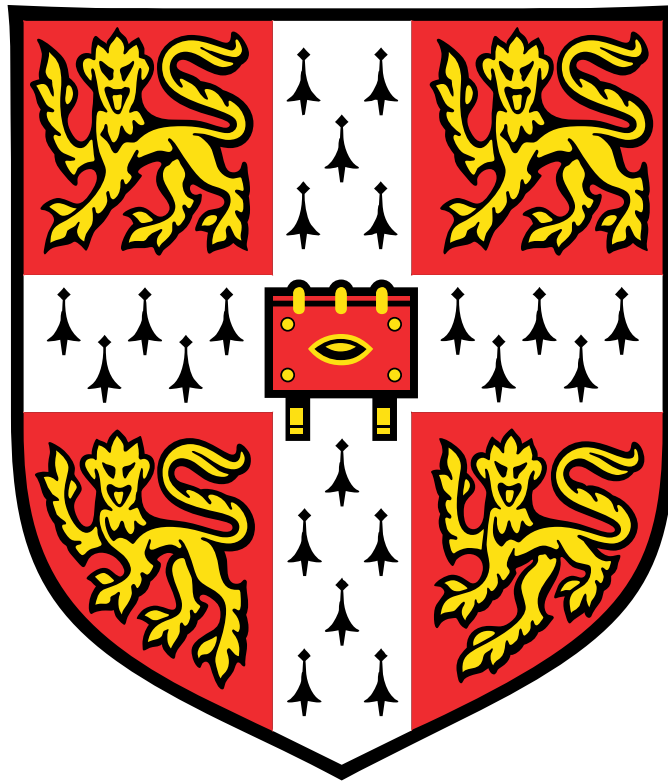


Part III Computability and Logic:
24 Lectures in Michaelmas Term 2015

Thomas Forster

October 22, 2015



Contents

| | | |
|----------|--|-----------|
| 1 | Preface: Some Useful Reflections | 9 |
| 1.1 | What are the objects of computation? A conversation on FOM . | 9 |
| 2 | Introduction and some History | 11 |
| 2.1 | Definitions | 14 |
| 3 | Recursive Datatypes | 15 |
| 3.1 | Wellfounded Induction | 15 |
| 3.2 | Inductively Defined Sets | 18 |
| 3.3 | Horn Clauses and the Uniqueness Problem | 19 |
| 3.4 | Structural Induction | 21 |
| 3.5 | Engendering Relations | 22 |
| 3.6 | Rectypes and Least Fixed Points | 24 |
| 3.6.1 | Fixed Point Theorems | 24 |
| 3.6.2 | Rectypes as least fixed points | 27 |
| 3.7 | Finite vs Bounded vs Unbounded Character | 27 |
| 3.7.1 | Rectypes of Unbounded Character are Paradoxical | 28 |
| 3.7.2 | Bounded Character | 31 |
| 3.8 | Ordinals | 32 |
| 3.8.1 | Rank functions | 32 |
| 3.9 | Restricted Quantifiers | 32 |
| 3.10 | Infinitary Languages | 35 |
| 3.10.1 | “Wellfounded” is Infinitary Horn | 36 |
| 3.10.2 | Some Remarks on Infinitary Languages | 36 |
| 3.11 | Greatest fixed points (“Co-rectypes”) | 37 |
| 3.12 | Certificates | 37 |
| 3.12.1 | Free vs non-free rectypes (ambiguous parses) | 38 |
| 3.12.2 | A Last Crazy Thought | 38 |
| 4 | Functions | 41 |
| 4.1 | Primitive Recursion | 41 |
| 4.1.1 | Some quite nasty functions are primitive recursive | 45 |
| 4.1.2 | Justifying Circular Definitions | 45 |
| 4.2 | Exercises | 51 |

| | | |
|----------|--|------------|
| 4.2.1 | Primitive Recursive Relations | 51 |
| 4.2.2 | Simultaneous Recursion | 55 |
| 4.3 | μ -recursion | 56 |
| 4.3.1 | The Ackermann function | 56 |
| 5 | Machines | 63 |
| 5.1 | Finite State Machines | 63 |
| 5.1.1 | Kleene's theorem | 64 |
| 5.1.2 | The Thought-experiment and Myhill-Nerode | 66 |
| 5.1.3 | Nondeterministic Machines | 67 |
| 5.1.4 | Exercises | 70 |
| 5.2 | Machines with infinitely many states | 73 |
| 5.3 | μ -recursive = register machine-computable | 76 |
| 5.3.1 | A Universal Register Machine | 78 |
| 5.4 | Decidable and Semidecidable Sets | 78 |
| 5.4.1 | Zigzagging Autoparallelism: Volcanoes | 78 |
| 5.4.2 | Decidable and Semidecidable Sets | 80 |
| 5.4.3 | A Nice Illustration and a Digression | 84 |
| 5.4.4 | "In finite time"—a warning | 85 |
| 5.5 | Decidable and semidecidable sets of other things | 86 |
| 5.5.1 | Applications to Logic | 87 |
| 5.6 | The Undecidability of the Halting Problem | 90 |
| 5.6.1 | Rice's Theorem | 91 |
| 5.7 | Recursive Inseparability | 94 |
| 5.8 | Exercises | 95 |
| 6 | Representability by λ-terms | 101 |
| 6.1 | Some λ -calculus | 101 |
| 6.2 | Arithmetic with Church Numerals | 101 |
| 6.3 | Representing the μ operator in λ -calculus | 105 |
| 6.4 | Typed Lambda terms for computable functions | 107 |
| 6.4.1 | Combinators?? | 108 |
| 7 | Recursive and Automatic Structures | 109 |
| 7.1 | Automatic Structures | 109 |
| 7.1.1 | Automatic ordinals | 111 |
| 7.1.2 | Automatic theories | 111 |
| 7.2 | Recursive structures | 112 |
| 7.3 | Tennenbaum's Theorem | 114 |
| 7.4 | Recursive Saturation | 117 |
| 8 | Incompleteness | 119 |
| 8.1 | Proofs of Totality | 119 |
| 8.2 | A Theorem of Gödel's | 120 |
| 8.2.1 | The T -bad function | 120 |
| 8.3 | Undecidability of Predicate Calculus | 122 |

| | | |
|-----------|--|------------|
| 8.4 | Trakhtenbrot's theorem | 123 |
| 8.5 | Refinements of theorem 19 | 125 |
| 9 | WQO theory | 127 |
| 9.1 | WQOs | 128 |
| 9.1.1 | The Minimal Bad Sequence construction | 130 |
| 9.2 | Kruskal's theorem | 131 |
| 9.3 | Some <i>bonnes bouches</i> | 132 |
| 9.3.1 | How to get some large ordinals | 132 |
| 9.3.2 | Friedman's Finite Form of Kruskal's Theorem | 133 |
| 10 | Elementary Degree Theory | 135 |
| 10.1 | Computation relative to an oracle. | 136 |
| 10.2 | Priority Methods | 138 |
| 10.2.1 | Friedberg-Muchnik | 139 |
| 10.2.2 | Omitting Types | 142 |
| 10.2.3 | Baker-Gill-Solovay and $\mathcal{L}P = NP?$ | 143 |
| 11 | Proofs and Ordinals | 147 |
| 11.1 | The Ordinal ϵ_0 and the Consistency of Peano Arithmetic | 147 |
| 11.2 | The Goodstein function | 149 |
| 11.3 | Hierarchies of fast-growing functions | 156 |
| 11.3.1 | Good behaviour of the F_α , and the Schmidt conditions | 159 |
| 11.3.2 | Schmidt-coherence | 160 |
| 11.4 | Preposterously Large Countable Ordinals | 165 |
| 11.4.1 | Cantor normal form using $\omega \uparrow \uparrow \alpha$ | 169 |
| 12 | Constructive Mathematics | 171 |
| 12.1 | Diaconescu: the Axiom of Choice implies Excluded Middle | 173 |
| 12.1.1 | Least Number Principle Implies Excluded Middle | 174 |
| 12.1.2 | Linton-Johnstone and Markov's Principle | 175 |
| 12.2 | Proof theory, Curry-Howard and Realizability | 175 |
| 12.2.1 | Proof Theory | 175 |
| 12.2.2 | Curry-Howard | 176 |
| 12.2.3 | The Axiom of Choice in Constructive Set Theory: notes consequent to a lecture given by Sol Feferman on 10/i/2014, written up partly to amuse Valeria Paiva | 177 |
| 12.3 | Recursive Analysis | 179 |
| 12.4 | A constructive treatment of infinitesimals | 180 |
| 13 | Notes and Appendices | 181 |
| 13.1 | Chapter 3 | 181 |
| 13.1.1 | Horn clauses in rectype declarations | 181 |
| 13.1.2 | Infinitary Languages | 182 |
| 13.2 | Chapter 4 | 184 |
| 13.2.1 | A bit of pedantry | 184 |

| | |
|---|------------|
| 13.2.2 The Ackermann function | 185 |
| 13.3 Chapter 5 | 187 |
| 13.4 Chapter 6 | 187 |
| 13.5 Chapter 7 | 187 |
| 13.6 Chapter 10 | 187 |
| 13.7 Chapter 11 | 187 |
| 14 Answers to selected questions | 195 |

Health Warning!

These are the notes from which I am going to lecture this course; they are not a course handbook or anything of that nature. Since they are the notes from which I lecture, they naturally contain—to a first approximation—only those materials that I can *not* lecture off the top of my head. The things they have *in extenso* are (i) things whose details I am liable to forget, and (ii) things which at some point or other I found it a useful discipline to set out properly; (iii) things, such as definitions, that I don't see any point in putting up on the board but which can be supplied in handouts. Finally (iv) there are messages to myself about things that might turn into lecturable material but which have not yet done so, and unwary readers may be disconcerted by them. However, one respect in which these notes do resemble a course handout is in the presence of exercises. Some care has gone into their selection and placement and students can with profit attempt them all. I am going to try to ensure that those that have model answers—or at least moderately helpful discussions—(to be revealed later) will be marked with an asterisk. The usual inducements are available for nice L^AT_EX source code for answers to unasterisked exercises. [No fancy macros please!] Apologies for pillaging Part II example sheets from Professors Hyland and Johnstone. We start with a very easy exercise.

EXERCISE 1 *Spell your own name; what is your favourite colour?*

These notes will read oddly to some. They are designed for people who came to Part III Logic *via* Part II of the Cambridge Mathematics Tripos. Such people are strong mathematicians (so it is safe to set a cracking pace) but they are strong mathematicians who have never been lectured any Computation Theory (and not much logic either) so we start from a low base. If you come from outside Cambridge you may well find that you know a lot of this stuff already.

As always I am indebted to students for finding mistakes, infelicities and oversights in earlier versions of these notes (and for not roasting me for them)—and for supplying some model answers. Thanks are due (so far!) Jack Webster, Tim Talbot, Zhen Low, Zach Norwood, John Lapinskas, Paul Slevin, Philipp Kleppmann, Ben Millwood, Lovkush Agarwal, Jane Aston, Jonathan Lee, Shoham Letzter, Matvey Soloviev, Kris Cao, Ewan Davies, Jonathan Holmes, Maria Gorinova, Ramana Kumar, Auke Booij, Jason Long, Henk-Jaap Wagenaar and Patrick Stevens. The list grows yearly!

I am hugely indebted to various colleagues both for general guidance and specific help on particular theorems . . . Nathan Bowler, Adrian Mathias, James Cummings, Ashley Montanaro, Peter Smith, Zachiri McKenzie, Ian Grant, Bob Geroch and—most of all—Stanley Wainer who has been good enough to provide a guest lecture.

Finally I am indebted to Bob Geroch for the thought (section 8.2.1) that one could use proofs of termination to obtain natural examples of sentences unprovable in given systems of arithmetic.

Chapter 1

Preface: Some Useful Reflections

1.1 What are the objects of computation? A conversation on FOM

Rex Butler writes:

It is commonly pointed out, in textbooks and elsewhere, that we don't lose much by focusing our study of computability to only the set of natural numbers.

Herb Enderton writes:

Yes and no. I claim that the correct objects of study are not **numbers**, but **numerals**. (Remember the short-lived “New Math”? I have an old elementary-school workbook where on one page the instructions read, “Circle the correct numeral.”) That is, in computability theory one examines procedures that, *given* x will *return* $f(x)$. One cannot be given a number, but only a numeral. Numerals are strings of symbols—little pieces of language. As such, numerals can be communicated (to your computer, from your computer). Numbers cannot.

Arnon Avron writes:

I don't agree. We all know what addition is, but addition is done completely differently when one is using decimal representations of numbers, binary ones, or a representation as a list of strokes. If computation is done only with numerals or strings of symbols, then I don't see how completely different functions, applied to the same numerals (say $111 + 11$) can all be referred to as “addition” (by the way: “string of symbols” is also just an abstract object—a symbol

written by me with a pen using my “beautiful” handwriting is not really identical to the “same” symbol typed by machine. Moreover: what is a string? What should be the maximal distance between two symbols in a “string” that we still call it a “string”?).

I maintain that all real computations and construction are done with abstract objects (not *too* abstract—but this is another issue). Already the solutions of construction problems in Geometry were algorithms for *abstract* lines and points, using abstract ruler and compass, not anything done by actual ruler and compass!

Herb Enderton:

Yes, descriptions are not unique. Conclusion: Computing on graphs is no different from computing on \mathbb{N} , except that there is a hidden equivalence relation that gets swept under the rug by the adoption of a coding. And under the rug is the right place for it!

Perhaps here is hidden your possible answer to my argument: there are equivalence relations involved. But “an equivalence relation” is a more abstract notion than the notion of a natural number. Moreover: what is the equivalence relation when it comes to different systems of numerals (except “representing the same number”)?

Rex Butler:

For example, suppose I want to choose a coding method for, say, graphs with rationally weighted edges . . .

Herb Enderton:

Here ‘coding’ is a loaded term. What you want is to *communicate* a graph to the computational procedure. So you describe exactly the graph you have in mind—and that description is nothing but a word over a finite alphabet of k letters. (Maybe k is the number of keys on your keyboard.) And the quality of the description matters—a poor description might not work.

And here is the connection: There is no difference between the set of words over a k -letter alphabet and the set of numerals in k -adic notation. (By k -adic notation, I mean standard base- k notation, but with the digits $\{1, 2, \dots, k\}$, without a zero digit.)

Chapter 2

Introduction and some History

Start with Hilbert and diophantine equations. The key notion behind computability is the slangy informal notion of *finite object*.

There is a surprisingly illuminating history to be found in [25].

To apply the theorems and insights of computation theory widely in mathematics we need the notion of a *finite object* or (perhaps better put): *object of finite character*. Classic contrastive explanation: rationals are finite objects but reals are not. (“Finite precision” vs “Infinite precision”). This matters to us because any sensible concept of algorithm that we come up with is going to be one that can cope with only a finite amount of information at any one time: its inputs must be things that are finite-objects in the sense we are trying to get straight.

Roughly, a finite object is an object that has a finite description in a countable language (and a countable language that has a finite description, at that). Such objects may well be infinite in some other sense. The graph of an polynomial function $\mathbb{R} \rightarrow \mathbb{R}$ certainly contains infinitely many points but it can still be given a finite description and is a finite object in our sense (at least if it has coefficients in \mathbb{Z}). (The function-in-extension is literally infinite, thought of as a set). Any countable language can be gnumbered¹ and we can contemplate which of the manipulations of the finite objects it characterises correspond to the computable manipulations of the gnumbers of the expressions of the language.

The concept of finite object has applications outside pure mathematics. Vertebrates have skeletons made from bones that (from the point of view of the animal) are rigid; movement only occurs at [a small finite number of] joints. Thus the human arm has only finitely many degrees of freedom so controlling its movements is a tractable problem: we have a motor cortex! The octopus tentacle has no skeleton and has infinitely many degrees of freedom, so controlling its movement is an intractable problem. There is no detailed central control of movements of the octopus tentacle:

¹Gödel-numbering; the ‘g’ is silent.

movements are controlled *locally* by a network of ganglia, one for each sucker. The octopus brain does not know the configuration of the tentacles.

Word problems for groups. Group presentations. Note that (old hands might have encountered this in Part II with Prof Leader's question on Sheet 3 q 8 part (vi) about the theory of simple groups of order 60) a presentation of a group does not straightforwardly give rise to a categorical first-order theory that characterises it. You cannot compute the first-order theory of a group from a presentation of it. Burnside groups?

update this reference

r-process and *s*-process: an example from Physics

Physicists who study nucleosynthesis distinguish between *s*-process nuclei and *r*-process nuclei (the '*s*' and the '*r*' connoting slow and rapid respectively). One can think of the two sets of *s*-process and *r*-process nuclei as inductively defined sets as follows.²

You are an *s*-process nucleus if you have very long (or infinite) half-life and are the result of a neutron capture by an *s*-process nucleus or the result of a β -decay of a result of neutron capture by an *s*-process nucleus;

You are an *r*-process nucleus if you are stable to neutron-drip and you are either (i) the result of a neutron capture by an *r*-process nucleus or (ii) the result of a β -decay of an *r*-process nucleus.

at some point have to explain 'acceptable enumeration'. If we are to deal with computation out in the wide world as computing with numerals then we have to be sure that our encoding/gnumbering is under control.

But we'll start with finite+bounded (not least because it enables us to motivate the (otherwise rather odd) move of taking nondeterminism seriously).

Hilbert's 1900 address set a number of tasks whose successful completion would inevitably involve more formalisation. It seems fairly clear that this was deliberate: Hilbert certainly believed that if formalisation was pursued thoroughly and done properly, then all the contradictions that were crawling out of the woodwork at that time could be dealt with once and for all.

One of the tasks was to find a method for solving all diophantine equations. What does this mean exactly? Let us review the pythagorean equation $x^2 + y^2 = z^2$, to see what "solving" might mean. It is easy to check that, for any two integers a and b ,

$$(a^2 - b^2)^2 + (2ab)^2 = (a^2 + b^2)^2,$$

and so there are infinitely many integer solutions to $x^2 + y^2 = z^2$. Indeed we can even show that every solution to the pythagorean equation (at least every solution where x , y and z have no common factor) arises in this way:

Notice that if $x^2 + y^2 = z^2$, then z is odd and precisely one of x and y is even. (We are assuming no common factors!) Let us take y to be the even one and x the odd one.

Evidently $x^2 = (z - y)(z + y)$, and let d be the highest common factor of $z - y$ and $z + y$. Then there are coprime a and b satisfying $z + y = ad$ and

²One could make a type-token point here: i think physicists sometimes refer to these nucleus-types as *species*.

$z - y = bd$. So $x^2 = abd^2$. This can happen only if a and b are perfect squares, say u^2 and v^2 , respectively. So $x = uvd$.

This gives us $z = \frac{u^2+v^2}{2} \cdot d$ and $y = \frac{u^2-v^2}{2} \cdot d$ and in fact d turns out to be 2.

Thus we have a complete description of all solutions (in integers) to the pythagorean equation.

Hilbert's question—and it is a natural one—was: can we clean up all diophantine equations in the way we have just cleaned up this one?

If there is a general method for solving diophantine equations, then we have the possibility of finding it. If we find it, we exhibit it, and we are done. To be slightly more specific, we have a proof that says “Let E be a diophantine equation, then ...”, using the rule of universal generalisation (UG).

On the other hand, if there is no such general method, what are we to do? We would have to be able to say something like: let \mathfrak{A} be an arbitrary algorithm; then we will show that there is a diophantine equation that \mathfrak{A} does not solve. But clearly, in order to do this, we must have a formal concept of an algorithm. Hilbert's challenge was to find one.

There are various formal versions of computation. We shall see how the set of strings recognised by a finite state machine gives rise to a concept of computable set. However, we will also see a fatal drawback to any analysis of computable set in terms of finite-state machines: the matching bracket language is not recognised by any finite-state machine but is obviously computable in some sense. The problem arises because each finite-state machine has a number of states (or amount of memory, to put it another way) that is fixed permanently in advance. Despite this, the concept of computability by finite-state-machines turns out to be mathematically interesting and nontrivial. However it is not what we are primarily after. The most general kind of computation that we can imagine that we would consider to be computation is deterministic, finite in time and memory but unbounded: no predetermined limit on the amount of time or memory used. There have been various attempts to capture this idea in machinery rigorous enough for one to prove facts about it. The (historically) first of the most general versions is Turing machines. There is also representability by λ -terms which we will see in chapter 6.

Another attempt is μ -recursion, which we will do in detail below. The other approach we explore in some detail is an analysis in terms of register machines. They do not have the historical *cachet* of Turing machines but are slightly easier to exploit, since they look more like modern computers.

