UCT

Computability, Fundamentals

Klaus Sutner

Carnegie Mellon University

Spring 2024

Where Are We?

- Turing machines provide the reference model of computability.
- Every Turing machine can be associated with an index $e \in \mathbb{N}$.
- There is a universal Turing machine $\mathcal{U}$ that can simulate any Turing machine given its index.
- The Halting problem "does $\mathcal{U}(e,e)\downarrow$?" is the prime example of an undecidable problem.
- Halting is difficult to determine even for very small machines.
- Halting gives rise to the class of semidecidable problem.

Index Computation

Recall the enumeration $\{e\}$ of all partial computable functions. Having an index e that represents a function makes it possible to manipulate functions by manipulating their indices instead.

For example, there is an easily computable function f such that $f(e, e')$ is an index for a machine $\mathcal{M}$ that composes $\mathcal{M}_e$ and $\mathcal{M}_{e'}$: first run $\mathcal{M}_e$, if there is output, feed it to $\mathcal{M}_{e'}$ and return the output of that computation:

$$\mathcal{M}_{f(e,e')} \text{ computes } \{e\} \circ \{e'\}$$

In terms of functions:

$$\{f(e, e')\} \simeq \{e\} \circ \{e'\}$$

This is just software engineering, no problem. Well, actually OCCTM.

Kleene's S-m-n theorem

Here is another index computation: we can fix a few arguments in a computable function and obtain another computable function. Yes, this is a no-brainer, but we are trying to be a bit formal here.

Theorem

For every $m, n \geq 1$ there is a primitive recursive function S^m_n such that

$$\{S^m_n(e, \boldsymbol{p})\}^{(n)}(\boldsymbol{x}) \simeq \{e\}^{(m+n)}(\boldsymbol{p}, \boldsymbol{x}).$$

Proof.

Klar (as per number theorist E. Landau).

You can also think of this as a form of currying.

□

WTF?

From a programming perspective, this is blindingly obvious; so obvious, it might look like a typo.

But we are dealing with a formal framework, not general intuition about computation, or some semantics-free programming language.

Think about what the theorem really says: OCCTM that, given an index e and a list of integers $\boldsymbol{p}$, computes the index of a new TM that behaves like $\mathcal{M}_e$ with some of the arguments bound to $\boldsymbol{p}$. Can be done for sure, but it's the usual pain.

TM Properties

We are often interested in collections of Turing machines (or their associated functions) that have certain properties. We can think of these collections as subset of $\mathbb{N}$ via indices. Typical example:

$$\mathsf{INF} = \{\, e \in \mathbb{N} \mid \{e\} \text{ converges on infinitely many inputs} \,\}$$

Note that for $\{e\} \simeq \{e'\}$ we have $e \in \mathsf{INF} \Leftrightarrow e' \in \mathsf{INF}$, we dealing with a structural property of the machines rather than some nonsense property of the indices (say, e is divisible by 17).

We will see that such structural properties are always undecidable unless they are trivial.

Why?

For this kind of argument, it is absolutely critical to first develop a solid intuition–the technical details are then are tedious (OCCTM) but not conceptually all that difficult[†].

Key Intuition:

INF seems even harder than Halting: we need to check whether $\mathcal{M}_e(x)\downarrow$ for for infinitely many x, so in a sense we have to solve infinitely many Halting problems.

We will provide a very elegant proof in a while, here is a more pedestrian approach that is a template for many other arguments of this kind.

[†]This is a special feature of computability theory, similar claims do not hold in other areas (say, proof theory or real analysis).

Towards Hardness of INF

Suppose index e is the input. We can build another TM $\mathcal{M}$ that, on input z, will test whether run $\{e\}(e)$ halts before z steps. If so, $\mathcal{M}$ returns 0; otherwise the new machine diverges.

More precisely, there is an easily computable function f (a program transformation) such that $e' = f(e)$:

$$\{e'\}(z) \simeq \begin{cases} 0 & \text{if } \{e\}_z(e) = z, \\ \uparrow & \text{otherwise.} \end{cases}$$

Recall that $\{e\}_z(e) = z$ means: the computation has not yet finished at step z.

Check that we are really concocting a computable function with index $e' = f(e)$ and the transformation f itself is also computable.

Inheriting Undecidability

Write $K \subseteq \mathbb{N}$ for the plain Halting problem.

