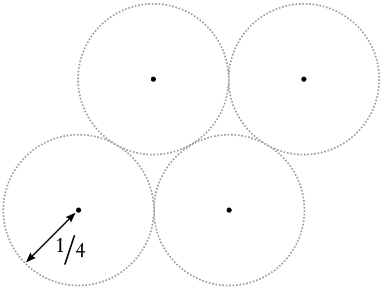
List decoding



(objects in drawing are higher-dimensional than they appear)

1. **Hamming distance.** Define $d(x,y) := \#\{i \mid x(i) \neq y(i)\}/n$. The *distance* of a coding scheme E is $\min\{d(E(x),E(y)) \mid x \neq y\}$.
2. **Lemma.** *WH* is an error correcting code of distance $1/2$.

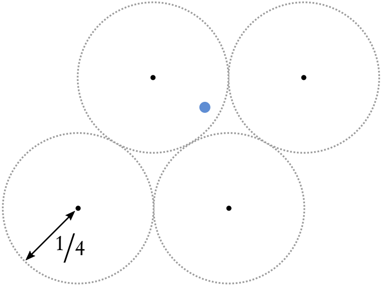
List decoding



(objects in drawing are higher-dimensional than they appear)

1. **Hamming distance.** Define $d(x, y) := \#\{i \mid x(i) \neq y(i)\}/n$. The *distance* of a coding scheme E is $\min\{d(E(x), E(y)) \mid x \neq y\}$.
2. **Lemma.** *WH* is an error correcting code of distance 1/2.

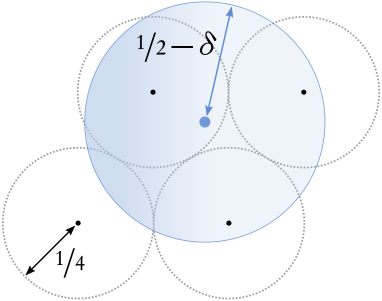
List decoding



(objects in drawing are higher-dimensional than they appear)

1. **Hamming distance.** Define $d(x, y) := \#\{i \mid x(i) \neq y(i)\}/n$. The *distance* of a coding scheme E is $\min\{d(E(x), E(y)) \mid x \neq y\}$.
2. **Lemma.** *WH* is an error correcting code of distance 1/2.

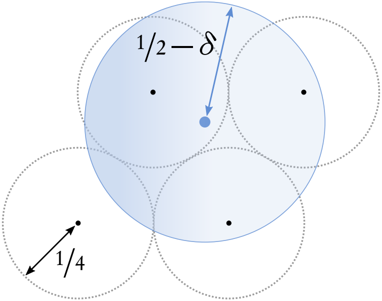
List decoding



(objects in drawing are higher-dimensional than they appear)

1. **Hamming distance.** Define $d(x,y) := \#\{i \mid x(i) \neq y(i)\}/n$. The *distance* of a coding scheme E is $\min\{d(E(x),E(y)) \mid x \neq y\}$.
2. **Lemma.** *WH* is an error correcting code of distance $1/2$.

List decoding



(objects in drawing are higher-dimensional than they appear)

1. **Hamming distance.** Define $d(x,y) := \#\{i \mid x(i) \neq y(i)\}/n$. The *distance* of a coding scheme E is $\min\{d(E(x),E(y)) \mid x \neq y\}$.
2. **Lemma.** *WH* is an error correcting code of distance $1/2$.
3. **Johnson bound.** If E is an error correcting code of distance larger or equal to $1/2$, then for all x and $\delta \geq 0$, there are at most $l = 1/2\delta^2$ elements $y_1, \ldots, y_l$ with $d(x, E(y_i)) \leq 1/2 - \delta$ for all i.

A subtlety

1. We are not quite in the setting of error-correcting codes.
2. In that field, usually a code gets damaged by switching bits.
3. Here, bits are *missing*; say, a $1-2\delta$ fraction of them.
4. That is, the 2δ fraction of non-missing bits is correct.
5. Then the error-correcting code approach can still be used:
   - Fill the empty positions with 0's to get a codeword z_0 and with 1's to get a codeword z_1.
   - One of them must be correct on $1/2+\delta$ of its bits.
   - Use list decoding on both z_0 and z_1.
   - Get two lists of size $l = 1/2\delta^2$.
   - Merge them.