What became clear about 70 years ago is that all attempts to formalise the maximal idea of a computable function result in the same class of functions. This gives rise to **Church's thesis**. Although not normally presented as such, Church's thesis is really just a claim that this endeavour to illuminate—by formalisation—our intuitive idea of a computable function has now been completed: we will never need another notion of “computable”.³

³Philosophically inclined readers may wish to reflect on the curious fact that Church's

How can we be so confident? Well, we have a completeness theorem. All completeness theorems have two legs: a semantic concept and a syntactic concept. In Part II you saw an elementary and pleasing example of a completeness theorem: the completeness of propositional logic. It states that two classes of formulæ are one and the same class. (i) The set of truth-table tautologies; and (ii) the class of formulæ deducible from axioms K, S and T by means of substitution and *modus ponens*. (ii) is a syntactic concept, and (i) is a semantic concept.



The semantic concept in the computability case is Turing-computable or register machine-computable. The syntactic concept is a bit harder. The first attempt at it is **primitive recursive**; we will discover the correct syntactical concept by examining what goes wrong with primitive recursive functions.

But before that we have to get all our definitions out of the way.

2.1 Definitions

Intension-extension (talk over this); graph of a function. We will use lambda notation.

‘■’ signifies the end of a proof.

We use the pig classification for cuteness of theorems. The more occurrences of ‘’ the tastier the theorem; the more occurrences of ‘’ the nastier the construction.

\mathbb{N} is the set of natural numbers and \aleph_0 is its cardinality, \mathbb{Z} is the set of integers, \mathbb{Q} is the set of rationals, and ω is the first infinite ordinal.

Cardinality: ‘ $|X|$ ’ denotes the cardinal of the set X .

$[X]^n$ is $\{x \subseteq X : |x| = n\}$.

we write ordered pairs as $\langle x, y \rangle$

range of a function: $f''\mathbb{N} := \{f(n) : n \in \mathbb{N}\}$

$\text{Dom}(f) := \{n \in \mathbb{N} : f(n) \downarrow\}$ (see p. 75)

:: for consing—both ways round, for both **cons** and **snoc**

$X \rightarrow Y$ is the set of partial functions from X to Y

thesis is a metamathematical allegation of which no formal proof can be given. As soon as we formalise “All attempts to formalise our informal notion of finite-but-unbounded computation result in the same formal notion”, the reference to informal computation becomes a reference to a formal notion and the sense is lost. You may wish to google ‘Church’s translation argument’ in this connection.

Chapter 3

Induction, Wellfoundedness and Recursion in a General Context

Induction can only be understood backwards, but it must be lived forwards.

Kierkegaard

3.1 Wellfounded Induction

Suppose we have a carrier set with a binary relation R on it, and we want to be able to infer

$$\forall x \psi(x)$$

from

$$(\forall x)((\forall y)(R(y, x) \rightarrow \psi(y)) \rightarrow \psi(x))$$

In words, we want to be able to infer that everything is ψ from the news that you are ψ as long as all your R -predecessors are ψ . **y is an R -predecessor of x** if $R(y, x)$. Notice that there is no “case $n = 0$ ” clause in this more general form of induction: the premiss we are going to use implies immediately that a thing with no R -predecessors must have ψ . The expression “ $(\forall y)(R(y, x) \rightarrow \psi(y))$ ” is called the **induction hypothesis**. The first line says that if the induction hypothesis is satisfied, then x is ψ too. Finally, the inference we are trying to draw is this: **if** x has ψ whenever the induction hypothesis is satisfied, **then** everything has ψ . When can we do this? We must try to identify some condition on R that is equivalent to the assertion that this is a legitimate inference to draw in general (i.e., for any predicate ψ).

Why should anyone want to draw such an inference? The antecedent says “ x is ψ as long as all the immediate R -predecessors of x are ψ ”, and there are

plenty of situations where we wish to be able to argue in this way. Take $R(x, y)$ to be “ x is a parent of y ”, and then the inference from “children of blue-eyed parents have blue eyes” to “everyone has blue eyes” is an instance of the rule schematised above. As it happens, this is a case where the relation R in question does *not* satisfy the necessary condition, for it is in fact the case that children of blue-eyed parents have blue eyes and yet not everyone is blue-eyed.

To find what the magic ingredient is, let us fix the relation R that we are interested in and suppose that the inference

$$\frac{(\forall y)(R(y, x) \rightarrow \psi(y)) \rightarrow \psi(x)}{(\forall x)(\psi(x))} \quad R\text{-induction}$$

has failed for some choice ψ of predicate. Then we will see what this tells us about R . To say that R is well-founded all we have to do is stipulate that this failure (whatever it is) cannot happen for any choice of ψ .

Let ψ be some predicate for which the inference fails.

Then the top line is true and the bottom line is false. So $\{x : \neg\psi(x)\}$ is nonempty. Let us call this set A for short. Using the top line, let x be something with no R -predecessors. Then all R -predecessors of x are ψ (vacuously!) and therefore x is ψ too. This tells us that if y is something that is not ψ , *then there must be some y' such that $R(y', y)$ and y' is not ψ either*. If there were not, y would be ψ . This tells us that the collection A of things that are not ψ “has no R -least member” in the sense that everything in that collection has an R -predecessor in that collection. That is to say

$$(\forall x \in A)(\exists y \in A)(R(y, x))$$

To ensure that R -induction can be trusted it will suffice to impose on R the condition that $(\forall x \in A)(\exists y \in A)(R(y, x))$ never hold, for any nonempty $A \subseteq \text{dom}(R)$. Accordingly, we will attach great importance to the following condition on R :

DEFINITION 1 *R is well-founded iff for every nonempty subset A of $\text{dom}(R)$ we have $(\exists x \in A)(\forall y \in A)(\neg R(y, x))$
(x is an “ R -minimal” element of A .)*

This definition comes with a health warning: it is easy to misremember. The only reliable way to remember it correctly is to rerun in your mind the discussion we have gone through: well-foundedness is precisely the magic property one needs a relation R to have if one is to be able to do induction over R . No more and no less. The definition is not *memorable*, but it is *reconstructible*.

Wellfoundedness is a very important idea to be found all over Mathematics, even in places where the word is not used. Noetherian rings are rings with a certain wellfoundedness property. Hilbert’s basis Theorem is the news that certain constructions preserve wellfoundedness

You may be more familiar with a definition talking about “no infinite descending chains”. These two definitions are not equivalent without DC, the **Axiom of Dependent Choices**:

$$(\forall x \in X)(\exists y \in X)(R(x, y)) \rightarrow (\forall x \in X)(\exists f : \mathbb{N} \rightarrow X)(f(0) = x \wedge (\forall n)(R(f(n), f(n+1))))$$

If DC fails we can let X be an infinite Dedekind-finite set (not the same size as any of its proper subsets) and consider the tree of wellorderings of subsets of X ordered by reverse end extension (so longer wellorderings come lower in the tree). This tree is not wellfounded (it has subsets—such as the tree itself—with no minimal elements) but has no infinite descending chain.

REMARK 1 Suppose $R' \subseteq R$ are wellfounded relations on a fixed domain. Then R' -induction is a weaker principle than R -induction.

Proof:

If $R'(x, y) \rightarrow R(x, y)$ then $(\forall y)(R(y, x) \rightarrow \phi(y))$ implies $(\forall y)(R'(y, x) \rightarrow \phi(y))$ and $(\forall x)((\forall y)(R'(y, x) \rightarrow \phi(y)) \rightarrow \phi(x))$ implies $(\forall x)((\forall y)(R(y, x) \rightarrow \phi(y)) \rightarrow \phi(x))$ and finally

$$(\forall x)((\forall y)(R(y, x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow (\forall z)(\phi(z))$$

implies

$$(\forall x)((\forall y)(R'(y, x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow (\forall z)(\phi(z))$$

■

Reflect that if R' is the empty relation then R -induction is trivial. For consider: if R is the empty relation then

$$(\forall x)((\forall y)(R(y, x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow (\forall z)(\phi(z))$$

is

$$(\forall x)((\forall y)(\perp \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow (\forall z)(\phi(z))$$

which is

$$(\forall x)((\forall y)(\top \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow (\forall z)(\phi(z))$$

which is

$$(\forall x)(\phi(x)) \rightarrow (\forall z)(\phi(z)).$$

EXERCISE 2 ^(*)¹ Provide a sequent calculus or natural deduction proof that $(\forall x)((\forall y)(R(y, x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow (\forall z)(\phi(z))$ and $(\forall xy)(R'(x, y) \rightarrow R(x, y))$ together imply

$$(\forall x)((\forall y)(R'(y, x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow (\forall z)(\phi(z)).$$

If $(\forall y)(\neg R(y, x))$ then we say x is a **zero element**.

EXERCISE 3 Let $\langle A, R \rangle$ and $\langle B, S \rangle$ be wellfounded binary structures.

¹Yes, i know, you haven't been lectured natural deduction or sequent calculus at this stage. This is for revision

(i) Show that the pointwise product is also a wellfounded binary structure.

Define a relation T on $A \rightarrow B$ by $T(f, g)$ iff $(\forall a, a' \in A)(R(a, a') \rightarrow S(f(a), g(a')))$.

(ii) Give an example to show that T need not be wellfounded even if R and S are. (iii) Show that in contrast the restriction of T to those elements of $A \rightarrow B$ that take only finitely many nonzero values is wellfounded.

THEOREM 1 *The Recursion Theorem*

If $\langle X, R \rangle$ is a wellfounded structure and $G : X \times \mathcal{P}(V) \rightarrow V$ then there is a unique f satisfying

$$(\forall x \in X)(f(x) = G(x, \{f(y) : R(y, x)\}))$$

[Aside: In earlier versions i had $G : X \times V \rightarrow V$, and various people picked me up on it. The point is that you want G to be able to cope with [any ordered pair whose first component is in X] and whose second component is a subset of the range of f . Such a thing is at any rate a set, and so is in $\mathcal{P}(V)$, that being the collection of all sets. If you are in the world of sets V and $\mathcal{P}(V)$ are the same thing, so there doesn't seem to be much point in distinguishing between them—particularly if it makes the statement of the theorem longer. But yes, $\mathcal{P}(V)$, is better than V]

EXERCISE 4 *Prove theorem 1*

The proof is entirely straightforward once wellfoundedness is understood. You need the concept of an *attempt*, and you prove by induction that every element of the domain of R is in the domain of some attempt. You also show that any two attempts agree on their intersection. Then you form the union of all attempts. ■

3.2 Inductively defined sets aka Recursive Datatypes aka Rectypes

Rectypes: have **founders** and **constructors**.

Examples of rectypes:

The empty set is Kuratowski-finite (“Kfinite”); if X is Kfinite then $X \cup \{y\}$ is Kuratowski-finite.

The empty set is N-finite; if X is N-finite and $y \notin X$ then $X \cup \{y\}$ is N-finite.

The empty set is hereditarily finite²; if x and y are hereditarily finite so is $x \cup \{y\}$.

Classically Kfinite and N-finite are the same; constructively they are not. In this context sets which are plain vanilla-finite (both K-finite and N-finite in the way you know and love) are said to be *inductively finite*:

²Set Theory and Logic 2012/3 sheet 3 q 6.

The empty set is inductively finite; if x is inductively finite so is $x \cup \{y\}$.

[put somewhere a proof that Kfinite sets closed under binary union and $\bigcup X$ is kfinite if X is a kfinite set of kfinite sets]

Further examples:

Natural numbers

Formulæ (Backus-Naur Form),

Proofs

The family of words in a group presentation

primitive recursive functions (later!)

lists, trees.

Some of you may remember exercises 6 and 10 on PTJ's Part II Set Theory and Logic sheet 3 2012/3. If you didn't do them then you should consider doing them now. (They are probably on the web somewhere) There is a discussion of them in the materials on my Part II Materials page.

All these rectypes are *of finite character*: the operations that construct them are finitary.

3.3 Horn Clauses and the Uniqueness Problem

An inductively defined set can always be thought of as The Least thing above X satisfying F and containing y . When can we do it? Sometimes obviously possible sometimes obviously impossible. Interesting cases in the middle: forcing and field extensions. There is an **Existence Problem** and a **Uniqueness Problem**: is there a minimal thing above X satisfying F and containing y ? If there is, is it unique? For example: if *thing* means *total order* then there is a minimal thing³, but it's not unique. There is a syntactic reason for this.

DEFINITION 2 .

A Horn clause is a disjunction of atomics and negatomics of which at most one is atomic.

A Horn property is a property captured by a [closure of a] Horn expression;

A Horn theory is a theory all of whose axioms are universal closures of (conjunctions of) Horn clauses.

If 'least' means 'least with respect to \subseteq ' then there is a nice logical theorem: it works iff F is *Horn*. Intersection-closed. ' $f''X^n \subseteq X$ ' is Horn. The easy

³That is the order extension principle, a consequence of Zorn's lemma.

direction i am leaving as an exercise; it will say: if F is a Horn property, then for every x the F -closure of x exists and is well-defined and unique.⁴

Observe that “is a total order” is not a Horn property.

The reason for the appearance of Horn clauses here is that a rectype declaration is always a pile of Horn sentences. For example, we declare the natural numbers by

$$\mathbf{N}(0); (\forall x)(\mathbf{N}(x) \rightarrow \mathbf{N}(S(x)))$$

We declare the datatype of α -lists by

$$\alpha\text{-list}(\mathbf{null}); (\forall l)((\alpha\text{-list}(l) \wedge \alpha(x)) \rightarrow \alpha\text{-list}(\mathbf{cons}(x, l)))$$

The class of models of a Horn theory is closed under various constructions, e.g. substructure, direct limits. This is a cute fact that you should remember (and prove, too—it’s not difficult), but we won’t make any use of it in this course⁵.

Symmetric, irreflexive, antisymmetric, transitive, reflexive are Horn properties. If F is a Horn property [of relations] then we have the notion of the F -closure of a relation. The [graph of] the F -closure of a (binary) relation-[in-extension] R is $\bigcap\{S \supseteq R : F(S)\}$.

The significance of these ideas for us here is that the [graph of the] F -closure of a relation is a rectype. For example, the transitive closure of a relation—thought of as a set of ordered pairs—is closed under a certain binary operation on pairs. The assertion that a set of ordered pairs is so closed is a Horn sentence. And, since it is a Horn sentence, the union of a \subseteq -directed family of transitive relations is another transitive relation.

Being a group is a universal horn property [by which we mean that the axioms of group theory are universally quantified Horn formulæ] and we have the notion of closing a set of elements under an operation to obtain a group. Ditto ring, integral domain . . . but not field! [miniexercise: which of the field axioms is not horn?] That is why the concept of “least field extending \mathcal{F} containing some given elements” is not completely straightforward. You can obtain the least field extending \mathcal{F} containing some given elements but you don’t do it by taking the intersection of lots of fields.

Horn-ness of the declaration is not only sufficient for the closure to be legitimate, well-defined etc etc, but is necessary. See appendix 13.1.1.

⁴You may recall from an earlier Part II Set Theory and Logic sheet that the collection of transitive relations on a fixed set is a complete poset. If you didn’t prove it then, prove it now. Observe that the only feature of the property *transitive* that you have used in the proof is the fact that it is a Horn property.

⁵There is even a converse, something along the lines of: if the class of models of ϕ is closed under substructure and direct limits [and certain other things which i forget] then ϕ is logically equivalent to a Horn sentence. This might be proved in Dr Löwe’s *Model Theory* course in Lent.

3.4 Structural Induction

Recursive datatypes support **Structural Induction** (“ancestral induction” in Russell-and-Whitehead. ‘ancestral’ is in Russell-and-Whitehead, the idea—if not this particular terminology—goes back to Frege) and declaration of functions by **recursion**. Observe that this justification is constructive.

The way to understand structural induction is as a simple-minded generalisation from mathematical induction over \mathbb{N} : if one wants to show that every member of a retype has property F one first establishes that all the founders are F (as it were, prove $F(0)$) and that F -ness is preserved by the constructors (as it were: $F(n) \rightarrow F(n+1)$) at which point one infers that everything has F .

This section is so short because—altho’ this idea is epoch-making—it’s terribly simple, and there’s actually not much to say. One could make the point that lots of inductions that are represented as induction over \mathbb{N} are best understood as inductions over other retypes. For example in Logic there are various results about languages that one usually proves by mathematical induction over the number of quantifiers and connectives. These proofs are all (morally!) structural inductions over the retype of the language that is being reasoned about. There now follows a rather nice illustration from Part Ia of the Computer Science Tripos.

EXERCISE 5

“We define the length of a propositional formula by recursion as follows:

$$\begin{aligned} |a| &= 1, \\ |\top| &= 1, \\ |\perp| &= 1, \\ |A \wedge B| &= |A| + |B| + 1, \\ |A \vee B| &= |A| + |B| + 1, \\ |\neg A| &= |A| + 1. \end{aligned}$$

We define a translation which eliminates disjunction from propositional formulae by the following recursion:

$$\begin{aligned} tr(a) &= a, \quad tr(\top) = \top, \quad tr(\perp) = \perp, \\ tr(A \wedge B) &= tr(A) \wedge tr(B), \\ tr(A \vee B) &= \neg(\neg tr(A) \wedge \neg tr(B)), \\ tr(\neg A) &= \neg tr(A). \end{aligned}$$

Prove by structural induction on propositional formulae that

$$|tr(A)| \leq 3|A| - 1,$$

for all Boolean propositions A .”

EXERCISE 6

1. Declare the rectype of α -lists. (Observe that it is free.) Suppose the type α has been equipped with a quasiorder \leq_α . We say that an α -list l_1 **stretches into** another α -list l_2 if there is a 1-1 increasing map f from the addresses of l_1 to the addresses of l_2 such that, for all addresses a , $a \leq_\alpha f(a)$. That is to say: think of an α -list as a function defined on a proper initial segment of \mathbb{N} . Give a definition of stretching by list-recursion.
2. Declare the rectype of α -trees, and observe that it is free. Define stretching for α -trees, and give a recursive definition.

kfiniteness exercises here...?

With \mathbb{N} we can prove things by induction and define things by recursion. With other rectypes we can (as i have just illustrated) do (“structural”) induction, and we can also define functions by recursion. Natural and important examples of functions defined by recursion on other rectypes include recursive semantics for languages (which of course are recursive datatypes). There are communities who care a very great about the details of recursive semantics: theoretical computer scientists (there is even a 1B CS course devoted to it) and Linguists (the linguists speak of *compositional* semantics rather than *recursive* semantics). Mostly (but see subsection 3.10.2) we can take this kind of thing for granted.

3.5 Engendering Relations

All rectypes—since they are generated by constructors—will have a sort of **engendering relation**⁶ that is related to the constructors that generate the recursive datatype rather in the way that $<_{\mathbb{N}}$ is related to the successor function. The engendering relation is that binary relation that holds between an object x in the rectype and those objects “earlier” in the rectype out of which x was built. Thus it holds between a formula and its subformulae, between a natural number and its predecessors and so on. Put formally, the (graph of the) engendering relation is the transitive closure of the union of the (graphs of the) constructors.⁷

Some examples: $<_{\mathbb{N}}$ is the engendering relation of \mathbb{N} ; \in^* (the transitive closure of the membership relation) is the engendering relation of the cumulative hierarchy; the subformula relation is the engendering relation of the set of wffs of a language.

The (graph of, extension of) the engendering relation is itself a rectype. For example, $<_{\mathbb{N}}$ is the smallest set of ordered pairs containing all pairs $\langle 0, n \rangle$ with $n > 0$ and closed under the function that applies S to both elements of a pair (i.e., $\lambda p. \langle S(\text{fst } p), S(\text{snd } p) \rangle$).

The following triviality is important.

THEOREM 2 *The engendering relation of a rectype is well-founded.*

⁶This is not standard terminology.

⁷A joke from Allen Hazen: “is the transitive closure of” is the transitive closure of “is the transitive closure of”.

Proof: Let X be a subset of the rectype that has no minimal element in the sense of $<$, the engendering relation. We then prove by structural induction (“on x ”) that $(\forall y)(y < x \rightarrow y \notin X)$. ■

EXERCISE 7 (*)

- (i) Prove by structural (“mathematical”) induction on n that every $X \subseteq \mathbb{N}$ such that $n \in X$ has an S -least member;
- (ii) Prove by structural induction on n that $(\forall m \leq n)$ (every set containing m has a minimal element).

So obviously every nonempty subset of \mathbb{N} has an S -minimal element.

Related to this is the observation that if we can prove $(\forall n)F(n)$ by course-of-values induction then we can prove $(\forall n)(\forall m < n)F(m)$ by ordinary mathematical (structural) induction.