Note that $\{e'\}$ is constant 0 when $e \notin K$. On the other hand, if $e \in K$, $\{e\}(z)$ will converge only for all $z < s$, the number of steps it takes $\{e\}(e)$ to stop. Then

$$e \notin K \iff \{f(e)\} \text{ halts everywhere } \iff f(e) \in \mathsf{INF}$$

Since f is computable, and K is undecidable, INF cannot be decidable.

In a way, we are switching time with space in this argument.

Lower Bounds

This is just a lower bound, we know INF is not decidable, but we cannot say anything more constructive about its level of unsolvability. As it turns out, INF is not even semidecidable nor co-semidecidable. We need a better classification for this kind of problem.

Next up: a hugely powerful theorem that makes it possible to establish all kinds of results about computability in a very elegant and concise manner. It is a bit abstract, though.

Batten down the hatches.

Second Recursion Theorem

The next result is utterly amazing: it shows that we can solve functional equations of computable functions in mind-numbing generality.

Theorem (Kleene, Second Recursion Theorem, 1938)

Let $F : \mathbb{N}^{n+1} \nrightarrow \mathbb{N}$ be a computable function. Then there exists an index $e^\star$ such that for all $\boldsymbol{x} \in \mathbb{N}^n$:

$$\{e^\star\}(\boldsymbol{x}) \simeq F(e^\star, \boldsymbol{x}).$$

Moreover, $e^\star$ can be computed effectively from an index for F.

Think of $e^\star$ as some program, x as input and F as an interpreter running $e^\star$ on x: then the claim is obvious, even for arbitrary $e^\star$. However, the theorem holds for arbitrary computable F!

No Way

At first glance, this sounds utterly wrong.

What if $F(e, x) \simeq \{e\}(x) + 1$?

Then we need $\{e^\star\}(x) \simeq \{e^\star\}(x) + 1$.

Solution: Any totally undefined function $\{e^\star\}$.

This is why it's important to deal with Kleene equality $\simeq$ rather than ordinary equality $=$.

In general it requires extra effort to show that a function obtained from the theorem is, say, total. A priori, all we get is a computable function, no more. And, this computable function may be undefined in many places, including everywhere.

Weird Consequences

The theorem has some rather strange consequences:

- Let $F(e, x) \simeq e$. Then $e^\star \simeq \{e^\star\}(x)$.
- Let $F(e, x) \simeq \{x\}(e)$. Then $\{x\}(e^\star) \simeq \{e^\star\}(x)$.

These F are clearly computable, so there is no way around these conclusions–no matter how strange they look.

These examples are certainly bizarre, but not particularly useful. As it turns out, there are applications where the recursion theorem is absolutely critical.

Quines

In a similar vein, programs with the property that $e \simeq \{e\}()$ are referred to as quines, programs that print themselves.

The real challenge here is that one needs to deal with the idiosyncrasies (more often: idiocies) of a particular programming language.

Exercise

Write a quine in your favorite programming language.

The Fixed Point Version

Here is slightly more straightforward version of the recursion theorem that often comes in handy.

Corollary (Rogers, 1967)

Let $f : \mathbb{N} \to \mathbb{N}$ *be a recursive function. Then there exists an index* $e^\star$ *such that*

$$\{e^\star\}(\boldsymbol{x}) \simeq \{f(e^\star)\}(\boldsymbol{x}).$$

Proof.

Define $F(e, \boldsymbol{x}) \simeq \{f(e)\}(\boldsymbol{x})$.

$\square$

Intuition

Think of f as some program transformation. For example, $f(e)$ could be the program that does sequential composition of program e with itself.

Then, no matter how f is chosen, there will always be a program $e^\star$ for which application of f does not change the I/O behavior. So every program transformation has a fixed point, a program whose behavior does not change under the transformation.

This seems to contradict the fact that we can make the transformation change the behavior, but there is a way out: we are dealing with partial functions that can be undefined at the places where things are different. In fact, the result may well be totally undefined.

Example: Self-Composition

Suppose that

$$\{f(e)\}(z) \simeq \{e\}\big(\{e\}(z)\big)$$

So we are looking for functions that are idempotent: $g \simeq g \circ g$.

The totally undefined function works, but there are many other solutions: for example, we could replace every primefactor p^k in z by p.

Exercise

Come up with another example of a program transformation where the recursion theorem has easily explainable solutions.