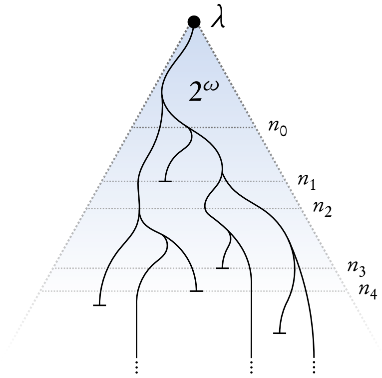
The coding procedure

1. For input A, we construct Γ^A block by block.
2. An initial segment $A \upharpoonright n$ is coded by a string of length 2^n.
3. Namely, $\Gamma^A = WH(A \upharpoonright 1) \circ WH(A \upharpoonright 2) \circ WH(A \upharpoonright 3) \dots$

The recovery procedure

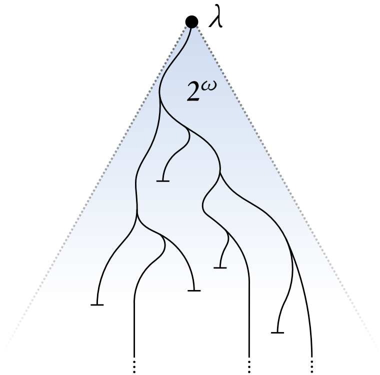
1. Now assume we know a positive upper density fraction 2δ of the bits of Γ^A. W.l.o.g. choose $\delta \in \mathbb{Q}$. Let $I_n = \{2^n, \dots, 2^{n+1} - 1\}$.
2. **Lemma.** If a set $D \subseteq \mathbb{N}$ has $\rho(D) \geq 2\delta > 0$, then for infinitely many n, the upper density of D inside I_n is at least δ.
3. Since D is c.e. and $\delta \in \mathbb{Q}$, the set of such n is c.e. and therefore contains a computable set $\{n_0 < n_1 < n_2 < \dots\}$.
4. Let Γ_0^A be a version of Γ^A where missing bits are filled with 0's. Let Γ_1^A be a version of Γ^A where missing bits are filled with 1's.
5. For all i, let $\sigma_{n_i}^0 := \Gamma_0^A \upharpoonright I_{n_i}$ and $\sigma_{n_i}^1 := \Gamma_1^A \upharpoonright I_{n_i}$.
6. Apply list decoding to these two corrupted codewords.
7. Merge the resulting lists, as discussed above.

The recovery procedure



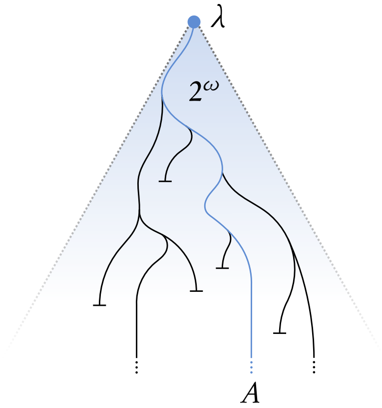
1. We get a computable tree whose paths are candidates for A.
2. Johnson bound $\Rightarrow$ width of tree is bounded by $2l = 1/\delta^2$.

The recovery procedure



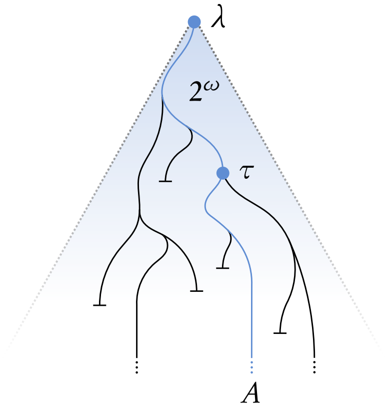
1. We get a computable tree whose paths are candidates for A.
2. Johnson bound ⇒ width of tree is bounded by $2l = 1/\delta^2$.

The recovery procedure



1. We get a computable tree whose paths are candidates for A.
2. Johnson bound $\Rightarrow$ width of tree is bounded by $2l = 1/\delta^2$.

The recovery procedure



1. We get a computable tree whose paths are candidates for A.
2. Johnson bound $\Rightarrow$ width of tree is bounded by $2l = 1/\delta^2$.
3. Hardcode a node τ where A becomes isolated in that tree. □

4 The limits

Non-uniformity

1. **Theorem.** There is no *tt*-functional Γ and finite set of Turing functionals $\Psi_1, \Psi_2, \ldots, \Psi_k$ with the property that for any A, for any partial function $\varphi\colon \mathbb{N} \to \{0,1\}$ with $\rho(\mathrm{dom}(\varphi)) \geq 1/3$, if Γ^A extends φ, then $A \in \{\Psi_i(\varphi) \mid i \leq k\}$.
2. **That is:** There is no coding that will work with a finite number of decoding procedures. In this sense our main result is optimal; the "infinite non-uniformity" for decoding is necessary.

Sublinear density

1. Let $\varkappa_D : n \mapsto |D \cap \{0, \ldots, n-1\}|$.
2. **Theorem.** There is a non-computable set X such that for all $Y \leq_T X$, and all computable functions $h \in o(n)$, there exists a partial computable function φ such that
   - $Y(n) = \varphi(n)$ for all $n \in \mathrm{dom}(\varphi)$, and
   - $\varkappa_{\mathrm{dom}(\varphi)} \notin o(h)$.
3. **Intuition.** There is a non-computable Turing degree such that for every set in it there is a correct prediction procedure making *sub*linearly, but arbitrarily close to linearly, many predictions.
4. **That is:** We really need positive density for the main result.

Thank you for your attention.

Journal of Symbolic Logic, Volume 78, Issue 4, 2013