A Taste of Recursion Theory

Aiden Sagerman

August 2021

Table of Contents

Recursive Sets

Notation

We write ω *for the set of natural numbers* $\mathbb{N} = \{0, 1, 2, \dots\}$.

Recursive Sets

Notation

We write ω *for the set of natural numbers* $\mathbb{N} = \{0, 1, 2, \dots\}$.

Definition (Recursive Sets and Functions)

We say a set $A \subseteq \omega$ is *recursive* or *computable* if there exists an algorithm which always terminates and which determines whether $x \in A$ for any $x \in \omega$.

Recursive Sets and the Church-Turing Thesis

Remark

The Church-Turing thesis, which we discussed last week, tells us that a function is recursive when it can be computed by a Turing machine.
Since any reasonable programming language is Turing complete, we can treat recursive sets as "sets which can be computed by a computer program" and recursive functions as "functions which can be computer by a computer program."

Programs

Definition (Programs)

Let $\mathcal{L} = \{a, b, c, \ldots, z, ., :, \ldots, \text{new line}\}$. A *program* is a string $P \in \mathcal{L}^{<\omega}$ such that P compiles in your chosen language.

Programs

Definition (Programs)

Let $\mathcal{L} = \{a, b, c, \ldots, z, ., :, \ldots, \text{new line}\}$. A *program* is a string $P \in \mathcal{L}^{<\omega}$ such that P compiles in your chosen language.

We will also suppose the following:

1. Every program takes an input $a \in \omega$
2. There is a function `output` which returns an output in ω.
3. If a program P halts but does not return an output, say it outputs 0.

Programs

Examples

1. `input` x
   `output` $x + 5$

Programs

Examples

1.
```
input x
output x + 5
```

2.
```
input x
if x = 1:
    output 1
else:
    output 37
```

Programs

Examples

1.
```
input x
output x + 5
```

2.
```
input x
if x = 1:
    output 1
else:
    output 37
```

3.
```
input x
while x ≥ 0 :
        x = x + 1
```

Indices for Programs

Proposition

There are countably many programs.

Indices for Programs

Proposition

There are countably many programs.

Proof.

We know that $\mathcal{L}^{<\omega}$ is countable, as it is a countable union of the finite sets $\mathcal{L}, \mathcal{L}^2, \ldots$. Since the set of programs is contained in $\mathcal{L}^{<\omega}$, it is countable as well. □

Indices for Programs

Notation

Since there are countably many programs, we may write them $P_0, P_1, P_2, \ldots$.

Definition

Let $\{e\}$ denote the function which maps an input x to the first `output` call given by the program P_e (or 0, if P_e terminates but does not give an output).

If $\{e\}$ converges on an input x, we write $\{e\}(x)\downarrow$. If we want to specify that $\{e\}$ converges to y on an input x, we write $\{e\}(x)\downarrow= y$.

Indices for Programs

Warning

Since programs do not necessarily terminate, $\{e\}$ does not necessarily have domain ω. For example, let P_e denote the program

input x
while $x \geq 0$ **:**

Since P_e terminates for no inputs, the domain of $\{e\}$ is $\emptyset$.

If a program P_e does not terminate—that is, diverges—on an input x, we write $\{e\}(x)\uparrow$

Computable Sets

Definition (Computable Sets, again)

A set $A \subseteq \omega$ is computable if there exists some $e \in \omega$ such that $\{e\}(x) = 1$ iff $x \in A$ and $\{e\}(x) = 0$ otherwise.

Examples of Computable Sets

Example

Question: Is the set $\{1, 2, 3\}$ computable?

Examples of Computable Sets

Example

Question: Is the set $\{1, 2, 3\}$ computable?

Answer: It is. Consider the following program:

```
input x
if x = 1 or x = 2 or x = 3 :
    output 1
else:
    output 0
```

Examples of Computable Sets

Example

Question: Is the set of even natural numbers computable?

Examples of Computable Sets

Example

Question: Is the set of even natural numbers computable?

Answer: It is. Consider the following program:

input x

for $y < x$:

    if $2y = x$:

        output 1

else:

    output 0

Examples of Computable Sets

Example

Other computable sets include:

1. Any finite or cofinite set.
2. ω and $\emptyset$.
3. The set of prime numbers.
4. The complement of any computable set.