And it is of course dead easy to prove by course-of-values induction that $(\forall n)(\forall m < n)(\forall X \subseteq \mathbb{N})(m \in X \rightarrow X \text{ has a } <\text{-least member})$.

You have probably always been more-or-less happy that mathematical induction over \mathbb{N} and “strong” induction (or whatever you called it) over \mathbb{N} are equivalent. The time has come to make this explicit in your own mind so you can explain it to your students when the time comes.

Does every wellfounded relation arise from a rectype?

In general, structural induction over a rectype is equivalent to wellfounded induction over the engendering relation over that rectype. Wellfounded induction is in principle more general because there is always the possibility (in principle) of a relation being wellfounded without being the engendering relation of any rectype. Does this ever happen? It’s not quite clear how to frame this question so as to launch an illuminating research project. For the moment you might wish to contemplate the following *amuse gueule* which looks rather like a counterexample.

REMARK 2 *Every set of power sets has an \in -minimal member.*

Proof:

Let \mathcal{X} be a set of power sets with no \in -minimal element. We will show that \mathcal{X} is empty.

Suppose not; we will prove by induction that every wellfounded set belongs to everything in \mathcal{X} . Suppose A is a set such that, for all $a \in A$, a belongs to everything in \mathcal{X} . Let $\mathcal{P}(y)$ be an arbitrary member of \mathcal{X} , and let X be a member of \mathcal{X} that is also a member of $\mathcal{P}(y)$. Then $(\forall a \in A)(a \in X)$, which is to say, $A \subseteq X$. But $X \in \mathcal{P}(y)$ so all subsets of X are also in $\mathcal{P}(y)$, so in particular $A \in \mathcal{P}(y)$ as desired. But $\mathcal{P}(y)$ was an arbitrary member of \mathcal{X} .

This proves by \in -induction on the wellfounded sets that they all belong to everything in \mathcal{X} . But then $\bigcap \mathcal{X}$ must be a proper class, which is impossible. So \mathcal{X} must have been empty. ■

I am indebted to Tonny Hurkens for drawing my attention to this delightful fact. Savour the extreme minimalism! Not only does this proof not use choice, replacement or hardly any separation... it doesn't use any foundation: *the fact that \in -power sets is wellfounded is nothing to do with the cumulative hierarchy.*

3.6 Rectypes and Least Fixed Points

3.6.1 Fixed Point Theorems

We start with some old material from Part II, no longer examinable.

I assume you know the Tarski-Knaster theorem from Part II, so i shall not recapitulate it here. You were told the Bourbaki-Witt theorem, but were not shown a proof. So we'll have one here.

We say $f : X \rightarrow X$ is **inflationary** if $(\forall x \in X)(x \leq f(x))$.

THEOREM 3 *Every inflationary function from a chain-complete poset into itself has arbitrarily late fixed points.*

Proof: Let $\langle X, \leq \rangle$ be a chain-complete poset, f an inflationary function $X \rightarrow X$ and x a member of X . We will show that f has a fixed point above x .

The key device is the inductively defined set of things obtainable from x by repeatedly applying f and taking sups of chains—the smallest subset of X containing x and closed under f and sups of chains. Let us call this set $C(x)$. Our weapon will be induction.

We will show that $C(x)$ is always a chain. Since it is closed under sups of chains, it must therefore have a top element and that element will be a fixed point.

Let us say $y \in C(x)$ is **normal** if $(\forall z \in C(x))(z < y \rightarrow f(z) \leq y)$. We prove by induction that if y is normal, then $(\forall z \in C(x))(z \leq y \vee f(y) \leq z)$. That is to say, we show that—for all normal y — $\{z \in C(x) : z \leq y \vee f(y) \leq z\}$ contains x and is closed under f and sups of chains and is therefore a superset of $C(x)$. Let us deal with each of these in turn.

1. (Contains x) $x \in \{z \in C(x) : z \leq y \vee f(y) \leq z\}$ because $x \leq y$. ($x \leq y$ because x is the smallest thing in $C(x)$ —by induction!) The set of things $\geq x$ contains x , is closed under f and sups of chains and is therefore a superset of $C(x)$.
2. (Closed under f) If $z \in \{z \in C(x) : z \leq y \vee f(y) \leq z\}$, then either
 - (a) $z < y$, in which case $f(z) \leq y$ by normality of y and $f(z) \in \{z \in C(x) : z \leq y \vee f(y) \leq z\}$; or

- (b) $z = y$, in which case $f(y) \leq f(z)$ so $f(z) \in \{z \in C(x) : z \leq y \vee f(y) \leq z\}$; or
 - (c) $f(y) \leq z$, in which case $f(y) \leq f(z)$ (f is inflationary) and $f(z) \in \{z \in C(x) : z \leq y \vee f(y) \leq z\}$.
3. (Closed under sups of chains) Let $S \subseteq \{z \in C(x) : z \leq y \vee f(y) \leq z\}$ be a chain. If $(\forall z \in S)(z \leq y)$, then $\sup(S) \leq y$. On the other hand, if there is $z \in S$ s.t. $z \not\leq y$, we have $f(y) \leq z$ (by normality of y); so $\sup(S) \geq f(y)$ and $\sup(S) \in \{z \in C(x) : z \leq y \vee f(y) \leq z\}$.

Next we show that everything in $C(x)$ is normal. Naturally we do this by induction: the set of normal elements of $C(x)$ will contain x and be closed under f and sups of chains.

1. (Contains x) Vacuously!
2. (Closed under f) Suppose $y \in \{w \in C(x) : (\forall z \in C(x))(z < w \rightarrow f(z) \leq w)\}$. We will show $(\forall z \in C(x))(z < f(y) \rightarrow f(z) \leq f(y))$. So assume $z < f(y)$. This gives $z \leq y$ by normality of y . If $z = y$, we certainly have $f(z) \leq f(y)$, as desired, and if $z < y$, we have $f(z) \leq y \leq f(y)$.
3. (Closed under sups of chains) Suppose $S \subseteq \{w \in C(x) : (\forall z \in C(x))(z < w \rightarrow f(z) \leq w)\}$ is a chain. If $z < \sup(S)$, we cannot have $(\forall w \in S)(z \geq f(w))$ for otherwise $(\forall w \in S)(z \geq w)$ (by transitivity and inflationarity of f), so for at least one $w \in S$ we have $z \leq w$. If $z < w$, we have $f(z) \leq w \leq \sup(S)$ since w is normal. If $z = w$, then w is not the greatest element of S , so in S there is $w' > w$ and then $f(z) \leq w' \leq \sup(S)$ by normality of w' .

If y and z are two things in $C(x)$, we have $z \leq y \vee f(y) \leq z$ by normality of y , so the second disjunct implies $y \leq z$, whence $z \leq y \vee y \leq z$. So $C(x)$ is a chain as promised, and its sup is the fixed point above x whose coming was foretold. ■

3.6.1.1 Exercises on fixed points

EXERCISE 8 Show that the fixed point of the Tarski-Knaster theorem is \leq_X -minimal.

EXERCISE 9 Let $\langle A, \leq \rangle$ and $\langle B, \leq \rangle$ be total orderings with $\langle A, \leq \rangle$ isomorphic to an initial segment of $\langle B, \leq \rangle$ and $\langle B, \leq \rangle$ isomorphic to a terminal segment of $\langle A, \leq \rangle$. Show that $\langle A, \leq \rangle$ and $\langle B, \leq \rangle$ are isomorphic.

You used an analogue of the function in the Tarski-Knaster proof of the Cantor-Bernstein theorem (theorem ??). What can you say about the set of its fixed points?

EXERCISE 10 Let R be a binary relation on a set X . Let F be a fuzzy on X . Define a new fuzzy on X by $xF'y$ iff $(\forall x')(x'Rx \rightarrow (\exists y')(y'Ry \wedge x'Fy')) \wedge (\forall y')(y'Ry \rightarrow (\exists x')(x'Rx \wedge y'Fx'))$. Show that for all X , R and F there is a fixed point for the function taking F to F' . Naturally you have used the Tarski-Knaster theorem. What is the lattice you are using? Now do the same with the assumption that F is an equivalence relation not a mere fuzzy. What lattice are you using now? Prove that it is not distributive.

EXERCISE 11 (The Gale-Stewart theorem) A combinatorial game G of length n is defined by a set A (the “arena”) from which players I and II pick elements alternately, thereby building an element of A^n (a “play”). G is a subset of A^n , and I wins a play p of G iff $p \in G$. Otherwise II wins.

Provide a formal notion of **winning strategy** for games of this sort, and prove that one of the two players must have a winning strategy in your sense.

Now replace ‘ n ’ by ‘ ω ’ in the above definition, so that plays are infinite sequences. Give A the discrete topology and A^ω the product topology.

Use Bourbaki-Witt to show that if G is open in the product topology then one or the other player must have a winning strategy.

[This is not best possible. The game is determinate as long as G is Borel... but that needs AC]

EXERCISE 12 What might the well-founded part of a binary relation be? Use one of the fixed point theorems to show that your definition is legitimate.

EXERCISE 13 An old examination question, principally for revision.

(i) State and prove the Tarski-Knaster fixed point theorem for complete lattices.

(ii) Let X and Y be sets and $f : X \rightarrow Y$ and $g : Y \rightarrow X$ be injections. By considering $F : \mathcal{P}(X) \rightarrow \mathcal{P}(X)$ defined by

$$F(A) = X \setminus g(Y \setminus f(A))$$

or otherwise, show that there is a bijection $h : X \rightarrow Y$.

(iii) Suppose U is a set equipped with a group Σ of permutations. We say that a map $s : X \rightarrow Y$ is piecewise- Σ just when there is a finite partition $X = X_1 \sqcup \dots \sqcup X_n$ and $\sigma_1 \dots \sigma_n \in \Sigma$, so that $s(x) = \sigma_i(x)$ for $x \in X_i$. Let X and Y be subsets of U , and $f : X \rightarrow Y$ and $g : Y \rightarrow X$ be piecewise- Σ injections. Show that there is a piecewise- Σ bijection $h : X \rightarrow Y$.

(iv) If $\langle P, \leq_P \rangle$ and $\langle Q, \leq_Q \rangle$ are two posets with order-preserving injections $f : P \rightarrow Q$ and $g : Q \rightarrow P$, must there be an isomorphism? Prove or give a counterexample.

EXERCISE 14 (Probably only for Group Theorists)

A group G is **complete** iff it is isomorphic to $\text{Aut}(G)$, its automorphism group. Show that every group embeds in a complete group. (You may assume that for any G there is a set of groups containing G and closed under Aut and unions of chains.)

Show also that if G has trivial centre so does $\text{Aut}(G)$, and thence that every group with trivial centre embeds in a complete group with trivial centre.

3.6.2 Rectypes as least fixed points

If you are a rectype it is because you are the lfp of a certain increasing function from the complete poset $\langle V, \subseteq \rangle$ of all sets⁸ into itself. Here are some examples.

$$\mathbb{N} = \bigcap \{X : (\{0\} \cup S^*X) \subseteq X\}.$$

(which says that \mathbb{N} is the lfp for $\lambda X.(\{0\} \cup S^*X)$)

H_{\aleph_1} is the least fixed point for $x \mapsto \mathcal{P}_{\aleph_1}(x)$.

($\mathcal{P}_{\aleph_1}(x)$ is the set of countable subsets of x . See Q9 Sheet 4, PTJ Set Theory and Logic 2012/3 for other examples, such as the set of hereditarily small sets).

It is possible to think of the way that rectypes support structural induction as arising from their status as least fixed points for monotone operations.

Explain this in more detail!!

EXERCISE 15 A **D-finite set** is a set without a countably infinite subset.

- (i) Prove that every hereditarily D-finite set is inductively finite;
- (ii) Provide a constructive proof that every hereditarily Kfinite set is N-finite.

Do not attempt part (ii) unless/until you are fluent in constructive logic. In both these cases we mean ‘hereditarily’ in the sense of the least fixed point.⁹

[Brief excursion into Set Theory. If we do not assume that \in is wellfounded then “hereditarily finite” could mean something other than V_ω . It could be the greatest fixed point for $x \mapsto \mathcal{P}_{\aleph_0}(x)$ (the set of finite subsets of x) which of course will be $\bigcup \{x : x \subseteq \mathcal{P}_{\aleph_0}(x)\}$. This object might or might not be a set. All bets are off.]

3.7 Finite vs Bounded vs Unbounded Character

Restricting oneself to Horn Clauses in a datatype declaration solves the Uniqueness Problem. There remains the Existence Problem. The first attempt at cracking the Existence Problem represents the target object as a least fixed point for some function from the complete poset of sets-under-inclusion into itself. Such a fixed point is the intersection of a family. Can we be sure that the family is nonempty? The intersection of the empty set is the universe, and that

⁸And don’t tell me that $\langle V, \subseteq \rangle$ isn’t a complete poset because it hasn’t got a top element. Go And Sit In The Corner.

⁹That is: the set of hereditarily D-finite sets is the \subseteq -least set identical to the set of its D-finite subsets.

is clearly not the answer one wants! This is one of the situations where the fact that ZFC countenances only small sets makes for very unnatural developments. This is the **Empty Intersection Problem**. It plays out differently depending on whether the datatype being declared is of finite, infinite-but-bounded or absolutely infinite character.

We have seen examples of reatypes of finite character (\mathbb{N} , language of first-order Logic, etc etc). Here are some of infinite character

- (i) H_{\aleph_1} ;
- (ii) The set of Borel sets in a topological space;
- (iii) The family of Conway games;
- (iv) The collection of all ordinals;
- (v) The Cumulative Hierarchy.

Explain these expressions

Reatypes (i) and (ii) are of bounded (countable) character; (iii)–(v) are of *unbounded* (“absolutely infinite”) character.

Minixercise: provide recursive declarations of (ii)–(v). (We declared (i) in section 3.6.2.)

You have probably not worried at all about whether or not reatypes of finite character (\mathbb{N} , language of first-order Logic, etc etc) exist as sets, having taken it for granted all along that they do—as indeed they do. The existence-as-sets of any and all reatypes of finite character is actually the precise content of the axiom of infinity; that’s what it’s for. Another way of saying that these objects can be taken to be sets is to say that they are not paradoxical objects.

3.7.1 Reatypes of Unbounded Character are Paradoxical

Let’s get out of the way the fact that reatypes of unbounded character are practically guaranteed *not* to be sets: they are paradoxical. (Interestingly the corresponding *co*-reatypes tend not to be paradoxical). The following are all paradoxical:

- (i) The class of hereditarily transitive sets (the lfp for $x \mapsto \{y \subseteq x : \bigcup y \subseteq y\}$;
- (ii) The class of wellfounded sets (the lfp for $x \mapsto \mathcal{P}(x)$);
- (iii) The class of hereditarily wellordered sets (the lfp for $x \mapsto \{y \subseteq x : y \text{ is wellordered}\}$).

(i) corresponds to the Burali-Forti paradox *via* the von Neumann implementation of ordinals; (ii) is Mirimanoff’s paradox.

If you are planning to master Set Theory you may wish to work through (i) and (ii). For (i) you want to show that the collection of hereditarily transitive sets is a paradoxical object (cannot be a set), and you also want to show that the collection of Von Neumann ordinals is precisely the collection of hereditarily transitive sets.

(A word of warning: not all the paradoxical collections that you may know are rectypes: the Russell class and its congeners— $\{x : x \notin^n x\}$ —are paradoxical but are not recursively defined.)

The contradictions associated with Least-fixed-points for constructors of Absolutely Infinite Character tend to be very easy to prove. Typically one needs only *subscission*:

$$x \setminus \{y\} \text{ exists for all } x \text{ and } y. \quad \textbf{Subscission}^{10}$$

I’m pretty sure that subscission suffices for (i)–(iii), and (i think) it suffices for the following, which is the most general impossibility result in this direction known to me. Be sure to use only subscission when answering exercise 16.

EXERCISE 16 (*)

Suppose f is monotone and injective: $(\forall xy)(x \subseteq y \longleftrightarrow f(x) \subseteq f(y))$. Let $A := \bigcap \{x : \mathcal{P}(f(x)) \subseteq x\}$. Then A is not a set.

So let’s not worry too much about trying to prove the sethood of rectypes of absolutely infinite character: it’s not to be expected, and indeed one can often prove the non-sethood of a least fixed point with quite modest set-theoretic assumptions (such as subscission, above).

The Empty Intersection Problem is a *huge* problem for rectypes of unbounded character (which in any case tend not to be sets). The problem is not merely that the collection of things over which we are intersecting is not a set, the problem is that the things we are intersecting over might not *themselves* even be sets but merely proper classes, and that in turn means—*prima facie* at least—that membership of the least fixed point is not first-order.

Wellfounded and “Regular” Sets in Set Theory

The cumulative hierarchy is a retype of unbounded (absolutely infinite) character: the obvious inductive definition of wellfounded sets defines $WF(x)$ as $(\forall y)(\mathcal{P}(y) \subseteq y \rightarrow x \in y)$.

If we want to do this in ZF we have an Empty Intersection Problem, because we can prove there are no such y .

There are various ways round this problem. We can say that x is wellfounded iff it belongs to all *classes* that contain all their subsets (so that the ‘ y ’ ranges over *all* classes and not just those that happen to be sets). But of course that would mean we are no longer in ZF but instead in Gödel-Bernays).

So let’s assume that the variables range only over sets, and play a few tricks, have some fun.

Suppose x satisfies $(\forall y)((\forall z)(z \subseteq y \rightarrow z \in y) \rightarrow x \in y)$.
Substitute $V \setminus y$ for y getting

¹⁰I don’t think this is standard terminology; i learnt it from Allen Hazen and I think the expression is his coinage.

$$(\forall y)((\forall z)(z \cap y = \emptyset \rightarrow z \notin y) \rightarrow x \notin y).$$

Contrapose getting

$$(\forall y)(x \in y \rightarrow \neg(\forall z)(z \cap y = \emptyset \rightarrow z \notin y)).$$

This is

$$(\forall y)(x \in y \rightarrow (\exists z)(z \cap y = \emptyset \wedge z \in y))$$

... which says that x is regular. *Regular set* is a way of defining *wellfounded set* without quantifying over classes. You will recall from Part II Set Theory and Logic that regular sets obey \in -induction. This tells you why they do!

In this piece of trickery we have used an axiom of complementation. You probably find that alarming but actually it's harmless. The *real* damage is done by things you probably didn't notice. We exploited the fact that any conditional $A \rightarrow B$ is logically equivalent to its contrapositive $\neg B \rightarrow \neg A$, and that $\neg\forall$ is equivalent to $\exists\neg$. These two principles are not constructively valid. There is a constructive theory of wellfounded sets, but it does not support proofs using " \in -minimal elements". (Similarly constructive arithmetic does not support the least-number principle).

Natural Numbers and Quine's trick

\mathbb{N} is a retype of finite character, and there is no problem about its sethood as long as we have an axiom of infinity. However there are subtleties that remind one of the definition of the cumulative hierarchy and which it is sensible to consider in connection with it.

The "top-down" definition of \mathbb{N} involves quantifying over infinite sets. The finite/infinite dichotomy feels a bit like the set/proper-class dichotomy so—just as we wanted to be able to define *well-founded set* without talking about *classes*—we would like to be able to define *natural number* without talking about *infinite sets*.

We now give a definition of \mathbb{N} (due to Quine) that does not involve quantification over infinite sets and prove that it is the same as the usual definition.

EXERCISE 17 (*) (Part III Logic and Combinatorics Exam 2006 q 12)

Let $P(|x|)$ be $|x \setminus \{y\}|$ if $y \in x$ and 0 if x is empty.

Define

$$q(n) \longleftrightarrow (\forall Y)((n \in Y \wedge (P"Y \subseteq Y)) \rightarrow 0 \in Y)$$

Establish that $q(n)$ iff n is a natural number according to the usual definition.

For a more detailed discussion of the history of this idea see [52] pp. 75–6.

The critical fact about this definition of $q(n)$ is that it makes sense even if the Y we are quantifying over are all finite. One can check whether or not $q(n)$ without examining any infinite sets.

(Like the definition of regular set this definition is not constructive.)

We can also define finite as

$$Fin(x) \longleftrightarrow (\forall X \subseteq \mathcal{P}(x))((\emptyset \in X \wedge (\forall x' \in X)(\forall y \in x)(x' \cup \{y\} \in X) \rightarrow x \in X)).$$

If n is, in fact, finite in this sense then the investigation that will establish this fact will not commit us to examining any infinite sets. However if it is not then the investigation will lead us into the infinite. So this definition is less clean than the definition of $q(n)$. Also it needs power set.