The Horror Proof

We will prove the second recursion theorem directly. Define

$$h(e, \boldsymbol{x}) \simeq F(S^1_n(e, e), \boldsymbol{x})$$

Let $\widehat{h}$ be an index for h and set

$$e^\star := S^1_n(\widehat{h}, \widehat{h})$$

Then

$$\begin{aligned}
\{e^\star\}(\boldsymbol{x}) &\simeq \{S^1_n(\widehat{h}, \widehat{h})\}(\boldsymbol{x}) \\
&\simeq \{\widehat{h}\}(\widehat{h}, \boldsymbol{x}) \\
&\simeq h(\widehat{h}, \boldsymbol{x}) \\
&\simeq F(S^1_n(\widehat{h}, \widehat{h}), \boldsymbol{x}) \\
&\simeq F(e^\star, \boldsymbol{x})
\end{aligned}$$

$\square$

Quoi?

This is one of the most infuriating proofs known to mankind. Every step is trivial equational reasoning, but the whole argument makes no sense.

The recursion theorist J. C. Owings called it "barbarically short" and "nearly incapable of rational analysis."

Owings also suggested a way to make sense out of it: think of it as a diagonal argument that fails. Cute, but does not really help either.

Did I mention that Kleene was a genius?

I Know Who I Am

Here is a good intuitive way to think about the RT. The typical definition of a computable function Q looks like so:

Q:

```
input $x$
some computation
return ...
```

But with the RT we can use an index $q = \widehat{Q}$ for Q inside the definition:

Q:

```
input $x$
some computation using $q$
return ...
```

Again . . .

This may sound breathtakingly wrong: after all, we are in the process of defining Q, so where the hell is the index $q = \widehat{Q}$ supposed to come from?

Sorry, but that is precisely what the recursion theorem says we can do, with impunity.

Maybe it helps to think of Q as being stored in digital memory and being able to read its own source code? Try.

Application: Halting

Assume for the sake of a contradiction that the halting set $K = \{\, e \mid \{e\}(e)\downarrow \,\}$ is decidable.

Define Q by

```
input $x$
**if** $q \in K$        // $K$ decidable
**then** $\uparrow$
**else** return 0
```

Then $Q(q)\downarrow$ implies $Q(q)\uparrow$, and $Q(q)\uparrow$ implies $Q(q)\downarrow$.

↑↓↓↑↑↓↓↑↑↓↓↑↑↓↑↓↓↑↑↑↑↑↑↓↑↓↓↑↑↓↓↑↑↓↑↓↓↓↑↑↑↓↑↓

Application: Ackermann

Recall the definition of the Ackermann function, a function defined by a double recursion. We write z^+ for $z+1$.

$$\begin{aligned} A(0,y) &= y^+ \\ A(x^+,0) &= A(x,1) \\ A(x^+,y^+) &= A(x,A(x^+,y)) \end{aligned}$$

A is a perfect example of a Herbrand-Gödel-computable function and it is known that these are the same as Turing-computable. The Ackermann function was introduced as an example of a function that is computable, but not primitive recursive (it grows faster than any primitive recursive function).

It turns out that A is total, but that requires a proof (by double induction or induction to ω^2).

Ackermann and RT

Define Q by

```
input $x$, $y$
**if** $x = 0$
**then** return $y + 1$
**elseif** $y = 0$
        **then** return $\{q\}(x-1, 1)$
        **else** return $\{q\}(x-1, \{q\}(x, y-1))$
```

Then $Q = \{q\}$ is none other than the Ackermann function and we have another proof of its computability. Of course, one still has to work to demonstrate totality.

Enumerating RE Sets

Define

$$W_e = \{\, x \mid \{e\}(x)\downarrow \,\} \subseteq \mathbb{N}$$

to be the eth semidecidable set, so $(W_e)_{e\geq 0}$ is an effective enumeration of all semidecidable sets.

We can use this enumeration to ask questions about semidecidable sets. For example, we might be interested in all non-empty semidecidable sets:

$$\mathsf{NE} = \{\, e \mid W_e \neq \emptyset \,\}$$

This is essentially the same as Halting Somewhere, so we already know that NE is semidecidable.

Note that membership in NE is determined by the properties of W_e, not e itself (like being prime, having an odd number of digits, . . .)

Index Sets

Any such set is called an index set.