Examples of Computable Sets

Example

Other computable sets include:

1. Any finite or cofinite set.
2. ω and $\emptyset$.
3. The set of prime numbers.
4. The complement of any computable set.

Remark

The vast majority of the sets we care about in mathematics are computable.

Uncomputable Sets

Proposition

Almost all sets of natural numbers are uncomputable.

Uncomputable Sets

Proposition

Almost all sets of natural numbers are uncomputable.

Proof.

We know from a variation of Cantor's diagonal argument that the there are uncountably many subsets of ω. But we have already shown that there are countably many programs P_e, so there are only countably many $A \subseteq \omega$ such that A is computed by P_e. □

The Halting Problem

Theorem

Define a set $K \subseteq \omega$ by

$$K = \{e \in \omega : \{e\}(e)\downarrow\}$$

That is, K is the list of indices e such that the eth program P_e converges on the input e. Then K is not computable.

This is called the halting problem. A version of this result was proved independently by both Alonzo Church and Alan Turing in 1936.

The Halting Problem

Proof Sketch.

	P_0	P_1	P_2	$\dots$
0	1	0	$\uparrow$	$\dots$
1	$\uparrow$	$\uparrow$	3	$\dots$
2	0	0	0	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\square$

The Halting Problem

Proof Sketch.

	P_0	P_1	P_2	...
0	1	0	$\uparrow$	...
1	$\uparrow$	$\uparrow$	3	...
2	0	0	0	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

□

The Halting Problem

Proof Sketch.

	P_0	P_1	P_2	$\dots$
0	$\not 1\ \uparrow$	0	$\uparrow$	$\dots$
1	$\uparrow$	$\not\uparrow\ 0$	3	$\dots$
2	0	0	$\not 0\ \uparrow$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

□

The Halting Problem

Proof.

Suppose K is computable by a program P_e. Then we can define the following program:

The Halting Problem

Proof.

Suppose K is computable by a program P_e. Then we can define the following program:

input x
run P_e on x
if P_e outputs 0:
    output 1
if P_e outputs 1
    diverge

The Halting Problem

Proof.

Suppose K is computable by a program P_e. Then we can define the following program:

```
input x
run P_e on x
if P_e outputs 0:
    output 1
if P_e outputs 1
    diverge
```

Call the above program P_f. Is $f \in K$? □

The Halting Problem

Proof.

First suppose $f \notin K$. Then $\{f\}(f)$ diverges, which happens only when P_e outputs 1 with the input f. But this is the same as saying $f \in K$, which is a contradiction.
Now suppose $f \in K$. This means that $\{f\}(f)$ converges, which happens when P_e outputs 0 with the input f. But this is the same as saying $f \notin K$, which is a contradiction. □

Relative Computability

Notice our program P_f is essentially computing the set

$$D = \{e \in \omega : \{e\}(e) \uparrow\} = \omega \setminus K$$

We did this assuming that we could black box a program P_e which computed K. Can we formalize the idea of computing one uncomputable set from another?

Relative Computability

Question:

Can every uncomputable set be “computed from” every other uncomputable set?

Relative Computability

Question:

Can every uncomputable set be “computed from” every other uncomputable set?

Answer

No, they cannot.

To see this, we first need to define what it means to be “computed from” another set.

Oracles

Definition

As before, let $\mathcal{L} = \{a, b, c, \ldots, z, ., :, \ldots, \text{new line}\}$ be our set of symbols. We will add a new function $\texttt{orc}(x)$, called an *oracle*, to our language.

Formally, define $\mathcal{L}' = \mathcal{L} \cup \{\texttt{orc}(x)\}$. Our programs will now instead be elements of $\mathcal{L}'^{<\omega}$ which compile in our chosen language. When the compiler encounters the $\texttt{orc}(x)$ function, the function will return either 0 or 1.

Oracles

Definition

As before, let $\mathcal{L} = \{a, b, c, \ldots, z, ., :, \ldots, \text{new line}\}$ be our set of symbols. We will add a new function $\mathtt{orc}(x)$, called an *oracle*, to our language.
Formally, define $\mathcal{L}' = \mathcal{L} \cup \{\mathtt{orc}(x)\}$. Our programs will now instead be elements of $\mathcal{L}'^{<\omega}$ which compile in our chosen language. When the compiler encounters the $\mathtt{orc}(x)$ function, the function will return either 0 or 1.