This definition is actually, literally, Kuratowski's definition of "finite", from which our earlier definition of Kfinite was borrowed.

3.7.2 Bounded Character

In contrast the general idea is that recursive families of *bounded* character are safe (*i.e.* not paradoxical) and can be proved to be sets if we try hard enough. How does one try?

There are two ways.

"From Below"

This is the usual solution in the ZF world: define a function that enumerates the "layers" of the retype, and then use an instance of the axiom scheme of replacement to form the set of all the layers. The \bigcup axiom then gives you the retype. This works for retypes of finite character because there are only ω layers, and we use replacement for a function defined on \mathbb{N} that enumerates the layers. Retypes of infinite but bounded character require a longer construction but the idea is the same. (For example the construction of H_{\aleph_1} closes off within ω_2 steps, or in precisely ω_1 steps if we have AC).

trim this para a bit

"From above"

This is the morally correct way, but if you define a retype as the intersection of all sets containing the founders and closed under the constructors you will be in trouble unless there are some sets containing the founders and closed under the constructors. In these circumstances one is in peril of the Empty Intersection Problem that we mentioned earlier.

The Empty Intersection Problem is much more obviously a problem for retypes of infinite character than for retypes of finite character (tho' not for Borel sets in a fixed topological space). *HC* (see Q9 Sheet 4, PTJ Set Theory and Logic 2012/3) the set of hereditarily countable sets, is the \subseteq -least set containing all its countable subsets, so it's the intersection of all the sets that contain all their countable subsets. How do we know there are such sets? If we have countable choice then V_{ω_1} is such a set [miniexercise: why?] but we can actually prove it without any use of choice.

One powerful argument in favour of adopting the axiom scheme of replacement is that it enables us to prove the sethood of least fixed points for operations of infinite (if bounded) character.

3.8 Ordinals

One particularly important retype of infinite character is the ordinals. The ordinals are a retype of unbounded character. In this course—mostly—the only ordinals we will be concerned with are the *countable* ordinals, and the countable ordinals form a set, Cantor’s *second number class*. It’s a set because it’s a surjective image of \mathbb{R} .

See my TMS talk [23] and my tutorial on countable ordinals [24].

We can declare the ordinals in the same way that we declare the naturals, but we have to add another constructor, of **sup** that takes a set of ordinals and returns an ordinal. This is not free, because $\{2n : n \in \mathbb{N}\}$ and $\{2n+1 : n \in \mathbb{N}\}$ give the same output when whacked with **sup**. This means that proving that the engendering relation on ordinals (or, strictly, its restriction to the ordinals themselves namely $<_{On}$) is a wellorder is actually quite tricky. Look again at the proof of theorem 3. You will now see that the bulk of the work goes into showing that the family of all the iterated images of the bottom element under the inflationary function form a total ordering. But this family has a recursive declaration which is the same as the recursive declaration of the ordinals.

The engendering relation on On is simply $<_{On}$. Transfinite induction and recursion work because this relation is wellfounded.

3.8.1 Rank functions

Every wellfounded structure has a homomorphism into the ordinals, a *rank function*, defined by recursion. And *vice versa*: every structure with a rank function is wellfounded. Rank functions are *parsimonious* in the following sense. When $\langle X, R \rangle$ and $\langle Y, S \rangle$ are wellfounded structures a morphism $f : \langle X, R \rangle \rightarrow \langle Y, S \rangle$ is *parsimonious* if for all $x \in X$, $f(x)$ is an S -minimal member of $\{y : (\forall x' R x)(f(x') S y)\}$.

If $\langle X, R \rangle$ be a wellfounded structure, then the rank function $\rho : X \rightarrow On$ is the unique parsimonious morphism $f : \langle X, R \rangle \rightarrow On$.

I’m no categorist so don’t take my word for it, but i think the following is true. On is the terminal object in the category of wellfounded structures where the morphisms are parsimonious maps; \mathbb{N} is the terminal object in the category of retypes of finite character and parsimonious maps.

3.9 Restricted Quantifiers

Quantifiers in the style ‘ $(\forall x R y)(\dots)$ ’ and ‘ $(\exists x R y)(\dots)$ ’ are said to be **re-stricted**. The intended semantics treats them as ‘ $(\forall x)(x R y \rightarrow \dots)$ ’ and ‘ $(\exists x)(x R y \wedge \dots)$ ’. In principle this syntax can be used whatever the relation R is (and there is CS literature on this general situation, where this phenomenon

is called ‘guarded quantification’¹¹) but the two *loci classici* of restricted quantifiers are (i) [wellfounded] set theory, with the quantifiers ‘ $(\forall x \in y)(\dots)$ ’ and ‘ $(\exists x \in y)(\dots)$ ’ and (ii) arithmetic of \mathbb{N} , where the quantifiers are ‘ $(\forall x < y)(\dots)$ ’ and ‘ $(\exists x < y)(\dots)$ ’. In both these classical settings the binary relation doing the guarding is the engendering relation of the rectype¹²

Notion of end-extension (preserves formulæ in which the only quantifiers are restricted quantifiers)

DEFINITION 3

Let $\mathfrak{M} \subseteq \mathfrak{M}'$ be structures for a language with a binary relation symbol ‘ R ’. We say \mathfrak{M}' is an **end-extension** of \mathfrak{M} (with respect to ‘ R ’ understood) if $(\forall m \in M)(\forall m' \in M')(m' R m \rightarrow m' \in M)$.¹³

EXERCISE 18 (*)

Give a [n inductive] definition of the **wellfounded part** of a binary structure, and establish that every [binary] structure is an end-extension of its wellfounded part.

Next we note without proof quantifier-pushing and quantifier-squashing for $<_{\mathbb{N}}$ -restricted formulæ. . . “Collection”.

Explain axiom scheme of collection here

REMARK 3

$(\forall x <_{\mathbb{N}} y)(\exists z)\psi(x, y, z)$ is equivalent to $(\exists w)(\forall x <_{\mathbb{N}} y)(\exists z <_{\mathbb{N}} w)\psi(x, y, w)$; (“quantifier pushing”)

$(\exists u)(\exists v)\phi(u, v)$ is equivalent to $(\exists w)(\exists u <_{\mathbb{N}} w)(\exists v <_{\mathbb{N}} w)\phi(u, v)$ (“quantifier-squashing”)

You learnt in Part II that quantifier-free formulæ are preserved upward and downward; formulæ that have no unrestricted quantifiers are preserved upward and downward where the extensions are end-extensions.

More formally:

If \mathfrak{M} is a substructure of \mathfrak{N} and ϕ is quantifier-free then $\mathfrak{M} \models \phi$ iff $\mathfrak{N} \models \phi$ ¹⁴;

If \mathfrak{M} is a substructure of \mathfrak{N} is an end-extension of \mathfrak{M} and ϕ has no unrestricted quantifiers then $\mathfrak{M} \models \phi$ iff $\mathfrak{N} \models \phi$.

¹¹I don’t think this different terminology reflects a difference in motivation. My guess is that the reason why CS people use a different word is simply that they didn’t know that logicians had got there before them.

¹²Well, \in is not the engendering relation of the cumulative hierarchy, but its transitive closure is, and a binary relation is wellfounded iff its transitive closure is. This is an exercise somewhere—or should be! Prove it

¹³However, we will also use this expression in a setting where a string t is a string s with extra stuff on the end. . . we will say that t is an end-extension of s .

¹⁴You proved this in Part II.

Quantifier hierarchies for restricted quantifiers. (Not sensible without a well-foundedness condition. If there is a universal set then every expression in the language of set theory is in $\Pi_2 \cap \Sigma_2$. Miniexercise: how so?)

Explain axiom scheme of collection at this point

In the theory of the cumulative hierarchy there is a normal form theorem for restricted quantifiers proved using collection/replacement.

A Δ_0 -formula in the language of set theory is a formula built up from atomics by means of boolean connectives and restricted quantifiers. Thereafter a Σ_{n+1} (respectively Π_{n+1}) formula is the result of binding variables in a Π_n (respectively Σ_n) formula with existential (respectively universal) quantifiers. We immediately extend the Σ_n and Π_n classes by closing them under interdeducibility-in-a-theory- T , and signal this by having ‘ T ’ as a superscript so our classes are Σ_n^T and Π_n^T .

This linear hierarchy of complexity for formulæ will be very useful to us in understanding T if we can be sure that every formula belongs to one of these classes¹⁵: it is standard that we can give a Π^{n+1} truth-definition for Σ_n formulæ. That is to say, we desire a *normal form theorem* for T .

It is easy to check that if T is not ludicrously weak we can show that Π_n^T and Σ_n^T are closed under conjunction and disjunction. To complete the proof of the normal form theorem we would need to show that these classes are closed under restricted quantification. After all, if ϕ is a Π_n^T formula what kind of a formula is $(\exists x \in y)\phi$? It would be very simple if it, too, were Π_n^T . It’s plausible that it should be Π_n^T (it has the same number of blocks of unrestricted quantifiers after all) but it is not at all obvious. Nevertheless there are sound philosophical reasons why we might expect it to be—at least if $V = WF$. The point is that WF is a recursive datatype, and recursive datatypes always have a sensible notion of restricted quantifier, and typically one can prove results of this kind for the notion of restricted quantifier that is in play. In general, when dealing with a recursive datatype, we can define Δ_0 formulæ—as above—as those with no unrestricted quantifiers, where we take restricted quantifiers to be ‘ $(\exists x)(R(x, y) \wedge \dots)$ ’ and ‘ $(\forall x)(R(x, y) \rightarrow \dots)$ ’, and R is the engendering relation. We find that Δ_0 formulæ behave in many ways as if they contained no quantifiers at all. An unrestricted quantifier is an injunction to scour the whole universe in a search for a witness or a counterexample; a restricted quantifier invites us only to scour that part of the universe that lies in some sense “inside” something already given. The search is therefore “local” and should behave quite differently: that is to say, restricted universal quantification ought to behave like a finite conjunction and ought to distribute over disjunction in the approved de Morgan way. (And restricted existential quantification too, of course).

The **Prenex Normal Form Theorem** states that every expression in LPC is logically equivalent to a formula in *Prenex Normal Form*, which is to say a formula wherein all the propositional connectives lie within the scope of all the quantifiers. You were not told this at Part II, but you might like to prove it now.

¹⁵well, *lots* of these classes: after all if ϕ is in Σ_n^T it is also in Π_{n+1}^T .

What we will now see is that, if we have the axiom scheme of collection, then we can prove an analogue of the Prenex Normal Form Theorem:

THEOREM 4 *Given a theory T , which proves collection, for every expression ϕ of the language of set theory there is an expression ϕ' s.t. $T \vdash \phi \longleftrightarrow \phi'$ and every restricted quantifier and every atomic formula occurs within the scope of all the unrestricted quantifiers.*

Proof: It is simple to check that $(\forall x)(\forall y \in z)\phi$ is the same as $(\forall y \in z)(\forall x)\phi$ (and similarly \exists), so the only hard work involved in the proof is in showing that

$$(\forall y \in z)(\exists x)\phi$$

is equivalent to something that has its existential quantifier out at the front. (This case is known in logicians' slang as "quantifier pushing".) By collection we now infer

$$(\exists X)(\forall y \in z)(\exists x \in X)\phi,$$

and the implication in the other direction is immediate.

This shows that Σ_n is closed under restricted universal quantification. Dually we infer that Π_n is closed under restricted existential quantification. It is of course immediate that Σ_n is closed under restricted existential quantification and that Π_n is closed under restricted universal quantification. ■

collection follows trivially from the existence of a universal set

Now have the analogue of the prenex normal form theorem we can complete the proof that every formula belongs to one of the classes Π_n^T or Σ_n^T .

3.10 Infinitary Languages

DEFINITION 4 *The language $L_{\kappa\lambda}$ is (like first-order Logic) a recursive datatype but differs from it in being closed under conjunctions and disjunctions of lists of expressions of length $< \kappa$ and allows us to bind $< \lambda$ variables with \forall at one hit.*

(All the quantifiers in an infinite block have to be of the same flavour)

Thus ordinary predicate calculus is $L_{\omega,\omega}$. There are various other languages we can notate in this way, some of which we will consider. On the whole these other languages are quite nasty: compactness fails, for example. $L_{\omega_1,\omega}$ is often considered, and it admits a kind of compactness theorem. $L_{\kappa,\kappa}$ is well-behaved if κ has some nice large cardinal properties of the kind you may learn about if you are doing Part III Topics in Set Theory. Our immediate use is for L_{ω_1,ω_1} .

(Don't be spooked by the infinitary nature of the constructors into thinking that the subformula relation for these languages is illfounded.)

3.10.1 “Wellfounded” is Infinitary Horn

Exercise 19 is a simple exercise in the style of Sheet 3 of last year’s Part II Logic and Set Theory. (It’s actually part (iv) of Part II Paper 3 q 16H in 2011.)

EXERCISE 19 *Show that there is no first-order theory of a wellfounded relation.*

However wellfoundedness is first-order in L_{ω_1, ω_1} .

$$(\forall x_1)(\forall x_2) \dots \bigvee_{i < j < \omega} \neg R(x_j, x_i)$$

This is a formula of L_{ω_1, ω_1} . Indeed it is even Horn!

Given that wellfoundedness is infinitary Horn we should not be surprised to find that the class of wellfounded structures is closed under products, quotients, substructures and directed unions under end-extension (recall p 33). Question 2009-3-16G from http://www.maths.cam.ac.uk/undergrad/pastpapers/2009/Part_2/index.html makes you think about why the directed unions have to be ordered by end-extension ... and mere inclusion is not enough.

The following non-obvious fact will come in useful later, and the reader is invited to prove it if they have not already done so.

EXERCISE 20 (*)

The lexicographic product of two wellfounded strict partial orders is wellfounded.

The pointwise product of two wellfounded strict posets is wellfounded by horn-ness, and every subset of a wellfounded relation is wellfounded because ‘wellfounded’ is \forall in L_{ω_1, ω_1} .

3.10.2 Some Remarks on Infinitary Languages

Some of the infinitary expressions to which we are accustomed are illfounded. And semantics for them can depend sensitively on things that it would seem shouldn’t matter. There are ways of putting brackets into an infinite sum so that the result is no longer an infinite sum but an illfounded expression:

$$a_0 + (a_1 + (a_2 + \dots))$$

Some are straight-out illfounded:

$$\sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2 + \dots}}}}$$

Since the subformula relation on these expressions is not wellfounded there is no way of defining a recursive semantics for them. Indeed it is a condign lesson of first-year Analysis that semantics for these infinitary expressions is not a trivial exercise, and (pace the remarks on p. 22) one spends a lot of the early stages of Analysis learning that a lot of things that oughtn’t to matter, do, indeed, not matter. See the footnote on page 74.

3.11 Greatest fixed points (“Co-rectypes”)

Rectypes are least fixed points. If you have used Tarski-Knaster to prove the existence of rectypes as least fixed points, you will be prepared for the news that there are things called *co-rectypes* that are greatest fixed points.

- The collection H_{\aleph_0} of (wellfounded) hereditarily finite sets is the lpf $\bigcap \{Y : \mathcal{P}_{\aleph_0}(Y) \subseteq Y\}$ for the function $\lambda y. \mathcal{P}_{\aleph_0}(y)$ where $\mathcal{P}_{\aleph_0}(y) = \{x \subseteq y \mid |x| < \aleph_0\}$. $\bigcap \{Y : \mathcal{P}_{\aleph_0}(Y) \subseteq Y\}$ is better known as V_ω .

The corresponding *greatest* fixed point is $\bigcup \{Y : Y \subseteq \mathcal{P}_{\aleph_0}(Y)\}$. Whether or not this object is the same as V_ω depends on whether or not the axiom scheme of foundation holds. In principle it might contain *Quine atoms* (objects $x = \{x\}$) and other such suspect entities.

- The cumulative hierarchy is the lfp for \mathcal{P} : $\bigcap \{Y : \mathcal{P}(Y) \subseteq Y\}$

The greatest fixed point corresponding to this is $\bigcup \{Y : Y \subseteq \mathcal{P}(Y)\}$. On the (modest) assumption that every set has a transitive superset this is the whole of V .

EXERCISE 21

1. The rectype of α -lists is the lfp for a certain function. What function, precisely? The corresponding gfp is the datatype of α -streams (infinite lists). Declare a co-rectype of α -trees and define stretching for it.
2. Do the same for α -trees.

EXERCISE 22 *Co-rectypes support co-induction. Explain and justify co-induction.*

3.12 Certificates

We close this chapter with a little *sleeper*. Most of the rectypes we are interested in are rectypes of finite character. If you are an element of a rectype of finite character then there is a good finite reason for you to be a element. These good finite reasons are sometimes called *certificates*, tho’ of course there can be certificates for infinite things too. A **certificate that** p (wrt some very weak background theory T) is an object x satisfying some [very elementary, probably quantifier-free] property ϕ s.t. $T \vdash (\exists x)(\phi(x)) \rightarrow p$.

[a warning (for the future, not the present); ‘ p ’ could have parameters in it, so that a certificate (wrt some very weak background theory T) that $p(\vec{y})$ is an object x satisfying some [very elementary, probably quantifier-free] property ϕ s.t. $T \vdash (\forall \vec{y})((\exists x)(\phi(\vec{y}, x)) \rightarrow p(\vec{y}))$.]

These things are sometimes called *proofs* but I shall stick to ‘certificate’, because the word ‘proof’ has other uses here. (In fact proofs will be certificates of a special kind). For example, if you are the number 0 you are a natural

number by *flat*; if you are $S(n)$ for some natural number n then a good finite reason for you to be a natural number is whatever-is-a-good-reason-for- n -to-be-a-natural-number plus a cry of “*successor!*”. This evidently gives us a recursive conception of certificate-that-an-object-is-a-natural-number, and that certificate is evidently the act of counting from 0 up to that number.

More generally, if an object x in a retype is constructed from objects $y_1 \dots y_n$ by means of a constructor f , then a certificate that x belongs to the retype is the ordered pair of f and the list of certificates for the \vec{y} . [the word *pedigree* or perhaps *provenance* might be better]. Clearly certificates for a retype themselves form a retype.

When T is a theory, the retype of T -proofs is the retype of proofs/certificates of membership of the retype that is the theory T .

Presumably “ p is a certificate that x is in retype A ” is Δ_0 or perhaps Δ_1 in the language with the engendering relation. So belonging to a retype is Σ_1 in some suitable language... semidecidable sets!

3.12.1 Free vs non-free retypes (ambiguous parses)

A retype is *free* if every element of it has precisely one certificate. Any free retype is a surjective image of its retype of certificates (and retypes of certificates are always free). If a retype is not free, and not of finite character, there will be a mess, and we need the axiom of choice to clear it up. Consider for example the retype founded by all the countable sets, where the single (infinitary) constructor is countable union (union of countably many inputs). The assertion that every object of this retype has a certificate implies that a union of countably many countable sets is countable.

The retype of finite sets is not free: a finite set can be constructed from \emptyset and adjunction $(x, y \mapsto x \cup \{y\})$ in more than one way: “there is more than one certificate”. This complicates definition by recursion because it requires us to check that all certificates for a set x get treated the same way by the recursion. It doesn’t make recursion illegitimate, but it does mean we have to be careful. For example if we declare V_ω as containing \emptyset and being closed under $x, y \mapsto x \cup \{y\}$ then one cannot successfully define $f : V_\omega \rightarrow V$ by $f(\emptyset) = \emptyset$; $f(x \cup \{y\}) = f(x) \times \{y\}$, because we could have $x \cup \{y\} = z \cup \{w\}$ with $x \neq z \wedge y \neq w$.

3.12.2 A Last Crazy Thought

The set of diatonic melodies (or, for that matter, the set of piano pieces of finite length that are compatible with the rules of late C19th harmony) is a countable set, indeed a retype (as countable sets typically are). So what is it to create one of its members? Surely all we can do is discover them? This is a known problem for platonistic philosophies of mathematics: they appear to leave no space for creativity. This looks to me like a problem that the notion of

rectype-with-certificates can shed light on. *You create a piece by executing the certificate.*

Chapter 4

Functions: Primitive Recursive and μ -recursive

“Can you do addition?” the White Queen asked. “What’s one and one and one and one and one and one and one and one and one and one?”

“I don’t know,” said Alice “I lost count.”

“She can’t do addition” the Red Queen interrupted.

see [11], available online.

4.1 Primitive Recursion

We consider primitive recursion over \mathbb{N} in the first instance.