More precisely, let's say that $P \subseteq \mathbb{N}$ is a non-trivial index set if

- $W_e = W_{e'}$ implies $e \in P \Leftrightarrow e' \in P$,
- $e_Y \in P$ and $e_N \notin P$ for some e_Y and e_N.

Since an index set is just a set of natural numbers we can ask whether it is decidable, semidecidable, co-semidecidable, . . .

Application: Rice's Theorem

Typical examples are

- W_e is empty
- W_e is finite
- $W_e = \mathbb{N}$
- W_e is decidable

Claim

All of these index sets seem undecidable; in other words, all these properties of semidecidable sets are undecidable.

Rice's Theorem

One can slog one's way through separate proofs for all these properties; good exercise. For example $W_e = \mathbb{N}$ is just another way of saying $e \in \mathsf{TOT}$.

But there is a universal tool that kills them all off, in one fell swoop.

Theorem (Rice 1953)

Every non-trivial index set is undecidable. In other words, every non-trivial property of semidecidable sets is undecidable.

Proof of Rice

Let P be a non-trivial index set.

For the sake of a contradiction assume P is decidable.

Define Q by

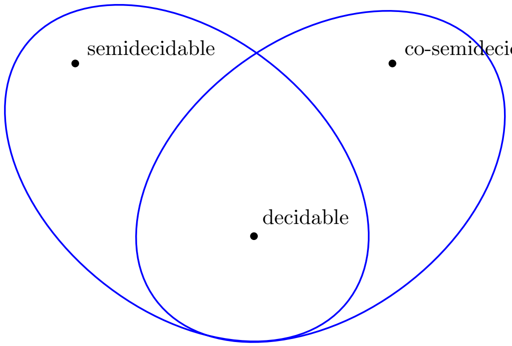
```
input $x$
**if** $q \in P$              // $P$ decidable
**then** return $\{e_N\}(x)$
**else** return $\{e_Y\}(x)$
```

But then $q \in P$ implies $Q \simeq \{e_N\}$ and thus $q \notin P$.

On the other hand, $q \notin P$ implies $Q \simeq \{e_Y\}$ and thus $q \in P$.

$\square$

Our World



So far, we have examples of sets in three distinct complexity classes.

Question: Could this be everything?

Bad Answer

As usual, we can resort to set theory and counting arguments: there are $2^{\aleph_0}$ subsets of $\mathbb{N}$.

Only countably many of them are decidable, semidecidable and co-semidecidable.

So almost all sets $A \subseteq \mathbb{N}$ must be more complicated.

True, but like all cardinality based existence arguments unsatisfactory and bordering on useless. We want concrete examples, and perhaps a nice hierarchy.

The Misery of Counting

Here is another example where mere counting provides very little information. Cantor proved in 1874 that there are uncountably many transcendental reals: the algebraic reals are countable.

At that time, the only known transcendentals were Liouville numbers and e (C. Hermite), π came a bit later (1882, by F. Lindemann).

$0.110001000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001\ldots$

Needless to say, Cantor's result has nothing to say about numbers such as

$$e + \pi \qquad \text{or} \qquad e\,\pi \qquad \text{or} \qquad e^{\pi}$$

Kleene Normal Form

In an effort to find concrete examples, it helps to try to pin down the difference between decidable and semidecidable more carefully.

We claim that there is a trivially decidable relation $T(e,x,t)$ and a simple decoding function D such that

$$\{e\}(x)\downarrow \iff \exists t\; T(e,x,t)$$

$$\{e\}(x) \simeq D(\min\big(\,t \mid T(e,x,t)\,\big))$$

Think of the t in $T(e,x,t)$ as the number of steps in the computation (given e,x,t we can easily construct the actual computation).

D then simply extracts the output of the computation.

Unboundable Search

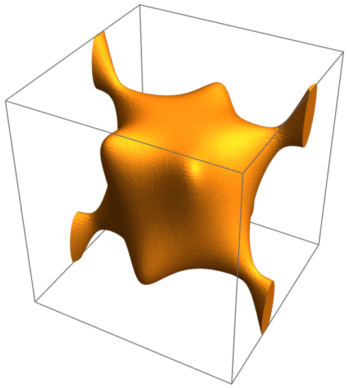
Again, $T(e, x, t)$ is trivial to compute.