Remark

We can still define countably many programs $P_0, P_1, \ldots,$ but this will now be a slightly different list due to $\mathtt{orc}$.

Oracles

Definition

Let $A, B \subseteq \omega$. We say that A is *relatively computable from* B or *Turing reducible to* B if there exists a program P_e such that P_e computes A, assuming that the oracle correctly answers questions about membership in B.

The function given by a program P_e which has an oracle that answers questions about a set B is written $\{e\}^B$. Then A is computable from B when there exists some $e \in \omega$ such that $\{e\}^B$ is the characteristic function for A.

Oracles

Definition

Let $A, B \subseteq \omega$. We say that A is *relatively computable from B* or *Turing reducible to B* if there exists a program P_e such that P_e computes A, assuming that the oracle correctly answers questions about membership in B.

The function given by a program P_e which has an oracle that answers questions about a set B is written $\{e\}^B$. Then A is computable from B when there exists some $e \in \omega$ such that $\{e\}^B$ is the characteristic function for A.

This is a little complicated, so let's look at an example.

Oracle Examples

Example

Let K and D as before. We will show that D is Turing reducible to K (that is, relatively computable from K). Define a program P_e as follows:

```
input x
if orc(x) = 1:
    output 0
else:
    output 1
```

Now, what this program does depends on orc. But if the oracle answers questions about K—that is, if $\texttt{orc}(x) = 1$ when $x \in K$ and $\texttt{orc}(x) = 0$ when $x \notin K$—then this program will compute D.

More Oracle Examples

Examples

1. Every recursive set is Turing reducible to every other recursive set, as they can be computed without ever calling the oracle.

More Oracle Examples

Examples

1. Every recursive set is Turing reducible to every other recursive set, as they can be computed without ever calling the oracle.
2. Every recursive set is Turing reducible to K, as they can be computed without calling the oracle. (In fact, they're Turing reducible to every set for the same reason.)

More Oracle Examples

Examples

1. Every recursive set is Turing reducible to every other recursive set, as they can be computed without ever calling the oracle.
2. Every recursive set is Turing reducible to K, as they can be computed without calling the oracle. (In fact, they're Turing reducible to every set for the same reason.)
3. However, K is not reducible to any recursive set. Since this one is a little complicated, we will prove it separately.

More Oracle Examples

Proof.

Suppose we could compute K from a recursive set A. This means that there exists a problem P_e which computes K with A in the oracle (or $\{e\}^A = K$, if we identify K with its characteristic function).

More Oracle Examples

Proof.

Suppose we could compute K from a recursive set A. This means that there exists a problem P_e which computes K with A in the oracle (or $\{e\}^A = K$, if we identify K with its characteristic function).

Since A is recursive, there exists a program P_f such that P_f computes A. But this means that P_f acts exactly the same as an oracle that answers questions about A. Then we can define a new program $P_{e'}$ which is exactly the same as P_e, but which replaced every call to the oracle with running P_f. Then $P_{e'}$ would compute K, so K is recursive, which is a contradiction. □

Turing Equivalence

Notation

If A is Turing reducible to B, we write $A \leq_T B$.
If $A \leq_T B$ and $B \leq_T A$, we write $A \equiv_T B$.

Turing Equivalence

Notation

If A is Turing reducible to B, we write $A \leq_T B$.
If $A \leq_T B$ and $B \leq_T A$, we write $A \equiv_T B$.

Theorem

The relation $\equiv_T$ is an equivalence relation on $\mathcal{P}(\omega)$.

Turing Equivalence

Proof.

Reflexivity and symmetry are left as an excercise.

Turing Equivalence

Proof.

Reflexivity and symmetry are left as an excercise.
To see transitivity, suppose $A \leq_T B$ and $B \leq_T C$. This means there is a program P_e that computes A when the oracle answers questions about B, and a program P_f that computes B when the oracle answers questions about C.

Turing Equivalence

Proof.