DEFINITION 5 *The rectype of **primitive recursive functions** is the \subseteq -least class of functions containing the **initial** functions, which are*

*the **successor** function: $n \mapsto n + 1$, written S ;*

*the **projection** functions: proj_n^m is an m -ary function which returns the n th of its arguments;*

*the **constantly zero** function: the function that always returns 0.*

... and closed under

*(i) **composition** (see below) and*

*(ii) **primitive recursion**:*

$$f(\vec{x}, 0) := g(\vec{x}); \quad f(\vec{x}, S(y)) := h(\vec{x}, y, f(\vec{x}, y)). \quad (4.1)$$

*In 4.1 we say f is declared by **primitive recursion** over g and h . We **recurse** on the variable ‘ y ’. The ‘ \vec{x} ’ variables are the **snail variables**—those you just carry around and do not recurse on.*

What might primitive recursion on other rectypes be?

EXERCISE 23 (*)

Need a discussion answer

1. Consider for example the rectype of α -lists from p. 20. What is primitive recursion on α -lists?

If α is equipped with an order \leq_α we say that an α -list l_1 (thought of as a function $l_1 : [1, n] \rightarrow \alpha$, for some n) stretches into an α -list l_2 (thought of as a function $l_2 : [1, m] \rightarrow \alpha$, for some $m \geq n$) if there is an order-preserving injection $g : [1, n] \hookrightarrow [1, m]$ such that $(\forall i \leq n)(l_1(i) \leq_\alpha l_2(g(i)))$.

Give a definition of stretching by primitive recursion on α -lists.

2. Next consider the rectype of α -trees, where the set of children of each node (a litter) is a list of α -trees.

Give a rectype declaration for rectype of α -trees.

What is primitive recursion on α -trees?

Define stretching for α -trees, both directly as above (for lists), and by primitive recursion.

EXERCISE 24

$\mathbb{N} \times \mathbb{N}$ is a rectype, with a founder $\langle 0, 0 \rangle$ and two constructors S -left and S -right.

What is primitive recursion on this datatype?

We define $\binom{n}{k}$ by $\binom{n}{0} = 1$ and $\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k}$.

Is this a primitive recursive declaration in your sense?

Some observations

- Make a mental note, for use later, that we will write ' \underline{n} ' for the string

$\overbrace{S(S(S \dots 0 \dots))}^{n \text{ times}}$ ' \underline{n} ' is thus a constant term not a variable.¹

- This rectype of functions is a rectype of **function declarations**, pieces of syntax. To be strictly correct one should think of them not as functions at all, but as *notations for functions*. If you want to think of them as functions (and people do, he acknowledges wearily) one has to think of them as functions-in-intension. We will consider a function(-in-extension) to be primitive recursive if it has a primitive recursive declaration (as a function-in-intension).

¹Syntax buffs might wish to think about the precise status of formulæ containing such terms. I suspect they may have the same status as expressions like $(\exists x_1 \dots x_n) \bigwedge_{i \neq j \leq n} (x_i \neq x_j)$. See the discussion on page 186

- In the recursion step for primitive recursion we find the line:

$$f(\vec{x}, y + 1) := h(\vec{x}, y, f(\vec{x}, y)).$$

One might wonder what the ‘ y ’ is doing in the *definiendum*². One of my students thought it was there to tell you how often you had been through the loop. It does do that, true, but that’s not all it does. For consider: if, for any primitive recursive function f , the value of $f(y + 1)$ depended only on $f(y)$ and not also on y , then if f ever took the same value twice³ it would be forced to be eventually periodic.

- Observe that in our projection functions (“pick the i th thing from these k things”) the ‘ i ’ and ‘ k ’ are concrete numerals. We have countably many functions not one. If the positions occupied by these numerals could be occupied by variables then the rule of substitution would allow us to write things like: “take the $f(i)$ th thing from this $g(n)$ -tuple” and for suitable choices of primitive recursive f and g this would be a primitive recursive function that might not be total. Worse still [for all i know] it could even be unsolvable whether functions declared in this way are total. Don’t go there.
- The composition operation under which the retype of primitive recursive function declarations is closed is slightly more complicated than the familiar composition of functions of one argument, simply because we are allowing functions of many variables. It is fiddly but it’s obvious in the sense that it is what you think it is. In some of the literature it is called *substitution* and it is worth noting though that, if $x, y \mapsto f(x, y)$ is a primitive recursive function of two variables, then $x \mapsto f(x, x)$ is a primitive recursive function of one variable.
- For any numeral \underline{n} , the function with constant value n is primitive recursive—compose $x \mapsto 0$ with successor n times.
- Notice that, although there is no limit on the number of variables we can compute with, we recurse on only *one*. On the face of it this declaration looks very restrictive: “only allowed one call”, but it turns out to be surprisingly fertile.
- Note at the outset that this datatype of function declarations is countably presented (see section ??) and so has only countably many elements.
- The basic functions are in some obscure but uncontroversial sense computable; clearly the composition of two computable functions is computable, and if g and h are in some sense computable, then f declared

²Latin: *definiendum* = the thing being defined; *definiens* = the thing doing the defining. Probably neo-latin not classical latin.

³Well, not quite, because if it’s only *twice* or k times for some concrete k that it takes the same value, then that fact can be hard-coded with lots of **if then else** commands but if it took the same value infinitely often. . .

over them by primitive recursion is going to be computable in the same sense. That is why this definition is *prima facie* at least a halfway sensible stab at a definition of computable function.

Here are some declarations:

DEFINITION 6

- (i) *Predecessor*: $P(0) := 0$; $P(S(x)) := x$.
- (ii) *Bounded subtraction*: $x \dot{-} 0 := x$; $x \dot{-} S(y) := P(x \dot{-} y)$.
- (iii) *Addition*: $x + 0 := x$; $x + S(y) := S(x + y)$.
- (iv) *Multiplication*: $x \cdot 0 := 0$; $x \cdot (S(y)) := (x \cdot y) + x$.

In (iii) we find the significance of the White Queen's claim that Alice can't do addition.

Consider the following sequence of binary functions $\mathbb{N}^2 \rightarrow \mathbb{N}$:

DEFINITION 7

$$\begin{aligned} f_0(m, k) &:= m + k; \\ f_{S(n)}(m, 0) &:= m; \\ f_{S(n)}(0, m) &= f_n(m, 1); \\ f_{n+1}(S(m), S(k)) &= f_n(m, f_{S(n)}(S(m), k)). \end{aligned}$$

EXERCISE 25 (*) Check that all the f_i with $i \in \mathbb{N}$ are primitive recursive.

This is from Doner-Tarski [19], who actually define this hierarchy of functions on *all* ordinals, not just finite ordinals, tho' of course they need some clauses to deal with limit ordinals. In fact one clause suffices:

DEFINITION 8

$$f_\gamma(\alpha, \beta) := \sum_{\eta < \beta, \zeta < \alpha} f_\zeta(f_\gamma(\alpha, \eta), \alpha).$$

[Observe that, for $\gamma \geq \omega$, all values of f_γ are infinite ordinals]

[Check this: i think the Doner-Tarski functions of finite subscript all restrict to total functions $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, whereas any D-T function with an infinite subscript sends at least some pairs of naturals to infinite ordinals.]

EXERCISE 26 What is the next function in the Doner-Tarski hierarchy after exponentiation? And the one after that?

EXERCISE 27 (*)

Show that, if f is primitive recursive, so are

1. the function $\sum_f(n) = \sum_{0 \leq x < n} f(x)$ that returns the sum of the first n values of f ; and
2. the function $\prod_f(n) = \prod_{0 \leq x < n} f(x)$ that returns the product of the first n values of f .

4.1.1 Some quite nasty functions are primitive recursive

EXERCISE 28 (For those who did *Graph Theory in Part II*)

You saw a proof of the finite version of Ramsey's theorem. Examine this proof and characterise the bounds it gives. Are these bounds described by a primitive recursive function?

Andrei sez: The Paris-Harrington function grows like F_{ϵ_0} , the ϵ_0 th Hardy function, which is far beyond primitive recursive.

The function for Paris-Harrington for pairs is ackermannian, so it is already beyond primitive recursive.

4.1.2 Justifying Circular Definitions

Recursive definitions are *prima facie* circular and therefore *prima facie* illegitimate. (Chapter 8 of [65] contains a beautiful discussion of the criteria a definition must meet if it is to be legitimate. It inspired an entire generation of logicians.) In section 4.1 we proved that, for example, every primitive recursive function is total, but in no case did we prove that there actually was a function answering to the circular definition. Not only do we have to do that, we also have to show, for any constraint (co's at this stage it is only a constraint not a definition) that any two functions answering to that constraint have the same graph.

So we have to prove existence and uniqueness. There are several ways to do it.

Consider everybody's favourite example of a recursively defined function:

$$\mathbf{fact}(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot \mathbf{fact}(n-1)$$

It is circular, since the *definiendum* appears inside the *definiens*.

One thing we can do immediately is define each restriction $\mathbf{fact} \upharpoonright [0, k]$. That's easy. The hard part is to glue them together. For each k we can define a function $\mathbf{fact}\text{-}k$ which is $\mathbf{fact} \upharpoonright [0, k]$. We show that any two functions in $\{\mathbf{fact}\text{-}k : k \in \mathbb{N}\}$ agree wherever they are both defined, so there is a good notion of *limit* of the family—and that limit is what we want. There is no problem in showing that the limit is unique, and the obvious strategy for doing this merely writes out in detail the proof of a particular instance of a fixed-point theorem. We could even spell out the fixed-point theorem. The connections between fixed point theorems as recursive datatypes was set out in the previous chapter. We will return to this theme in chapter 6.

Rewrite this section define \upharpoonright

The other thing we can do is outline a general strategy for transforming a circular recursive definition into a direct non-recursive definition. There is some heavy machinery to hand that will do the work for us immediately: the graph of any primitive recursive function is an inductively defined set, so we can define—for example—the graph of the factorial function as:

$$\bigcap \{Y \subseteq \mathbb{N} \times \mathbb{N} : \langle 0, 1 \rangle \in Y \wedge (\forall u, v)(\langle u, v \rangle \in Y \rightarrow \langle u+1, (u+1) \cdot v \rangle \in Y)\}$$

This is noncircular (at least it will be once we have a noncircular definition for multiplication(!)) but it is logically expensive, since it uses a higher-order quantifier, which makes it \forall_1^2 . And we can do much better than that.

Eliminating the circularities in a first-order way

We will need the concept of a certificate or proof from section 3.12.

Suppose we have defined f by primitive recursion:

$$\begin{aligned} f(0, \vec{s}) &:= g(\vec{s}); \\ f(S(n), \vec{s}) &:= h(f(n, \vec{s}), n, \vec{s}). \end{aligned}$$

This declaration can be thought of as a definition of a rectype of tuples, to wit: the graph of f . The founders of this rectype are the tuples $\langle 0, \vec{s}, g(\vec{s}) \rangle$; the constructor is the operation that takes a tuple $\langle n, \vec{s}, k \rangle$ and returns the tuple $\langle n+1, \vec{s}, h(k, n, \vec{s}) \rangle$. Thus $y = f(x, \vec{s})$ iff the tuple $\langle x, \vec{s}, y \rangle$ belongs to the rectype of the preceding paragraph, and if it does there will be a certificate to that effect. It turns out that we can assert the existence of such a certificate in a noncircular way.

What are these certificates? Actually it won't matter much precisely how we think of them. A certificate that $f(S(n), \vec{s}) = x$ could be the ordered pair of a-certificate-that- $f(S(n), \vec{s}) = y$ -for-some- y with a-certificate-that- $h(y, n, \vec{s}) = x$, but in practice we can get away with taking the certificate that $f(S(n), \vec{s}) = x$ to be the list of all pairs $\langle i, \vec{s}, f(i, \vec{s}) \rangle$ with $0 \leq i \leq n$, and that is what we will in fact do.

Now that we have decided that a certificate is a list, we have to explain how to code up lists (as well as the things listed) as numbers, so that the existence of a certificate turns out to be assertable in the language of arithmetic.

We can encode sequences of natural numbers as natural numbers using the *prime powers trick*, which we see on page 74.

The prime powers trick enables us to prove the following:

REMARK 4 *If f is a function $\mathbb{N}^k \rightarrow \mathbb{N}$ declared by primitive recursion then there is a formula $\phi(y, x_1 \dots x_k, \vec{z})$ in the language with $0, 1, +, \times$, exponentiation and $=$ containing no unrestricted quantifiers such that $y = f(x_1 \dots x_k)$ iff $(\exists z_1 \dots z_n) \phi(y, x_1 \dots x_k, \vec{z})$*

This works because by means of the prime powers trick we can encode finite sequences from \mathbb{N} as naturals, so we can encode certificates. We observed that $y = f(x, \vec{s})$ holds iff there is a certificate to that effect. So we need to be able to express “ C is a certificate that $y = f(x, \vec{s})$ ”. To do that we need to be able to code up lists of ordered pairs as natural numbers, and it is for this that we can use the prime powers trick. I won't go into the details because we are going to prove something rather stronger, namely that we can achieve the results of remark 4 even with the extra restriction of not using exponentiation.

THEOREM 5 *If f is a function $\mathbb{N}^k \rightarrow \mathbb{N}$ declared by primitive recursion then there is a formula $\phi(y, x_1 \dots x_k, \vec{z})$ in the language with $0, 1, +, \times, <$ and $=$ (“the language of ordered rings”) containing no unrestricted quantifiers such that $y = f(x_1 \dots x_k)$ iff $(\exists \vec{z}) \phi(y, \vec{x}, \vec{z})$*

(In fact—and this is a famous theorem of Davis, Putnam, Robinson and Matiyasievich⁴—we can even find a ϕ that contains no quantifiers at all, not even restricted quantifiers. I doubt if i will get round to proving it.)

Proof:

We will be using base- p representations of arbitrary numbers, and we will need to know that there are arbitrarily large primes. Well, there just *are* arbitrarily large primes, and we appeal to their existence when we want to establish the correctness of the recursive definition. The theorem we are trying to prove—that a function defined by primitive recursion can be captured by an \exists_1 formula in the language of ring theory—is a metatheorem *about* the language of ring theory, not a theorem *of* ring theory. So we don't need to worry about whether or not we can prove the infinitude of primes in ring theory.

Anyway, fix values for ' x ' (' x ' is the variable on which we are recursing) \vec{s} (the \vec{s} are the snail variables) and ' y '.

It is clear that the ring language can express " p is a prime" and " z is a power of p " and these will give us all the freedom in manipulating base- p representations that we need.

There is going to be a large number I and another large number O ("inputs" and "outputs"), encoding somehow the inputs (the list of naturals less than x) and a list of outputs (the corresponding values of f), and we are going to think of these two numbers as being written in base p where p is going to be a prime larger than any number that appears anywhere in the certificate. Thus our formula will begin with three existential quantifiers: ' $(\exists I)(\exists O)(\exists p)(\dots)$ '. The prime p will be chosen big enough so that the following picture makes sense.

| ... | p^2 | p^1 | p^0 | |
|-----|--------|--------|--------|-----|
| ... | 2 | 1 | 0 | I |
| ... | $f(2)$ | $f(1)$ | $f(0)$ | O |

We have to be very careful in talking about base- p representations of numbers in this context where we have neither exponentiation nor order information. (The display above is potentially very misleading!) One way of describing our predicament is that we normally think of the addresses in the base- p representation of a number as indexed by an ordered set that is a proper initial segment of $\langle \mathbb{N}, < \rangle$ —but we cannot use that index set here. Our places are indexed by a set X of numbers about which we know only that all its members are powers of p and that X contains all factors of its members. It is true that we can define an order relation on X and we do have an adjacency relation on X , since we can divide by p or multiply by p . However we do not have access to any bijection between X and any initial segment of \mathbb{N} . In particular, although we can identify a column in the above display by reference to a z -that-is-a-power-of- p , we cannot recover the exponent and thereby enumerate the columns.

Let's get some definitions out of the way.

⁴I would like to lecture it but it involves a bit *tooo* much number theory—for me at least.

Some Local Definitions

“ x divides into y ” is $x = y \vee (\exists w < y)(x \cdot w = y)$. Let’s write this as $x|y$.

$x \text{ DIV } y$ is the largest integer z s.t. $y \cdot z \leq x$ and $x \text{ rem } y$ is the remainder when x is divided by y .

Strictly speaking⁵ this is naughty, because introducing new terms like this expands the language and takes us out of the language of ordered rings; we should say instead

“ $z = x \text{ DIV } y$ iff $x \cdot z \leq y < x \cdot (z + 1)$ ” and

“ $w = x \text{ rem } y$ iff $(\forall z < x)(z = (x \text{ DIV } y) \rightarrow z \cdot y + w = x)$ ”.

Then, when we want to say $\phi(x \text{ DIV } y)$ we can write either

$$(\forall z)(x \cdot z \leq y < x \cdot (z + 1) \rightarrow \phi(z))$$

or

$$(\exists z)(x \cdot z \leq y < x \cdot (z + 1) \wedge \phi(z))$$

...depending on whether we want the quantifier to be ‘ \exists ’ or ‘ \forall ’ (which will in turn depend on whether the occurrence of $\phi(x \text{ DIV } y)$ is negative or positive).

We can say “ p is a prime” since that is $(\forall x < p)(\forall y < p)(x \cdot y \neq p)$.

We can capture “ z is a power of p ” by $(\forall w < z)(w|z \rightarrow p|w)$ —at least when p is prime. (And the task in hand will not require us to capture “ z is a power of p ” when p is not prime.)

We can express in the ring language what it is for a natural number O to have the entry o_z at the place in its base- p representation corresponding to z (where z is a power-of- p). We say:

“If we divide O by z and look at the remainder⁶ then divide that remainder by (z/p) , we find that the quotient is o_z .”

In symbols:

$$(O \text{ rem } z) \text{ DIV } (z/p) = o_z. \tag{R}$$

Jack Webster says ...

I might be wrong here, but I think there is a small mistake in a formula there (not that the actual formula is important):

That is, $(O \text{ rem } z) \text{ div } (z/p) = o_z$. Say $O = a + bp + cp^2 + dp^3$ and we take $z = p^2$. Then $(O \text{ rem } p^2) \text{ div } p = (a + bp)/p = b$, but we want c .

I think $(O \text{ div } z) \text{ rem } p$ works though. $(O \text{ div } p^2) \text{ rem } p = (c + dp) \text{ rem } p = c$. Alternatively $(O \text{ rem } pz) \text{ div } z$ does it too I think.

⁵Thank you, David Edey!

⁶which of course is just the truncation of O , the places remaining to the right of the place corresponding to z .

Let us abbreviate (R) to ' $R(O, z, o_z)$ ', and let us write

' i_z ' for the I -entry at the place corresponding to z (i.e., the unique i such that $R(I, z, i)$, namely $I \text{ rem } z \text{ DIV } (z/p)$);

and

' o_z ' for the O -entry at the place corresponding to z (i.e., the unique o such that $R(O, z, o)$, namely $(O \text{ rem } z) \text{ DIV } (z/p)$).

How do we tie together I and O ? We have to say several things:

- (1) For any $z < I$ that is a power-of- p , $\langle i_z, o_z \rangle$ is related-by-the-recursion-for- f to $\langle i_{(z/p)}, o_{(z/p)} \rangle$. We declared f by $f(n+1, \vec{s}) = g(f(n), n, \vec{s})$ so this is $o_z = g(o_{(z/p)}, i_{(z/p)}, \vec{s})$;
- (2) Initialising: we have to say $i_1 = 0$ and $o_1 = f(0, \vec{s})$;
- (3) The n th place of I is n , thus: $i_z = i_{(z/p)} + 1$;
- (4) And of course we have to say $(\exists z)(x = i_z \wedge y = o_z)$.