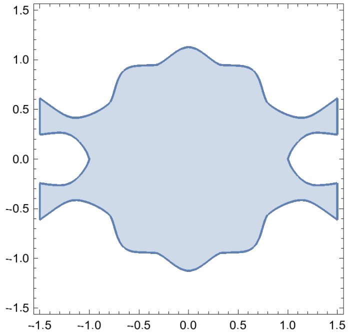
The problem is that we do not know how large t has to be to produce convergence: we **cannot** compute a bound on t from e and x. If we could compute a bound $\beta(x)$ ahead of time, everything is easy: just compute $T(e, x, \beta(x))$ and check; that's our decision algorithm for W_e.

But without a bound we are reduced to an unbounded search, and we are stuck with semidecidability.

Clever Idea:
Think of the unbounded search as a projection.

Geometric Projection





Projections

Definition

Suppose $R \subseteq \mathbb{N}^{n+1}$. The projection $S \subseteq \mathbb{N}^n$ of R is defined by

$$S(\boldsymbol{x}) \;:\Leftrightarrow\; \exists\, z\, R(z, \boldsymbol{x})$$

This is just the formal version of unbounded search:

```
input $\boldsymbol{x}$
foreach $w \in \mathbb{N}$ do
    if $(w, \boldsymbol{x}) \in R$
    then return YES
return NO
```

Note, though, that if the answer is NO, then this takes ω steps.

Who Cares?

In other words: the projection S of R is semidecidable whenever R is decidable: we can search for a witness z.

Much more is true: all semidecidable sets arise in this manner.

Lemma

Every semidecidable set is a projection of a decidable set.

This follows directly from Kleene normal form.

Logic and Geometry

Logically, the step from decidable to semidecidable corresponds to unbounded search, or existential quantification.

The step from semidecidable to co-semidecidable is negation.

These operations are logical, but we can also think of them as geometric: negation corresponds to taking complements, and search/existential quantification corresponds to projections.

Bringing geometry into the mix can be hugely helpful.

Piling On

Wild Idea: What if we close decidable sets under both projections and complements?

Well, we clearly get semidecidable and co-semidecidable ones.

But there is more: we also get projections of co-semidecidable sets, their complements, projections thereof, and so on and so on.

Of course, they might all turn out to be the same, but we will see they're all different.

Projections and Complements

Let's write down careful definitions for these purely set-theoretic operations (which, on the face of it, might have nothing to do with computability).

Let $A \subseteq \mathbb{N}^n$ where $n \geq 1$. We write

$$\text{proj}(A) = \{\, \boldsymbol{x} \in \mathbb{N}^{n-1} \mid \exists\, z\, (z, \boldsymbol{x}) \in A \,\} \subseteq \mathbb{N}^{n-1}.$$

for projections, and $\text{compl}(A)$ or $\overline{A}$ for the complement $\mathbb{N}^n - A$.

For any collection $\mathcal{C} \subseteq \mathfrak{P}(\mathbb{N}^n)$ of subsets of $\mathbb{N}^n$, $n \geq 1$, define $\text{proj}(\mathcal{C})$ to be the collection of all projections of sets in $\mathcal{C}$. Likewise, define $\text{compl}(\mathcal{C})$ to be the collection of all complements of sets in $\mathcal{C}$.

Arithmetical Hierarchy

Definition

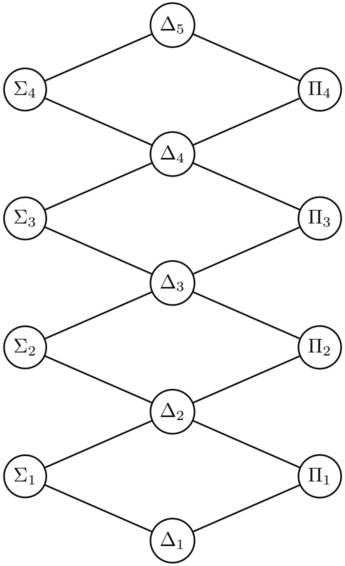
Define classes of subsets of $\mathbb{N}^n$:

$$\begin{aligned} \Sigma_0 = \Pi_0 &= \text{ all decidable sets} \\ \Sigma_{k+1} &= \mathsf{proj}(\Pi_k) \\ \Pi_k &= \mathsf{compl}(\Sigma_k) \\ \Delta_k &= \Sigma_k \cap \Pi_k \end{aligned}$$

Thus, $\Delta_0 = \Delta_1$ is the class of all decidable sets, Σ_1 is the class of all semidecidable sets, and Π_1 is the class of all co-semidecidable sets.