Reflexivity and symmetry are left as an excercise.
To see transitivity, suppose $A \leq_T B$ and $B \leq_T C$. This means there is a program P_e that computes A when the oracle answers questions about B, and a program P_f that computes B when the oracle answers questions about C.
Suppose the oracle answers questions about C. Then the program P_f tells us whether something is in B or not—that is, it functions exactly like an oracle that answers questions about B.

Turing Equivalence

Proof.

Reflexivity and symmetry are left as an excercise.
To see transitivity, suppose $A \leq_T B$ and $B \leq_T C$. This means there is a program P_e that computes A when the oracle answers questions about B, and a program P_f that computes B when the oracle answers questions about C.
Suppose the oracle answers questions about C. Then the program P_f tells us whether something is in B or not—that is, it functions exactly like an oracle that answers questions about B.
Thus take P_e and define a new program $P_{e'}$ which is exactly P_e, but which replaces every instance of $\texttt{orc}(x)$ with P_f run on input x. Then $P_{e'}$ computes A. □

Turing Equivalence

Corollary

The set $\mathcal{D} = \mathcal{P}(\omega)/\equiv_T$ is well-defined, and $(\mathcal{D}, \leq_T)$ is a partially ordered set.
We call $\mathcal{D}$ the set of Turing degrees, *and each element $\mathbf{a} \in \mathcal{D}$ a* Turing degree.

What Do We Know About $\mathcal{D}$?

What do we know about $\mathcal{D}$?

What Do We Know About $\mathcal{D}$?

What do we know about $\mathcal{D}$?

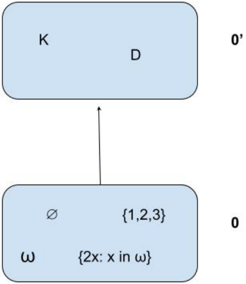
1. We know that if A and B are recursive sets, $A \equiv_T B$, so $\mathcal{D}$ has at least one degree. We call the degree that contains the recursive sets $\mathbf{0}$.

What Do We Know About $\mathcal{D}$?

What do we know about $\mathcal{D}$?

1. We know that if A and B are recursive sets, $A \equiv_T B$, so $\mathcal{D}$ has at least one degree. We call the degree that contains the recursive sets $\mathbf{0}$.
2. We know that $K \geq_T A$ for any recursive A, but $K \not\equiv_T A$. Then there must be at least one other Turing degree, which is called $\mathbf{0}'$, and which satisfies $\mathbf{0} \leq_T \mathbf{0}'$.

What Do We Know About $\mathcal{D}$?



What Do We Know About $\mathcal{D}$?

What do we *not* know about $\mathcal{D}$?

What Do We Know About $\mathcal{D}$?

What do we *not* know about $\mathcal{D}$?

1. Is $\mathcal{D}$ finite or infinite? Countable or uncountable?

What Do We Know About $\mathcal{D}$?

What do we *not* know about $\mathcal{D}$?

1. Is $\mathcal{D}$ finite or infinite? Countable or uncountable?
2. Is $\mathcal{D}$ totally ordered (for every $\mathbf{a}, \mathbf{b} \in \mathcal{D}$, either $\mathbf{a} \leq_T \mathbf{b}$ or $\mathbf{b} \leq_T \mathbf{a}$)? What about well-ordered?

What Do We Know About $\mathcal{D}$?

What do we *not* know about $\mathcal{D}$?

1. Is $\mathcal{D}$ finite or infinite? Countable or uncountable?
2. Is $\mathcal{D}$ totally ordered (for every $\mathbf{a}, \mathbf{b} \in \mathcal{D}$, either $\mathbf{a} \leq_T \mathbf{b}$ or $\mathbf{b} \leq_T \mathbf{a}$)? What about well-ordered?
3. If $\mathcal{D}$ is not totally ordered, how many incomparable elements are there?

What Do We Know About $\mathcal{D}$?

What do we *not* know about $\mathcal{D}$?

1. Is $\mathcal{D}$ finite or infinite? Countable or uncountable?
2. Is $\mathcal{D}$ totally ordered (for every $\mathbf{a}, \mathbf{b} \in \mathcal{D}$, either $\mathbf{a} \leq_T \mathbf{b}$ or $\mathbf{b} \leq_T \mathbf{a}$)? What about well-ordered?
3. If $\mathcal{D}$ is not totally ordered, how many incomparable elements are there?
4. Is $\mathcal{D}$ dense? If not, are there minimal elements (above $\mathbf{0}$)?