So our first order formula will be

$$(\exists I)(\exists O)(\exists p) \bigwedge \left(\begin{array}{l} p \text{ is prime} \\ (\exists z < I)((z \text{ is a power of } p \wedge y = o_z \wedge x = i_z) \\ i_1 = 0 \wedge o_1 = f(0, \vec{s}) \\ (\forall z < I)(z \text{ is a power of } p \rightarrow i_z = i_{(z/p)} + 1) \\ (\forall z < I)(z \text{ is a power of } p \rightarrow o_z = g(o_{(z/p)}, i_{(z/p)}, \vec{s})) \end{array} \right) \quad (\text{A})$$

Some clarifying observations

- How many powers of p are we interested in? Well, obviously the first x of them. Clearly if we take p to be the least prime bigger than any $f(n, \vec{s})$ for $n \leq x$ (the \vec{s} are fixed, remember) then the powers of p that are of interest (the “columns”) are precisely those powers of p that are less than $I = 1 \cdot p + 2 \cdot p^2 + \dots$. This explains the bound “ $< I$ ” whenever we quantify over powers of p .
- Why does the base for the representation of I and O have to be a *prime*? Base-ten representations have served us well enough. The answer is that we need to be able to identify powers of the base, and (as we saw above) it is easy to express “ z is a power of p ” in the ring language if p is a prime; not so easy if p is composite ... but then we don't need to!
- The last line in formula (A) above contains the function letter ‘ g ’ which is assumed to call a primitive recursive function. This is the clause that requires us to do some work in the proof by structural induction that ‘ $y = f(x, \vec{s})$ ’ can be captured by a \exists_1 expression when f is primitive recursive. By induction hypothesis ‘ $o_z = g(o_{(z/p)}, i_{(z/p)}, \vec{s})$ ’ is equivalent to an \exists_1 expression—because g is primitive recursive. We can pull the existential quantifiers to the front by appeal to remark 3.

- We have considered only the induction step concerning the constructor of primitive recursion, not composition. And we haven't considered the founder functions. But all that is easy.

■

Justification of Recursion, and Beth's theorem

We have considered the question of the existence and uniqueness of solutions to recursions; there is also the question of whether or not these solutions can be defined in the original language.

There is a theorem of Beth's here which seems to be relevant.

THEOREM 6 (*quoted from [31] p 301.*)

Let $L \subseteq L^+$ be first-order languages; let T be a theory in L^+ and $\phi(\vec{x})$ a formula of L^+ . Then the following are equivalent:

1. *If \mathfrak{A} and \mathfrak{B} are models of T and $\mathfrak{A} \restriction L = \mathfrak{B} \restriction L$ then, for all tuples \vec{a} in \mathfrak{A} , $\mathfrak{A} \models \phi(\vec{a})$ iff $\mathfrak{B} \models \phi(\vec{a})$;*
2. *$\phi(\vec{x})$ is equivalent modulo T to a formula $\psi(\vec{x})$ of L .*

Why can we not just use theorem 6 to say that (for example) the $+$ function is explicitly definable in the language with 0 and S since it is implicitly definable? We will show that $+$ cannot be explicitly defined in the language with just 0 and S.

Consider the theory T of zero and successor, plus a scheme of induction. We can define $+$ by recursion, and we can prove in the theory in the expanded language (with ' $+$ ' and ' $+'$ ') that if $+$ and $+'$ both obey the recursive definition then they are coextensive. We do this by induction, of course. (This sexed-up theory is known as *Presburger Arithmetic*) This does not imply that we can define $+$ in the language of 0 and S—and in fact it's known that we cannot. (why? How?)

However we cannot obtain a contradiction by applying Beth definability, because the theory of 0 and S does not *implicitly define* $+$ within the meaning of the act. For the theory T of 0 and S to implicitly define $+$ it would have to be the case that, for any model \mathfrak{M} of that theory, there is precisely one way of decorating it with a ternary predicate to obtain a model of Presburger arithmetic, and we shall now show that this is not so.

We have a model for the theory T of 0 and S plus induction that looks like \mathbb{N} plus lots of copies of \mathbb{Z} . It's totally ordered. Our model of $\text{Th}(0, S)$ -plus-induction which looks like $\mathbb{N} + \mathbb{Z} \times \mathbb{Q}$. (Question 11 of Sheet 3, Part II Set Theory and Logic 2015). Think of elements of your model as either naturals (we ignore them) or ordered pairs $\langle q, z \rangle$ of a rational and an integer. Now if \oplus is a binary operation on \mathbb{Q} that obeys the axioms for $+$ we can define a $+$ operation on our model by $\langle q_1, z_1 \rangle + \langle q_2, z_2 \rangle =: \langle (q_1 \oplus q_2), z_1 + z_2 \rangle$. $S(\langle q, z \rangle)$ is

of course $\langle q, z + 1 \rangle$. So all we have to do is show how to define more than one $+$ operation on \mathbb{Q} . Easy: move the origin! Let q an arbitrary rational; define $x \oplus y$ as $(x - q) + (y - q)$. This is clearly commutative and associative.

4.2 Exercises

(Both from a Part II sheet of PTJ's years ago)

EXERCISE 29 (*)

For each of the following functions $\Phi : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$, determine (a) whether Φ is order-preserving, and (b) whether or not it has a fixed point:

- (i) $\Phi(f)(n) = f(n) + 1$ if $f(n)$ is defined, undefined otherwise.
- (ii) $\Phi(f)(n) = f(n) + 1$ if $f(n)$ is defined, $\Phi(f)(n) = 0$ otherwise.
- (iii) $\Phi(f)(n) = f(n - 1) + 1$ if $f(n - 1)$ is defined, $\Phi(f)(n) = 0$ otherwise.

EXERCISE 30 (*)

(i) For partial functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$, define $d(f, g) = 2^{-n}$ if n is the least number such that $f(n) \neq g(n)$, and $d(f, g) = 0$ if $f = g$. [The inequality $f(n) \neq g(n)$ is understood to include the case where one side is defined and the other is not.] Show that d is a metric, and that it makes $[\mathbb{N} \rightarrow \mathbb{N}]$ into a complete metric space.

(ii) Show that the function Φ which corresponds to the recursive definition of the factorial function is a contraction mapping for the metric d , and hence obtain another proof that it has a unique fixed point.

(iii) [if you know what a contraction mapping is] Which (if any) of the functions defined in 29 are contraction mappings?

4.2.1 Primitive Recursive Relations

A relation is primitive recursive if the characteristic function of its graph is primitive recursive⁷ [we haven't defined characteristic function yet]

A relation-in-extension $R(\vec{x})$ is a **primitive recursive relation** (or predicate, the words are used interchangeably) iff there is a primitive recursive function $r : \mathbb{N}^\rho \rightarrow \{0, 1\}$ (ρ is the arity of R) such that $r(\vec{x}) = 0 \iff R(\vec{x})$. That is to say, an n -ary relation is primitive recursive iff the characteristic function of its graph is primitive recursive. Of course we can also talk of subsets of \mathbb{N}^k as being primitive recursive.

In the above setting we say that r **represents** R . We can take 1 to be **true** and 0 to be **false**, or vice versa, or 0 to be **true** and all other values to be **false**—it does not matter which way one jumps as long as one is consistent. In what follows **true** is 0, and **false** is 1. Other naturals don't get used for this purpose.

⁷Strictly: there is a primitive recursive function with the same graph as the characteristic function

EXERCISE 31 (*)

Show that $<_{\mathbb{N}}$ is a primitive recursive relation;

Show that $\leq_{\mathbb{N}}$ is a primitive recursive relation;

Show that $=$ is a primitive recursive relation.

The family of primitive recursive relations is closed under lots of operations.

Boolean Operations

We observe that \emptyset and \mathbb{N}^k have primitive recursive characteristic functions (as do all finite and cofinite subsets of \mathbb{N}^k).

If R and S are primitive recursive predicates represented by r and s , then

$R \vee S$ is represented by $r \cdot s$;

$R \wedge S$ is represented by $r + s$;

$\neg R$ is represented by $1 \dot{-} r$;

so boolean combinations of primitive recursive relations are primitive recursive.

Relational Algebra

Converse of a primitive recursive relation is primitive recursive.

What about composition of primitive recursive relations? We will see later (exercise 61) that relational composition (as in: nephew-of is *sibling-of* composed with *son-of*) of primitive recursive relations might not preserve primitive recursiveness.

Transitive closures? Presumably not

Substitution

If $R(\vec{x})$ is a primitive recursive relation then we can substitute terms $g(\vec{y})$ for the x s as long as the g are primrec. (use composition/substitution). So “ $x = f(\vec{y})$ ” is a primitive recursive relation if f is primitive recursive. [substitution performed on ‘ $a = b$ ’].

Bounded Quantification

If $R(x, \vec{y})$ is represented by $r(x, \vec{y})$ then $(\exists x \leq z)(R(x, \vec{y}))$ is represented by

$$\prod_{0 \leq x \leq z} r(x, \vec{y}).$$

We can capture bounded universal quantification by exploiting duality of the quantifiers, so $(\forall x \leq z)(R(x, \vec{y}))$ is represented by

$$1 \dot{-} \left(\prod_{0 \leq x \leq z} (1 \dot{-} (r(x, \vec{y}))) \right)$$

If-then-else

The set of primitive recursive functions is also closed under **if then else**, in the sense that if r is a primitive recursive predicate, then **if R then x else y** is also primitive recursive. Here's why. Declare:

$$\text{if-then-else}(0, x, y) := x; \text{ if-then-else}(S(n), x, y) := y.$$

if-then-else is evidently primitive recursive (and in fact it's so primitive that it doesn't actually involve any recursion at all!) and it is mechanical to check that

$$\text{if-then-else}(\text{proj}(r, x, y)_1^3, \text{proj}(r, x, y)_2^3, \text{proj}(r, x, y)_3^3)$$

evaluates to x if $r = 1$ and to y if $r = 0$.

Putting this together with the fact that bounded quantification is primitive recursive tells us that

THEOREM 7 *Functions declared in the style*

$$\text{if } (\exists x < y) R(x, \vec{z}) \text{ then } f(y, \vec{z}) \text{ else } g(y, \vec{z}).$$

are primitive recursive, as long as R , f and g are. ■

This is **bounded search**. Hofstadter [32] memorably calls this “BLOOP”.

We will use the string ‘**pair**’ to represent a primitive recursive bijection $\mathbb{N}^2 \rightarrow \mathbb{N}$. The following is a standard example:

$$\text{pair}(x, y) = \frac{(x+y) \cdot (x+y+1)}{2} + x$$

and **fst** and **snd** are the corresponding primitive recursive unpairing functions, so that

$$\begin{aligned} \text{fst}(\text{pair}(m, n)) &= m, \\ \text{snd}(\text{pair}(m, n)) &= n \text{ and} \\ \text{pair}(\text{fst}(r), \text{snd}(r)) &= r. \end{aligned}$$

EXERCISE 32 (*) *Check that **pair** is a bijection between $\mathbb{N} \times \mathbb{N}$ and \mathbb{N} , and show that it and the unpairing functions **fst** and **snd** are all primitive recursive*⁸.

In future when we write ‘**pair**(x, y)’ without comment we shall assume it is this function we are using. ‘ $\langle n, m \rangle$ ’ will denote a primitive (anonymous) pairing function.

Except perhaps when we are doing λ -calculus. Check.

We observe without proof that the graph of a primitive recursive function is a retype; so too is the graph of a primitive recursive relation. For example, the

⁸Observe that **pair** is a bijection $\mathbb{N}^2 \leftrightarrow \mathbb{N}$ that is a polynomial with rational coefficients. Can you find a bijection $\mathbb{Q}^2 \leftrightarrow \mathbb{Q}$ that is a polynomial with rational coefficients? Can you find a bijection $\mathbb{Z}^2 \leftrightarrow \mathbb{Z}$ that is a polynomial with rational coefficients?

graph of the primitive recursive relation $\leq_{\mathbb{N}}$ has all the pairs $\langle 0, n \rangle$ (for $n \in \mathbb{N}$) as founders, and has the single constructor $\langle n, m \rangle \mapsto \langle S(n), S(m) \rangle$.

We shall see in section ?? how evaluation sequences, where the value y of a recursive function f at some argument x is computed by unravelling the recursion, correspond pretty exactly to a certificate that the pair $\langle y, x \rangle$ belongs to the rectype that is the graph of f .

EXERCISE 33 (*) Euler's totient function ϕ is defined by

$$\phi(n) := |\{m < n : HCF(m, n) = 1\}|.$$

Prove that ϕ is primitive recursive.

EXERCISE 34

Show that, if R is a primitive recursive predicate, then the function sending n to the least $y < k$ such that $R(n, y, \vec{z})$ is also primitive recursive.

We will see later that we really do need the bound if we wish to secure primitive recursiveness. See exercise 61 p 95.

EXERCISE 35 (*)

1. The declaration:

```
Fib(0) := 1;
Fib(1) := 1;
Fib(n + 2) := Fib(n + 1) + Fib(n);
```

is not primitive recursive. Find a declaration of this function-in-extension that is primitive recursive.

2. The **iterate** $\text{It}(f)$ of f is defined by: $\text{It}(f)(m, n) = f^m(n)$. Notice that, even if f is a primitive recursive function of one argument, this function of two arguments is not *prima facie* primitive recursive. Show that it is primitive recursive nevertheless.

Take \mathcal{I} to be the inductively defined class of functions containing the successor function $S(n) = n + 1$, the functions **pair**, **fst**, **snd** and closed under composition and iteration. Show that if $a \in \mathbb{N}$ and $G(x, y)$ is in \mathcal{I} and $H(x)$ is defined by $H(0) = a$, $H(n + 1) = G(H(n), n)$, then $H(x)$ is in \mathcal{I} . [Hint: Consider **pair**($H(y), y$).]

EXERCISE 36 Show that all primitive recursive functions are total by structural induction on the rectype. The induction step for primitive recursion uses induction over \mathbb{N} .

This means that functions like the one that returns n when given $2n$ and fails on odd numbers are not primitive recursive. Nevertheless, you will often hear people say—as I say to you now—that you would be extremely unlucky to

encounter computable functions that are not primitive recursive unless you are a logician and go out of your way to look for trouble. The Ramsey functions (some of them, at least) are primitive recursive. (That was exercise 28 on page 45). Waring's g and G are not—at least they aren't defined that way as functions-in-intension—but it turns out after all that they are too...in the sense that the graphs of G and g are also the graphs of primitive recursive functions-in-intension.

The resolution of this apparent contradiction is that the function $n \mapsto (\text{if } n = 2k \text{ then } k \text{ else fail})$ is in some sense *coded* by the primitive recursive function that sends $2n + 1$ to 0 (meaning **fail**) and sends $2n$ to $n + 1$ (meaning n), and this function is primitive recursive.

So there is a way of thinking of the Möbius function as primitive recursive.

EXERCISE 37 *The Möbius function μ is defined by*

$$\mu(n) := \text{if } n \text{ is not square-free then } 0 \text{ else } (-1)^k$$

*where k is the number of distinct prime factors of n .*⁹

Prove that μ is primitive recursive. (You will somehow have to code up the negative integer -1 .)

4.2.2 Simultaneous Recursion

A **simultaneous recursion** or **mutual recursion** is where two or more functions are declared by recursions in which each calls some of the others as well as possibly itself.

The usual example is the **odd** and **even** functions, which represent the set $\{2n : n \in \mathbb{N}\}$ of even naturals and the set of odd naturals respectively. (Minexercise: supply this definition.) Here is another example—from a 1a Computer Science exam of some years ago.

$$\begin{aligned} f(n) &:= \text{if } n = 0 \text{ then } 0 \text{ else } g(f(n-1) + 1, 1) - 1; \\ g(n, m) &= f(f(n-1)) + m + 1. \end{aligned}$$

(It turns out that $f(n) = n$ and $g(m, n) = m + n$.)

EXERCISE 38 *On the face of it a simultaneous declaration like that of **even** and **odd** is not primitive recursive. Use the pipelining technique above to show how, nevertheless, any function that is declared in such a bundle can be given a declaration as a primitive recursive function.*

Enthusiasts might like to try to prove the following theorem of Rozsa Péter which I found in [53] p 12.

⁹Be warned that we will later use the letter ' μ ' for the function that returns the least object in a set ("μimum" not μöbius")

EXERCISE 39 Suppose functions g , h , and the j_i for $0 < i \leq k$ are primitive recursive, with $(\forall x)(j_i(x) \leq x)$ for every $i \leq k$, and that f is defined by

$$f(0, y) := g(y)$$

$$f(x + 1, y) := h(x, y, f(j_1(x), y), \dots, f(j_k(x), y)).$$

Show that f is primitive recursive too.

We will later need the fact that the function enumerating the primes in increasing order is primitive recursive. For this we need that a search $(\exists x < f(n))(\dots)$ is primitive recursive if f and the dots are primitive recursive. Bertrand's postulate might come in handy.

Where do we prove this?

EXERCISE 40 Show that the function $\pi(n) =$ the n th prime is primitive recursive

4.3 μ -recursion

Does the retype of primitive recursive functions exhaust the class of (total) functions that reasonable people would consider computable?

There are some functions that are clearly not everywhere defined but are equally clearly in some sense computable: $n \mapsto n/2$ if n is even and **fail** otherwise. We know that every primitive recursive function is everywhere defined, so does it follow that not every computable function is primitive recursive? Well no, not really, because we can encode this partial function by the total function $n \mapsto (n/2) + 1$ if n is even and 0 otherwise. If we want to demonstrate that there are computable functions that are not primitive recursive we have to do a bit more work, and that is where the Ackermann function comes in.

4.3.1 The Ackermann function

The following function:

$$\begin{aligned} A(0, n) &:= S(n) \\ A((S(m)), 0) &:= A(m, 1) \\ A(S(m), S(n)) &:= A(m, A(S(m), n)) \end{aligned}$$

is the *Ackermann function*. Brief inspection will reveal that this declaration is not primitive recursive. This function—unlike the Fibonacci function and the simultaneous recursion cases we saw—doesn't seem to have a primitive recursive declaration at all. We shall in due course establish that this is, indeed, the case. That is where the significance of the Ackermann function lies: it is a kosher recursive function—provably defined everywhere—that nevertheless has no primitive recursive declaration. As such it torpedoes the project to capture all computable functions by means of primitive recursion. So we have to establish that (i) it is total and (ii) has no primitive recursive declaration.

THEOREM 8 $A(n, m)$ is defined for all $n, m \in \mathbb{N}$.

Proof:

By induction on the lexicographic order of $\mathbb{N} \times \mathbb{N}$. (You proved in exercise 20 that a lexicographic product of finitely many wellfounded strict partial orderings is a wellfounded strict partial ordering).

Assume $A(x, y)$ is defined for all pairs $\langle x, y \rangle$ that precede $\langle n, m \rangle$ in the lexicographic ordering. One of the three possibilities below must happen:

1. $n = 0$. In this case $A(n, m) = m + 1$;
2. $m = 0$. Then $A(n, m) = A(n - 1, 1)$ which is defined by induction hypothesis, since $\langle n - 1, 1 \rangle <_{lex} \langle n, m \rangle$;
3. $n, m \neq 0$. Then $A(n, m)$ is $A(n - 1, A(n, m - 1))$. Now $\langle n - 1, z \rangle <_{lex} \langle n, m \rangle$ for all z . So $A(n, m)$ is defined as long as $A(n, m - 1)$ is defined, because this enables us to take z to be $A(n, m - 1)$. But $\langle n, m - 1 \rangle <_{lex} \langle n, m \rangle$, so by induction hypothesis $A(n, m - 1)$ is defined and can be taken to be one such z .

■

Here is another proof of theorem 8 this time by a double induction.

We prove by induction on n that $(\forall m)(A(n, m) \text{ is defined})$, and the induction step requires an induction on m .

Base case: $n = 0$. $A(0, m) := m + 1$.

Induction step:

Now assume $A(n, m)$ is defined for all m . We will prove that $A(n + 1, m)$ is defined for all m , and we will do this by induction on ' m '.

Base case:

$m = 0$. $A(n + 1, 0) := A(n, 1)$ by stipulation, and $A(n, 1)$ is defined by induction hypothesis.

Induction step:

So assume $A(n + 1, m)$ defined. We wish to be reassured that $A(n + 1, m + 1)$ is defined as well. The definition stipulates that $A(n + 1, m + 1) := A(n, A(n + 1, m))$, and by induction hypothesis (on ' m ', in the inner loop) $A(n + 1, m)$ is defined, and by induction hypothesis (on ' n ', in the outer loop) $A(n, A(n + 1, m))$ is defined.

■

The more jaded among you may feel that these two proofs are the same proof underneath. Perhaps they are. At any rate what we are seeing here is the simplest possible illustration that a total function might be proved total by an

induction over a lexicographic product of length ω^α for some α , or by induction over \mathbb{N} using nested loops.¹⁰ If we need to do an induction over a wellorder of length ω^2 we need (look at the exponent) *two* nested inductions.

Perhaps the best way to emphasise this point is to prove that the Ackermann function *dominates* all primitive recursive functions.

We need some technical details. They are not hard enough to justify being lectured, but they do matter enough to be worth doing as an exercise. (Do the various parts of the exercise in the order indicated)

EXERCISE 41 (*) *Prove the following:*

- (1) $(\forall m, n)(A(m, n) > n)$;
- (2) *A is strictly monotone increasing in its second argument;*
- (3) $A(m + 1, n) \geq A(m, n + 1)$;
- (4) *A is monotone increasing in its first argument;*
- (5) $A(m, 2n) < A(m + 2, n)$.