This is also known as the Kleene-Mostowski hierarchy.

A Diagram


Δ_5
Σ_4
Π_4
Δ_4
Σ_3
Π_3
Δ_3
Σ_2
Π_2
Δ_2
Σ_1
Π_1
Δ_1

Arithmetical Sets

Definition

A set $A \subseteq \mathbb{N}^n$ is arithmetical if it belongs to some class Σ_k.

It follows by the usual cardinality mumbo-jumbo that the class of arithmetical sets is countable, so we are missing almost all subsets of $\mathbb{N}$.

One very important example of a set we are missing is arithmetical truth: the set of all (code numbers of) formulas of arithmetic that are true is not an element in this hierarchy. To capture this set, one needs another hierarchy, the analytical hierarchy. There, arithmetic truth appears at level Δ^1_1, just outside of our arithmetical sets.

The Bottom of AH

Of practical relevance are mostly the first few levels of the arithmetical hierarchy. We have already seen examples of decidable, semidecidable sets and co-semidecidable sets.

Here are some other examples.

- TOT is Π_2.
- The indices of all finite r.e. sets form a Σ_2 set:

$$\text{FIN} = \{ e \mid W_e \text{ finite} \}$$

- The indices of all cofinite r.e. sets form a Σ_3 set:

$$\text{Cof} = \{ e \mid W_e \text{ cofinite} \}$$

None of these sets belong to the next lower Δ_k level of the hierarchy.

Heuristic

To say that $A \subseteq \mathbb{N}$ is Π_2 comes down to this: there is a decidable property P such that

$$\begin{aligned} x \in A &\iff \forall\, u\, \exists\, v\ P(u,v,x) \\ &\iff \neg\exists\, u\, \neg\exists\, v\ P(u,v,x) \end{aligned}$$

corresponding to a sequence project–complement–project–complement.

Writing down the set in terms of alternating quantifiers usually produces the right spot in the arithmetical hierarchy.

Decidable Sets

Consider the index set of all decidable (recursive) sets:

$$\mathsf{REC} = \{\, e \mid W_e \text{ decidable} \,\}$$

To locate REC in the arithmetical hierarchy note

$$\begin{aligned} e \in \mathsf{REC} &\Leftrightarrow \exists\, e' \,(W_e \cap W_{e'} = \emptyset \wedge W_e \cup W_{e'} = \mathbb{N}) \\ &\Leftrightarrow \exists\, e' \,(\forall\, \sigma\,(W_{e,\sigma} \cap W_{e',\sigma} = \emptyset) \wedge \forall\, x\, \exists\, \tau\,(x \in W_{e,\tau} \cup W_{e',\tau})) \\ &\Leftrightarrow \exists\, e' \,\forall\, x, \sigma\, \exists\, \tau\,(W_{e,\sigma} \cap W_{e',\sigma} = \emptyset \wedge x \in W_{e,\tau} \cup W_{e',\tau}) \end{aligned}$$

So, REC $\in \Sigma_3$. As it turns out, REC $\notin \Delta_3$.

The Big Question

At the bottom level of the hierarchy, it certainly seems that higher levels are strictly larger:

$$\Sigma_0 \subsetneq \Sigma_1 \subsetneq \Sigma_2 \subsetneq \Sigma_3 \subsetneq \dots$$

Except for the first inclusion, we don't know for sure.

In principle, it could happen that $\Sigma_{42} = \Sigma_{43}$. We need a proof that the hierarchy does not collapse.

Emil Post

Theorem (Post's Hierarchy Theorem)

All the inclusions $\Delta_k \subsetneqq \Sigma_k, \Pi_k \subsetneqq \Delta_{k+1}$ *are proper,* $k \geq 1$.

We will encounter a similar hierarchy in complexity theory (just add a polynomial time bound everywhere), but there it is not known whether the hierarchy is proper.

We will skip the proof (there are notes on our website if you are interested).

Exercises

Exercise

Show that FIN *is* Σ_2 *by constructing this set using projections and complementation.*

Exercise

Show that Cof *is* Σ_3 *by constructing this set using projections and complementation.*

Exercise

Find the position in the hierarchy of

$$\mathsf{TOT} = \{\, e \mid W_e = \mathbb{N} \,\}$$