The Jump Operator

Definition

Given $A \subseteq \omega$, the *jump* of A, written A', is the set $\{e \in \omega : \{e\}^A(e) \downarrow\}$.
Note that K from before is in fact $\emptyset'$.

The Jump Operator

Definition

Given $A \subseteq \omega$, the *jump* of A, written A', is the set $\{e \in \omega : \{e\}^A(e) \downarrow\}$.
Note that K from before is in fact $\emptyset'$.

Theorem

The jump operator is well-defined on degrees. That is, if $A, B \in \mathbf{a}$, then $A' \equiv_T B'$.

The Jump Operator

Definition

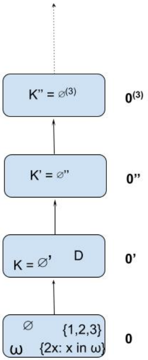
Given $A \subseteq \omega$, the *jump* of A, written A', is the set $\{e \in \omega : \{e\}^A(e)\downarrow\}$.
Note that K from before is in fact $\emptyset'$.

Theorem

The jump operator is well-defined on degrees. That is, if $A, B \in \mathbf{a}$, then $A' \equiv_T B'$.

This means that we can generate infinitely many degrees by taking $\mathbf{0}, \mathbf{0}', \mathbf{0}'', \ldots$.

The Jump Operator



Incomparable Degrees

Theorem

There exist two incomparable degrees. That is, there exist two degrees $\mathbf{a}, \mathbf{b} \in \mathcal{D}$ *such that* $\mathbf{a} \not\leq_T \mathbf{b}$ *and* $\mathbf{b} \not\leq_T \mathbf{a}$.

Incomparable Degrees

Theorem

There exist two incomparable degrees. That is, there exist two degrees $\mathbf{a}, \mathbf{b} \in \mathcal{D}$ *such that* $\mathbf{a} \not\leq_T \mathbf{b}$ *and* $\mathbf{b} \not\leq_T \mathbf{a}$.

Remark

It is sufficient to show that there are two sets $A, B \subseteq \omega$ such that $A \not\leq_T B$ and $A \not\leq_T B$. This is equivalent to saying that for each program P_e, P_e doesn't compute A from B or B from A.

Incomparable Degrees

Theorem

There exist two incomparable degrees. That is, there exist two degrees $\mathbf{a}, \mathbf{b} \in \mathcal{D}$ *such that* $\mathbf{a} \not\leq_T \mathbf{b}$ *and* $\mathbf{b} \not\leq_T \mathbf{a}$.

Remark

It is sufficient to show that there are two sets $A, B \subseteq \omega$ such that $A \not\leq_T B$ and $A \not\leq_T B$. This is equivalent to saying that for each program P_e, P_e doesn't compute A from B or B from A. We can think of this as two lists of infinitely many requirements that A and B have to satisfy:

1. R_e: $A \neq \{e\}^B$
2. R'_e: $B \neq \{e\}^A$

Incomparable Degrees

Proof Sketch.

	0	1	2	...
A_0	0	0	0	...
B_0	0	0	0	...

□

Incomparable Degrees

Proof Sketch.

	0	1	2	...
A_0	0	0	0	...
B_0	0	0	0	...

Is there some finite set $\sigma \supseteq B_0$ such that $\{0\}^{\sigma}(0) = 1$ or 0? □

Incomparable Degrees

Proof Sketch.

If yes, we make sure A_1 does the opposite, and make B_1 into that σ.

Incomparable Degrees

Proof Sketch.

If yes, we make sure A_1 does the opposite, and make B_1 into that σ.

For example, let's say that $\sigma = \{0, 2\}$, and it converges to 0. Then we alter A_0.

	0	1	2	...
A_1	1	0	0	...
B_1	1	0	1	...

This ensures that P_1 won't compute the if $0 \in A$ correctly from B. □

Incomparable Degrees

Proof Sketch.

If not, it doesn't matter, and we can just leave A_1 and B_1 as is.

Incomparable Degrees

Proof Sketch.

If not, it doesn't matter, and we can just leave A_1 and B_1 as is.

	0	1	2	...
A_1	0	0	0	...
B_1	0	0	0	...