DEFINITION 9

$f : \mathbb{N} \rightarrow \mathbb{N}$ **dominates** $g : \mathbb{N} \rightarrow \mathbb{N}$ if $(\exists n \in \mathbb{N})(\forall m > n)(f(m) > g(m))$.

Actually, what we are about to prove does not use this definition exactly, but it has the same flavour.

THEOREM 9 *For every primitive recursive function f there is a constant c_f such that*

$$(\forall \vec{x})(f(\vec{x}) < A(c_f, \max \vec{x})).$$

(We say c_f is **suitable** for f .)

Proof: We prove this by structural induction on primitive recursive functions.

It's easy to see that the theorem holds for $f = S$ (the successor function), f the 0 function, and f a projection.

Composition.

Suppose that the hypothesis is true for primitive recursive functions f_1, \dots, f_n , g and that g is (post-)composable with (f_1, \dots, f_n) . We will show that the hypothesis holds for $g(f_1(-), \dots, f_n(-))$.

Write $c_{f_1}, \dots, c_{f_n}, c_g$ for the constants suitable for f_1, \dots, f_n, g . Defining $m \max\{c_{f_1}, \dots, c_{f_n}, c_g\}$, we will show that $m+2$ is suitable for $g(f_1(-), \dots, f_n(-))$.

Let \vec{x} be a member of the domain common to the f_i . Renumbering if necessary, we may assume that $f_1(\vec{x}) = \max_i f_i(\vec{x})$. We have the following inequalities:

$$\begin{aligned} g(f_1(\vec{x}), \dots, f_n(\vec{x})) &< A(c_g, f_1(\vec{x})) && \text{definition of } c_g \\ &< A(c_g, A(c_{f_1}, \max \vec{x})) && \text{definition of } c_{f_1} \\ &\leq A(m, A(m, \max \vec{x})) && \text{definition of } m \end{aligned}$$

¹⁰The Ackermann function looks a bit like an attempt to define a fixed point for the Doner-Tarski operation DT that we saw earlier: definition 7 p. 44. Is there anything sensible one can say about this?

$$\begin{aligned}
&< A(m, A(m+1, \max \vec{x})) && \text{mono in both args} \\
&\leq A(m, A(m+2, \max \vec{x} - 1)) && \text{Ex 41 part (iii)} \\
&< A(m+1, A(m+2, \max \vec{x} - 1)) && \text{mono in first arg} \\
&= A(m+2, \max \vec{x}).
\end{aligned}$$

Therefore $m+2$ is suitable for $g(f_1(-), \dots, f_n(-))$.

You might be worried, Dear Reader, by the thought that the renumbering that ensures that it is f_1 that gives the biggest input to g depends on the choice of \vec{x} . It does, but this affects only the *second* argument to the Ackermann function in what follows, whereas it is the *first* argument that matters.

Primitive Recursion.

Suppose our hypothesis holds for g and h and that f is declared by primitive recursion over g and h ; that is, f is defined by:

$$\begin{aligned}
f(\vec{x}, 0) &= g(\vec{x}) \\
f(\vec{x}, y+1) &= h(\vec{x}, y, f(\vec{x}, y)).
\end{aligned}$$

Let c_g and c_h be constants suitable for g and h ; put $m = \max\{c_g, c_h\} + 1$. We will show by induction on y that, for every \vec{x} , $f(\vec{x}, y) < A(m, \max \vec{x} + y)$. This is clearly true for $y = 0$:

$$f(\vec{x}, 0) = g(\vec{x}) < A(c_g, \max \vec{x}) < A(m, \max \vec{x}).$$

Suppose that the assertion holds for $y \geq 0$. For every \vec{x} we have, using the induction hypothesis and basic properties of A ,

$$\begin{aligned}
f(\vec{x}, y+1) &= h(\vec{x}, y, f(\vec{x}, y)) \\
&< A(c_h, \max\{\max \vec{x}, y, f(\vec{x}, y)\}) \\
&< A(c_h, A(m, \max \vec{x} + y)) \\
&\leq A(m-1, A(m, \max \vec{x} + y)) \\
&= A(m, \max \vec{x} + y + 1).
\end{aligned}$$

This completes the induction. Writing x for $\max \vec{x}$, we have the inequality

$$f(\vec{x}, y) < A(m, x + y) \leq A(m, 2 \max\{x, y\}) < A(m+2, \max\{x, y\}).$$

(The final inequality follows from part (5) of exercise 41) Therefore $m+2$ is suitable for f . This completes the proof. \blacksquare

What this is telling us is that the ‘slices’ of the Ackermann function—that is to say the functions $\lambda n. A(m, n)$ —form an increasing ω -sequence of elements of the poset $\mathbb{N} \rightarrow \mathbb{N}$ ordered by dominance and that this sequence is cofinal in the primitive recursive functions.

COROLLARY 1 *The Ackermann function A is not primitive recursive.*

Proof: Suppose A is primitive recursive. Then $a : n \mapsto A(n, n)$ is also primitive recursive, so there is a constant c_a such that $A(c_a, n) > a(n)$ for every n . But this is definitely false for $n = c_a + 1$:

$$A(c_a, n) = A(c_a, c_a + 1) < A(c_a + 1, c_a + 1) = a(n).$$

■

You might think, Dear Reader, that this means there is a quantifier-pushing theorem along the lines of

$$(\forall x < y)(\exists n)\phi(n, x, y, \vec{w}) \longleftrightarrow (\exists a)(a = A(c_\phi, \max(y, x, \vec{w})) \wedge (\forall x < y)(\exists n < a)\phi(n, x, y, \vec{w}))$$

where ϕ is a primitive recursive predicate. Observe that the RHS is \exists^1 because the stuff after the existential quantifier is a primitive recursive predicate: “ $a = A(c_\phi, \max(y, x, \vec{w}))$ ” is a primitive recursive predicate.

If we are to prove this, we would need a lemma to the effect that: for every primitive recursive relation $\phi(y, \vec{x})$, there is a c_ϕ s.t., for all \vec{x} ,

$$(\exists y)(\phi(y, \vec{x})) \longleftrightarrow (\exists y < A(c_\phi, \max(\vec{x}))) (\phi(y, \vec{x}))$$

However, as we shall see later (exercise 51) the desired lemma is false. Primitive recursive relations and primitive recursive functions do not behave in the same way!

For a good readable discussion of the significance of the Ackermann function have a look at [60].

Why do we not simply gnumber the primitive recursive functions and diagonalise out of them? That would give us a total computable function that is demonstrably not primitive recursive—and at less effort. It would indeed, but this route to the result, via the Ackermann function, is more informative and more fun.

EXERCISE 42 (*) (1991:5:10 (CS))¹¹

Define the terms *primitive recursive function*, *partial recursive function*, and *total computable function*.

Ackermann’s function is defined as follows:

$$A(0, y) := y + 1; \quad A(x + 1, 0) := A(x, 1); \quad A(x + 1, y + 1) := A(x, A(x + 1, y)).$$

For each n define $f_n(y) := A(n, y)$. Show that for all $n \geq 0$, $f_{n+1}(y) = f_n^{y+1}(1)$, and deduce that each f_n is primitive recursive. Why does this mean that the Ackermann function is total computable?

EXERCISE 43 (*)

¹¹This was question 10 on paper 5 of the Cambridge Computer Science tripos 1991.

1. Write out a definition of a constructor of **double recursion** so that you now have a retype of doubly recursive functions. (Do not worry unduly about how comprehensive your definition is.)
2. What would a ternary Ackermann function be? Sketch a proof that the ternary Ackermann function you have defined dominates all doubly recursive functions.
3. Outline how to do the same for higher degrees.

The Ackermann function involves recursion on two variables in a way that cannot be disentangled. The point of exercise 43 is that there is also treble recursion and so on. A function is **n -recursive** if it is declared by a recursion involving n entangled variables. Exercise 43 invites you to prove analogues—for each n —of the facts we have proved about the Ackermann function: namely, for every n there are functions that are n recursive but not $(n-1)$ -recursive, and one can prove their totality by a well-founded induction over the lexicographic product ordering on \mathbb{N}^n . Is every total computable function n -recursive for some n ? Sadly, no, but I shall not give a proof. [we will see later an example of a manifestly computable total function that is not n -recursive for any n .]¹² It turns out that the correct response to the news brought by the Ackermann function to the effect that not every total computable function is primitive recursive is not to pursue 2-recursive, 3-recursive and so on but rather to abandon altogether the idea that computable functions have to be total in order to be computable. For a sensible general theory we need to consider **partial** functions.¹³ This is because we want unbounded search¹⁴ to be allowed. The new gadget we need is μ -recursion, which corresponds to unbounded search. This is a sensible new constructor to reach for because any strategy for computing g will give rise to a strategy for computing g^{-1} : simply try g with successively increasing inputs starting at 0 and continue until you get the answer you want—if you ever do. The point is that, if we have a deterministic procedure for getting values of g , we will have a deterministic procedure for getting values of g^{-1} . That is to say, it appears that the class of functions that are plausibly computable (in an intuitive sense of ‘computable’) is closed under inverse.

So we augment the constructors of the retype of primitive recursive functions by allowing ourselves to declare f by $f(n, \vec{x}) := (\mu y)(g(y, \vec{x}) = n)$, once given g . Then $\mu y.\Phi$ is the least y such that Φ (if there is one) and is undefined otherwise.

¹²The multiply recursive (n -recursive for any n) functions are all provably total in the Σ_2 inductive fragment of PA.

¹³On page 55 we encountered a naturally occurring computable partial function that was not *really* strictly partial because there was a computable total function that in some sense encoded the same information. When I write that we must embrace partial functions I mean we must embrace even those partial functions that *cannot* be coded as total function in the way division by 2 can.

¹⁴Fans of [32] might be helped by a reminder that Hofstadter calls unbounded search FLOOP (as opposed to BLOOP, *bounded* search, which we saw on page 53).

Notice that, even with this new constructor, the rectype of μ -recursive functions is still countably presented.

But there is a catch. The unbounded search constructor preserves computability as long as its argument is a total function, but the inverse function that it gives us is not guaranteed to be total itself! Think about inverting $n \mapsto 2n$. The result is a function that divides even numbers by 2 and fails on odd numbers. No problem there. For the moment let f be that function. The problem comes when we try to invert f : how do we ever discover what $f^{-1}(3)$ is? It ought to be 6 of course, but if we approach it by computing $f(0)$, $f(1)$ and so on, we get stuck because the endeavour to compute $f(1)$ launches us on a wild goose chase. We could guess that the way to compute $f^{-1}(3)$ is to try computing $f(6)$, but we do not want to even think about nondeterminism, because this severs our chain to the anchor of tangibility that was the motivation for thinking about computability in the first place.

The upshot is that we cannot rely on being able to iterate inversion, so we cannot just close the set of primitive recursive functions under both the old constructors and this new one and expect to get a sensible answer. As the $n \mapsto 2n$ example shows, FLOOP might output a function that you cannot then FLOOP. Nor can we escape by doctoring the datatype declaration so that we are allowed to apply inversion only to functions satisfying conditions that—like totality—are ascertainable solely at run-time. That would not be sensible.¹⁵

Fortunately it will turn out that any function that we can define by more than one inversion can always be defined using only one.¹⁶ I am going to leave the *precise* definition of μ -recursive up in the air for the moment. We will discover what it is by attempting to prove the theorem that a function is μ -recursive iff it is computable by a machine.

At first blush it seems odd to formalise computability in such a way that a function can be computable but undefined, but this liberalisation is the key that unlocks computation theory. Perhaps on reflection it isn't so odd after all: all of us who have ever written any code at all know perfectly well that the everywhere-undefined function is computable—since we have all inadvertently written code that computes it!

Specifically, this enables us to connect syntactic concepts of computability—namely, function declarations—to semantic concepts—namely, computability by machines . . . to which we now turn.

¹⁵It is true that one can obtain a declaration of the μ -recursive functions as a rectype by simply adding to the constructors for the primitive recursive functions the declaration:

If $\phi(\vec{x}, y)$ is a total μ -recursive predicate, then $f(\vec{x}) := (\mu y)(\phi(\vec{x}, y) = 0)$ is a μ -recursive function.

and some writers do this, but this is philosophically distasteful for the reasons given: it makes for a less abstract definition.

¹⁶Unfortunately (as we shall see) this is not proved by exhibiting an algorithm for eliminating extra inversions: it's less direct than that.

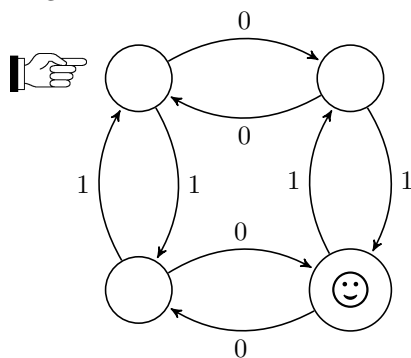
Chapter 5

Machines

There are various flavours of machines you may have heard of: Finite state machines both deterministic and nondeterministic, pushdown automata, linear bounded automata, Minsky machines, Turing machines and no doubt others. Each flavour of machine gives rise to a concept of computable function, and of course that is why they crop up here. However there are in fact only two kinds of machine we will be concerned with here (life is short) and they are *Finite State machine* and *Turing Machine* (or **Register Machine**). They correspond somehow to a minimal concept and a maximal concept of finite computation.

5.1 Finite State Machines

Let's start off with a nice picture that is just complicated enough to show all the features of interest and yet still simple enough for one still to be able to see what it's doing.



You start at the state indicated by the finger, and move from one state to another by following the labelled arrows—the labels on the arrows are letters from the input alphabet. If you land on a state (in this case there is only one) decorated with a smiley you **accept** the string. (And ‘accept’ is a term of art)

If you think about this machine for a bit you will see that it **accepts** precisely those strings that contain an odd number of 0s and an odd number of 1s. We express this by saying that it **recognises** the *set*

$$\{s : s \text{ contains an odd number of 0s and an odd number of 1s}\}.$$

That is to say: a machine **accepts** a string, but **recognises** a *set* of strings. Do not confuse these two verbs.

It's worth saying a little bit about finite state machines because nondeterminism arises naturally in this context.

Look at the Regular languages and Finite Machines notes on [51]. You might also derive some entertainment from the embarrassingly elementary http://www.dpmms.cam.ac.uk/~tf/cam_only/1aCSmaterials.html (designed for first-year computer science students!)

Pumping Lemma; nonexistence of a universal machine
Equivalence of deterministic and nondeterministic machines
Kleene's theorem

The pumping lemma says that the language recognised by a FSA is closed under a certain ["pumping"] operation.

For any two natural numbers a and b , the set of all base- a representations of natural-numbers-divisible-by- b is a regular language. However it is much easier to test whether a number is divisible by 7 if it is presented to us in octal than if it is presented to us in decimal. This reminds us that we are never computing with *numbers* but always with *representations* of numbers. See the remark of Enderton quoted on p 9.

No, that's not true, but something like it is, and can be used to make the same point.

EXERCISE 44 *Show that, for any base b , the set of base- b notations for natural numbers is a regular language.*

5.1.1 Kleene's theorem

Kleene's theorem states that a language is regular if it can be notated by a regular expression. One direction of this is fairly easy: showing that if there is a regular expression for a language L then there is a machine that recognises L . This breaks down into several steps, one for each constructor: slash, concatenation and Kleene star. We've seen how to do slash—after all $L(K_1|K_2)$ is just $L(K_1) \cup L(K_2)$ —but to do the other two involves nondeterministic machines and we don't encounter those until later.

The hard part is the other direction: showing how to find a regular expression for the language recognised by a given machine.

What we prove is something apparently much stronger:

For every machine \mathfrak{M} , for any two states q_1 and q_2 of \mathfrak{M} , and for any set Q of states, there is a regular expression $\phi(q_1, q_2, Q)$ which notates the set of strings that take us from state q_1 to state q_2 with all intermediate states lying within the set Q .

Of course all we are after is the regular expression formed by putting slashes between all the expression $\phi(q_1, q_2, Q)$ where q_1 is the initial state, q_2 is an accepting state, and Q is the set of all states. But it turns out that the only way to prove this special case is to prove the much more general assertion.

We prove this general assertion by induction. The only way to have a hope of understanding this proof is to be quite clear about what it is we are proving by induction. You are probably accustomed to having ‘ n ’ as the induction variable in your proofs by induction so let’s do that here.

“For all machines \mathfrak{M} , and for all subsets Q of the set of \mathfrak{M} ’s states with $|Q| = n$, and for any two states q_1 and q_2 of \mathfrak{M} , there is a regular expression $\phi(q_1, q_2, Q)$ which notates the set of strings that take us from state q_1 to state q_2 while only ever moving between states in the set Q ”

We fix \mathfrak{M} once for all (so that we are doing a ‘ \forall -introduction rule, or “universal generalisation” on the variable ‘ \mathfrak{M} ’) and we prove by induction on ‘ n ’ that this is true for all n .

At the risk of tempting fate, I am inclined to say at this point that if you are happy with what has gone so far in this section (and that is quite a big if!) then you have done all the hard work. The proof by induction is not very hard. The hard part lay in seeing that you had to prove the more general assertion first and then derive Kleene’s theorem as a consequence.

Proofs by induction all have two parts. (i) A base case, and (ii) induction step. I submit, ladies and gentlemen, that the base case—with $n = 1$ —is obvious. Whatever \mathfrak{M} , s and q are, you can either get from q_1 to q_2 in one hop by means of—character c , say (in which case the regular expression is c)—or you can’t, in which case the regular expression is ϵ .

Now let’s think about the induction step. Suppose our assertion true for n . We want to prove it true for $n + 1$.

We are given a machine \mathfrak{M} , and two states q_1 and q_2 of \mathfrak{M} . We want to show that for any set Q of states of \mathfrak{M} , with $|Q| = n + 1$, there is a regular expression that captures the strings that take the machine from q_1 to q_2 without leaving Q .

What are we allowed to assume? The induction hypothesis tells us that for any two states s' and t' and any set Q' of states with $|Q'| = n$, there is a regular expression that captures the strings that take the machine from q'_1 to q'_2 without leaving Q' . (I have written ‘ q'_1 ’ and ‘ q'_2 ’ and Q' because I don’t want to reuse the same variables!)

For any state r in Q , we can reason as follows: “Every string that takes \mathfrak{M} from q_1 to q_2 without leaving Q either goes through r or it doesn’t.

The strings that take \mathfrak{M} from q_1 to q_2 without either going through r or leaving Q are captured by a regular expression because $|Q \setminus \{r\}| = n$. Let w_1 be this regular expression.

The strings that take \mathfrak{M} from q_1 to q_2 via r are slightly more complicated. By induction hypothesis we have a regular expression for the set of strings that take \mathfrak{M} from q_1 to r without going through q_2 (while remaining in Q)—because $|Q \setminus \{q_2\}| = n$ —so let's call that regular expression w_2 . Similarly by induction hypothesis we have a regular expression for the set of strings that take \mathfrak{M} from r to q_2 without going through q_1 (while remaining in Q)—because $|Q \setminus \{q_1\}| = n$ —so let's call that regular expression w_3 . Finally by induction hypothesis we have a regular expression for the set of strings that take \mathfrak{M} from r back to r without going through q_1 or q_2 (while remaining in Q)—because $|Q \setminus \{q_1, q_2\}| = n - 1$ —so let's call that regular expression w_4 .

Now a string that takes \mathfrak{M} from q_1 to q_2 via r will consist of a segment that takes \mathfrak{M} from q_1 to r (captured by w_2) followed by a bit that takes it from r back to r any number of times (captured by w_4^*) followed by a bit that takes \mathfrak{M} from r to q_2 (captured by w_3).

So the regular expression we want is $w_1|w_2(w_4^*)w_3$.

This concludes the proof.

If you are a confident and fluent programmer in a language that handles strings naturally then you should try to program the algorithm on which this proof relies. It will give you good exercise in programming and will help you understand the algorithm.

5.1.2 The Thought-experiment and Myhill-Nerode

I am in a darkened room, whose sole feature of interest (since it has neither drinks cabinet nor coffee-making facilities) is a wee hatch through which somebody every now and then throws at me a character from the alphabet Σ . My only task is to say “yes” if the string of characters that I have had thrown at me so far is a member of L and “no” if it isn't (and these answers have to be correct!)

After a while the lack of coffee and a drinks cabinet becomes a bit much for me so I request a drinks break. At this point I need an understudy, and it is going to be you. Your task is to take over where I left off: that is, to continue to answer correctly “yes” or “no” depending on whether or not the string of characters that we (first I and then you) have been monitoring all morning is a member of L .