This is because nothing we could possibly do to B will allow P_0 to compute if $0 \in A$. □

Incomparable Degrees

Proof Sketch.

Then we look at B_1, and do the same thing!

	0	1	2	3	...
A_1	1	0	0	0	...
B_1	1	0	1	0	...

Assuming the first case, the first unused element of B is 3, so we would ask: Is there some finite set $\sigma \supseteq A_1$ such that $\{0\}^\sigma(3) = 1$ or 0?

By looking at the answers to these two questions, we're ensuring R_0 and R'_0. □

Incomparable Degrees

Proof.

We will build our sets in stages. At each stage, we start with two finite sets A_s and B_s, and extend them to A_{s+1} and B_{s+1}. On stage $s+1$, we will make sure R_s and R'_s are satisfied.

Incomparable Degrees

Proof.

We will build our sets in stages. At each stage, we start with two finite sets A_s and B_s, and extend them to A_{s+1} and B_{s+1}. On stage $s+1$, we will make sure R_s and R'_s are satisfied.

1. Start with $A_0 = B_0 = \emptyset$.

Incomparable Degrees

Proof.

We will build our sets in stages. At each stage, we start with two finite sets A_s and B_s, and extend them to A_{s+1} and B_{s+1}. On stage $s+1$, we will make sure R_s and R'_s are satisfied.

1. Start with $A_0 = B_0 = \emptyset$.
2. Suppose we are on stage $s+1$. Let x_{s+1} denote the first number not yet added to A_s. Then we ask the question: "Does there exist some set σ such that $B_s \subseteq \sigma$ and $\{s\}^{\sigma}(x_{s+1}) \downarrow$ to 1 or 0?"

□

Incomparable Degrees

Proof.

2. If the answer to the question is "yes," then we can make $B_{s+1} = \sigma$, and extend A_s with either 0 if it converges to 1 or 1 otherwise.

Incomparable Degrees

Proof.

2. If the answer to the question is "yes," then we can make $B_{s+1} = \sigma$, and extend A_s with either 0 if it converges to 1 or 1 otherwise.
3. Then we do the exact same process, but exchanging the role of A_s and B_s.

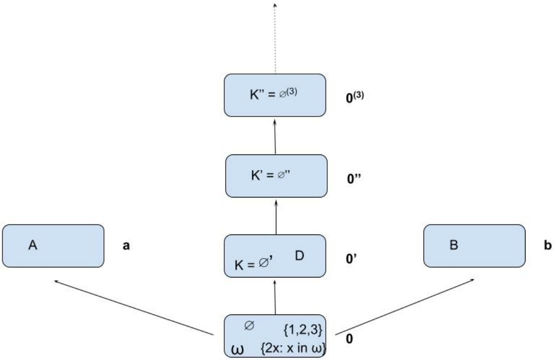
Incomparable Degrees

Proof.

2. If the answer to the question is "yes," then we can make $B_{s+1} = \sigma$, and extend A_s with either 0 if it converges to 1 or 1 otherwise.
3. Then we do the exact same process, but exchanging the role of A_s and B_s.

Since we ensured R_0 and R'_0 on step 1, R_1 and R'_1 on step $2, \ldots$, we know that A and B will be incomparable! □

Incomparable Degrees



Other Structure Results

Using similar (and sometimes more complicated) tools, we can show the following:

1. For any degree $\mathbf{a} >_T \mathbf{0}$, there is a $\mathbf{b} \in \mathcal{D}$ such that $\mathbf{a}$ and $\mathbf{b}$ are incomparable.
2. There are minimal degrees—that is, degrees $\mathbf{a} \in \mathcal{D}$ such that there exist no $\mathbf{b}$ with $\mathbf{0} <_T \mathbf{b} <_T \mathbf{a}$.
3. For any degree $\mathbf{b} >_T \mathbf{0}'$, there exists a degree $\mathbf{a}$ such that $\mathbf{a}' = \mathbf{b}$.
4. There exists a set of $2^{\aleph_0}$ incomparable degrees.

References

Horowitz, Jason. *Recursion Theory*, lecture notes, Proof School, delivered May and June 2020.

Kunen, Kenneth. *The Foundations of Mathematics* . College Publications, 2009.