What information do you want me to hand on to you when I go off for my drinks break? Can we devise in advance a form that I fill in and hand on to you when I go off duty? That is to say, what are the parameters whose values I need to track? How many values can each parameter take? How much space do I require in order to store those values?

The thought-experiment encourages us to think about what distinctions we need to make between strings from Σ^* if we are to be able to tell members of $L \subseteq \Sigma^*$ from members of $\Sigma^* \setminus L$. We can give a more formal, more *mathematical* account, due essentially to Myhill-Nerode.

Let $L \subseteq \Sigma^*$, *not* assumed to be regular. We will define an ω -sequence $\langle \sim_n : n \in \mathbb{N} \rangle$ of equivalence relations on Σ^* . \sim_0 is the equivalence relation of index 2 whose two equivalence classes are L and $\Sigma^* \setminus L$.

Thereafter we say

$$w \sim_{n+1} w' \iff w \sim_n w' \wedge (\forall x \in \Sigma)(w::x \sim_n w'::x)$$

The equivalence relations in this sequence are of monotonically increasing strictness, so if we iterate long enough we will reach a fixed point. Observe that the \sim_n are all of finite index [might be an idea to prove this in some detail]. The recursion is of finite character, so we know that we will reach a fixed point at stage ω (with $\bigcap_{n \in \mathbb{N}} \sim_n$) if not before. Let \sim (without the subscript) be the fixed point. Evidently we can think of the equivalence classes in Σ^* / \sim as the states of a machine: the initial state is the equivalence class of the empty string and the accepting states are those equivalence classes that meet L . If the machine is finite then clearly L is a regular language.

And *vice versa*: equally clearly, if L is regular, then there will be a fixed point of finite index: every machine that recognises L embodies a fixed point. Observe that we can prove by induction on n that \sim_n is a superset of any fixed point, so the fixed point this supplies is the least fixed point. Accordingly we can conclude that, for any regular language L , there is a unique minimal machine that recognises it.

This is the **Myhill-Nerode** theorem.

What if L is not finite? The construction we have just seen is a sensible construction and may well give us a sensible answer (as it does for example when L is the matching brackets language). Our determination to think of mathematical objects as *finite* objects (wherever possible) leads us to invent other machine architectures to help us think of these sensible (infinite) quotients as finite objects. However that is for later, and for the moment we occupy ourselves with finite state machines of the kind described, but allowed to be *nondeterministic*.

5.1.3 Nondeterministic Machines

A nondeterministic machine is just like a deterministic machine except that its transition behaviour isn't deterministic. If you know the state a deterministic machine \mathfrak{M} is in then you know what state it will go to when you give it character a (or b or whatever). With a nondeterministic machine you only know the set of states that it *might* go to next. Notice that a deterministic machine is simply a nondeterministic machine where this set-of-states-that-it-might-go-to is always a singleton.

Nondeterministic machines (hereafter **NFAs**—“nondeterministic finite automata”) are a conceptual nightmare. The fact that they are nondeterministic makes for a crucial difference between them and deterministic machines. In the deterministic case you don’t have to distinguish in your mind between its behaviour in principle and its behaviour in practice, since its behaviour in practice is perfectly reproducible. That means that you can think of a deterministic machine either as an abstract machine—a drawing perhaps—or as a physical machine, according to taste. With NFAs there is a much stronger temptation to think of them as actual physical devices whose behaviour is uncertain, rather than as abstract objects. And the difficulty then is that NFAs are not physically realisable in the way one would like.

If NFAs are so nasty, why do we study them? The answer is that they tie up some loose ends and enable us to give a smooth theoretical treatment that improves our understanding and appreciation. So let us get straight what they are for. We started this chapter with a connection between machines and languages. A machine accepts strings and recognises a language. A (physical) nondeterministic machine can accept strings in exactly the same way that a (physical) deterministic one does: you power it up, and feed in the characters one-by-one and when it’s finished reading the string its either in an accepting state or it isn’t. The subtlety is that a nondeterministic machine, on having read a string, might be in any of several states, perhaps some of which are accepting and perhaps some not. The only sensible definition we can give of an (abstract) nondeterministic machine recognising a language is this:

The language recognised by a nondeterministic machine \mathfrak{M} is the set of strings that one of its physical realisations might accept.

The task of remembering and understanding this definition is made much easier for you once you notice that the definition for recognition of languages by deterministic machines is simply a special case of this.

Nondeterministic machines are useful to us because of the combination of two facts.

- (i) **If L is a language recognised by a nondeterministic machine \mathfrak{M} then there is a deterministic machine \mathfrak{M}' which can be obtained in a systematic way from \mathfrak{M} that also recognises L .**
- (ii) **There are circumstances in which it is very easy to produce a nondeterministic machine that recognises a language but no obvious easy way to produce a deterministic one.**

Let us now prove (i) and illustrate (ii).

(i): Finding a DFA that emulates an NFA

Suppose I have a nondeterministic machine \mathfrak{M} , presented to me in its start state. I have a handful of characters $\{c_1, c_2, c_3 \dots\}$ that I feed to the machine one by

one. Initially I know the machine is in the start state. But after I've given it c_1 I know only that it is in one of the states that it can go to from the start state on being given c_1 . And after I've given it c_2 I know only that it is one of those states it can reach from the start state in two hops if given c_1 followed by $c_2 \dots$ and so on. We seem to be losing information all the time. But all is not lost. Although I do not have certain knowledge of *the state \mathfrak{M} is in*, I do nevertheless have certain knowledge of *the set of states that it might be in*. And this is something I can keep track of, in the following sense. I can say "If it's in one of the states s or s' or s'' and I give it character c then either it was in s in which case it's now in s''' or s'''' or it was in s' in which case it's now in \dots ". In other words

If
 I know the set of states that it might be in now
 (and know that it must be in one of them)
 and
 I know the character it is being given,
 then
 I know the set of states that it might be in next
 (and I know that it must be in one of them).

Now comes the trick. Think of the set-of-states-that-it-might-be-in as a state of a new machine! One way of seeing this is to think of the states of the new deterministic machine as the states of uncertainty *you* might be in concerning the state of the nondeterministic machine. We have seen something like this before: in the discussion of the thought-experiment we were viewing states of the machine as states of knowledge of the string-so-far; this time we are thinking of states of the new (deterministic) machine as states-of-knowledge-of-what-state-the-nondeterministic-machine-might-be-in.

Observe that this "power set construction" supplies us, free, with the empty set of states. Clearly there can be no arrow to it! This empty set of states can thus correspond to a **fail** state, and we can now make sense of the convention that missing arrows take you to a fail state. If you "cannot go anywhere" from state s when you receive character c , then in the power set construction of a DFA any (meta)state that contains s must have an arrow to the empty set of states labelled ' c '.

(ii) An Application of NFAs

I mentioned earlier that the concatenation of two regular languages is regular. Suppose I have a deterministic machine \mathfrak{M}_1 that recognises L and a deterministic machine \mathfrak{M}_2 that recognises K . The idea is to somehow "stick \mathfrak{M}_2 on the end of \mathfrak{M}_1 ".

The difficulty is that if w is a string in LK , it might be in LK for more than one reason, since it might be decomposable into a string-from- K followed

by a string-from- L in more than one way. So one can't design a machine for recognising LK by saying "I'll look for a string in K and then—when I find one—swap to looking for a string in L ". You have to start off imagining that you are in \mathfrak{M}_1 ; that much is true. However when you reach an accepting state you have to choose between (i) staying in \mathfrak{M}_1 and (ii) making an instantaneous hop through a trap-door to the start-state of \mathfrak{M}_2 . That is where the nondeterminism comes in. These instantaneous hops are called " **ϵ -transitions**". You do them *between* the clock ticks at which you receive new characters. I don't like **ϵ -transitions** and I prefer theoretical treatments that don't use them. However, they do appear in the literature and you may wish to read up about them.

For those who do like ϵ -transitions, here is a description of a nondeterministic machine that recognises LK . It looks like the disjoint union $\mathfrak{M}_1 \sqcup \mathfrak{M}_2$ of \mathfrak{M}_1 and \mathfrak{M}_2 . Transitions between the states of \mathfrak{M}_1 are as in \mathfrak{M}_1 and transitions between the states of \mathfrak{M}_2 are as in \mathfrak{M}_2 . In addition for each accepting state of \mathfrak{M}_1 there is a ϵ -transition to the start state of \mathfrak{M}_2 .

For those of you who—like me—do not like ϵ -transitions, here is a different nondeterministic machine that recognises LK . Like the last one, it looks like the disjoint union $\mathfrak{M}_1 \sqcup \mathfrak{M}_2$ of \mathfrak{M}_1 and \mathfrak{M}_2 . Transitions between the states of \mathfrak{M}_1 are as in \mathfrak{M}_1 and transitions between the states of \mathfrak{M}_2 are as in \mathfrak{M}_2 . In addition, whenever s is a state of \mathfrak{M}_1 and c a character such that $\delta(s, c)$ is an accepting state of \mathfrak{M}_1 , we put in an extra arrow from s to the start state of \mathfrak{M}_2 , and label this new arrow with a ' c ' too. The effect of this is that when you are in s and you receive a c , you have to guess whether to stay in \mathfrak{M}_1 (by going to an accepting state in \mathfrak{M}_1) or make the career move of deciding that the future of the string lies with K , in which case you move to the start state of \mathfrak{M}_2 .

The manner in which we got rid of ϵ -transitions in this case is perfectly general. You can always get rid of them by introducing a bit of nondeterminism in the way we have just done.

And you can go in the other direction too.

5.1.4 Exercises

1. Prove that $L((r|s)^*) = L((r^*s^*)^*)$ (Use induction on word length)
2. Prove that $L((rs^*)^*) \subseteq L((r^*s^*)^*)$ but that the reverse inclusion does not hold.
3. Describe¹ deterministic automata to recognise the following subsets of $\{0, 1\}^*$:
 - (a) The set of all strings with three consecutive 0's; provide a regular expression corresponding to this set as well;
 - (b) The set of all strings w such that every set of five consecutive characters in w contains at least two 0's;
 - (c) The set of all strings such that the 10th character from the right end is a '0'; provide a regular expression corresponding to this set as

¹This word is very carefully chosen!

well. *For pedants:* This could mean one of two things. Answer both of them.

4. Let L be a regular language over an alphabet Σ . Which of the following are regular languages?
 - (a) $\{w \in \Sigma^* : (\exists u \in \Sigma^*)(wu \notin L)\}$
 - (b) $\{w \in L : (\forall u \in \Sigma^*)((\text{length}(u) > 0) \rightarrow wu \notin L)\}$
 - (c) $\{w \in L : (\forall u, v \in \Sigma^*)((w = uv \wedge \text{length}(u) > 0) \rightarrow u \notin L)\}$
 - (d) The preceding question has a typo in it. Find it.
 - (e) S , an arbitrary subset of L .
 - (f) $\{w \in \Sigma^* : (\exists u, v \in \Sigma^*)(w = uv \wedge vu \in L)\}$ (*hint: needs a different approach ...*)
5. A combination lock has three 1-bit inputs and opens just when it receives the input sequence 101, 111, 011, 010. Design a finite deterministic automaton with this behaviour (with accepting state(s) corresponding to the lock being open).
6. Let Σ be an alphabet and let B and C be subsets of Σ^* such that the empty string is not in B . Let $X \subseteq \Sigma^*$ and show that if X satisfies the equation $X = BX \cup C$, then $B^*C \subseteq X$ and $X \subseteq B^*C$, i.e. the unique solution is $X = B^*C$. [Hint: use induction on number of “blocks”.]
7. Show that if in the previous question we allow $\epsilon \in B$, then $X = B^*D$ is a solution for any $D \supset C$.
8. Let $A = \{b, c\}$, $B = \{b\}$, $C = \{c\}$. Find the solutions $X_1, X_2 \subset A^*$ of the following pairs of simultaneous equations: (i) $X_1 = BX_1 \cup CX_2$; $X_2 = (B \cup C)X_1 \cup CX_2 \cup \{\epsilon\}$ (ii) $X_1 = (BX_1 \cup \{\epsilon\})$; $X_2 = BC(X_1 \cup \{\epsilon\})$.
9. There is an alphabet Σ with six letters a, b, c, d, e and f that represent the six rotations through $\pi/2$ radians of each face of the Rubik cube. Everything you can do to the Rubik cube can be represented as a word in this language. Let L be the set of words in Σ^* that take the cube from its initial state back to its initial state. Is L regular?
10. Construct an FDA to recognise binary representations of multiples of 3. You may assume the machine starts reading the most significant bit first. Provide a regular expression for this language.
11. For which primes p can you build a FDA to recognise decimal representations of multiples of p ? How many states do your machines have?
12. Let q be a number between 0 and 1. Let L be the set of sequences $s \in \{0, 1\}^*$ such that the binary number between 0 and 1 represented by s is less than or equal to q . Show that L is a regular language iff q is rational. What difference would it have made if we had defined L to be the set

of sequences $s \in \{0, 1\}^*$ such that the binary number between 0 and 1 represented by s is less than q .

13. Give regular grammars for the two following regular expressions over the alphabet $\Sigma = \{a, b\}$ and construct finite non-deterministic automata accepting the regular language denoted by them:

- (a) $ba|(a|bb)a^*b$
- (b) $((a|b)(a|b))^*|((a|b)(a|b)(a|b))^*$

14. For each of the following languages either show that the language is regular (for example by showing how it would be possible to construct a finite state machine to recognise it) or use the pumping lemma to show that it is not.

- (a) The set of all words not in a given regular language L .
- (b) The set of all palindromes over the alphabet a, b, c .
- (c) If L is a regular language, the language which consists of reversals of the words in L ; thus if L contains the word $abcd$, then the reversed language L^R contains $dcba$.
- (d) Given regular languages L and M , the set of strings that contain within them first a substring that is part of language L , then a substring from M ; arbitrary characters from the alphabet a, b, c are allowed before, between and after these strings.
- (e) Given regular languages L and M , the set of strings that contain within them some substring which is part of both L and M .

15. What is the language of boolean (propositional) logic? Is it regular? What about the version without infixes ("Polish notation") What about reverse Polish notation?

16. Give context-free grammars generating the following languages:

- (a) $\{a^p b^q c^r : p \neq q \vee q \neq r\}$
- (b) $\{w \in \{a, b\}^* : w \text{ contains exactly twice as many } a\text{'s as } b\text{'s}\}$

17. Let M be a finite deterministic automaton with n states. Prove that $L(M)$ is an infinite set if and only if it contains a string of length l with $n \leq l < 2n$.

EXERCISE 45 (*Part III Computability and Logic 2014, modified*).

An **interleaving** of two words w_1 and w_2 is a word obtained by inserting the characters from w_1 into w_2 in the order in which they appear in w_1 . Thus, for example, both the strings $b0a1c$ and $ba01c$ are interleavings of the two strings bac and 01 .

Now let L_1 and L_2 be regular languages over alphabets Σ_1 and Σ_2 respectively. Let the interleaving $L_1 \oplus L_2$ of two languages L_1 and L_2 be the set of words that can be obtained by interleaving words from L_1 with words from L_2 . Prove that

1. *The interleaving of two regular languages is regular*
2. *The interleaving of a regular and a context free language is context free*
3. *The interleaving of two context free languages is not always context-free.*²

5.2 Machines with infinitely many states

Finite state machines are the most impoverished conception of computing machine. Rather than progress through gradually richer architectures we are going to jump straight to the maximal concept of [finite!] computing machine. It does not matter what kind of architecture our machines have as long as they have unbounded memory and can run arbitrarily long. The paradigm we use for the sake of illustration is the *register machine*.

If you want to see a Turing machine at work go to

<http://robotzeitgeist.com/2010/03/model-turing-machine.html>

I don't know about you, but I for one am very struck by the fact that the Turing machine in this video has a camera to look at the tape. I suppose it ought to be obvious that—from the machine's point of view—the tape is part of the external world, so it has to use its exteroceptors (not its proprioceptors) to examine it.

A register machine has

- (i) finitely many registers $R_1 \dots R_n$ each of which holds a natural number; and
- (ii) A **program** that is a finite list of **instructions** each of which consists of a **label** and a **body**. Labels are natural numbers, and a body has one of the three forms:
 1. $R^+ \rightarrow L$: add 1 to contents of register R and jump to instruction with label L .
 2. $R^- \rightarrow L', L''$: if contents of R is nonzero, subtract 1 from it and jump to the instruction with label L' ; otherwise jump to the instruction with label L'' .
 3. HALT!

We can represent instructions of flavour (1.) as triples $(j, +, k)$ and instructions of flavour (2.) as quadruples $(j, -, k, l)$. Then a register machine program is a finite sequence of triples-or-quadruples, where the n th member of the sequence is the instruction to be executed when in state n .³

The **output** of the register machine is the contents of register 1 (say) when the machine executes a HALT command. Notice that we don't really specify the

² There is a pumping lemma for context-free languages which is not in the course. With the hint that $a^n b^m c^n d^m$ is not context-free it all becomes terribly easy!

³I lifted this from PTJ's book, but I won't make much use of it. There are some exercises in the body of this text which come from his Part II lectures of long ago. Pursue them at your own risk.

number of registers by stipulation but only indirectly by mentioning registers in the instructions in the program. If the program has only ten lines, it cannot mention more than ten registers, and so the machine can be taken to have only ten registers.

We say that a register machine \mathfrak{M} **computes** a function f iff, for all $n \in \mathbb{N}$, $f(n)$ is defined iff whenever we run \mathfrak{M} starting with n in register 1 (and 0—say—in every other register) it halts with $f(n)$ in register 1 and does not halt otherwise.

For functions of arity greater than 1 we use more registers. Details could be provided, but they don't really matter.

It is very important that the register machines can be effectively enumerated, but deeply unimportant how we do it, though one can collect a few hints.⁴

We need to think about how to encode machines ...

The sequence of length k whose n th entry is e_n is sent to $\prod_{0 < n \leq k} p_n^{1+e_n}$.

(p_n is of course the n th prime.⁵) Thus—for example—the sequence $\langle 1, 8, 7, 3 \rangle$ is sent to $2^{1+1} \cdot 3^{8+1} \cdot 5^{7+1} \cdot 7^{3+1}$.

The prime powers trick lets us code lists of numbers as numbers. If we do this, the usual list-processing functions **head**, **tail** and **cons** will be primitive recursive. Although it is simultaneously very important that the register machines can be effectively enumerated yet deeply unimportant how we do it, there is one fact about how we do it that we will need, and that is that the map from numbers to machines should be computable in some sense. We can describe a machine completely in a specification language of some kind, because a machine is after all a finite object, and it will have a finite description, and we can have a standardised uniform way of presenting these descriptions.

The specification language can be written in an alphabet with perhaps 256 characters (alphanumerics and punctuation; ASCII codes are numbers below 256!), so we can assign to each formula in the specification language a *Gödel number* which is a number to base 256. Thus if we identify a machine with its description in the language, it can be thought of as a numeral to base 256. This

⁴Indeed it is deeply important that it is unimportant, for this is another *invariance* point:

“That’s very important,” the King said, turning to the jury. They were just beginning to write this down on their slates, when the White Rabbit interrupted: “Unimportant, your Majesty means, of course,” he said in a very respectful tone, but frowning and making faces at him as he spoke.

“Unimportant, of course, I meant,” the King hastily said, and went on to himself in an undertone, “important–unimportant–unimportant–important–” as if he were trying which word sounded best.

Some of the jury wrote it down “important,” and some “unimportant”. Alice could see this, as she was near enough to look over their slates; “but it does not matter a bit,” she thought to herself.

see [11], available online.

⁵Observe that this encoding is not surjective: for example the number 14 does not encode any sequence. I don’t know if this matters.

numeral will not be a mere *name* of the machine, but an actual *description* of it.

\mathbb{N} is a rectype, and so is the set of machine descriptions in the specification language. The gnumbering function given is nice in the sense that it is a rectype homomorphism. (It's an *acceptable enumeration*).

If a formula is a list of symbols, we can define a Gödel enumeration of formulæ by list-recursion as shown in the following ML pseudocode.

```
gnumber h::[ ] = ASCII of h
|      h::t  = 256*gnumber(t) + ASCII of h;
```

DEFINITION 10 *Hartley Rogers [49] says a system of indices ψ is **acceptable** if, for every n , there are total computable f and g such that*

$$\psi_e^n \simeq \phi_{f(e)}^n \wedge \phi_e^n \simeq \psi_{g(e)}^n$$

Here's what i think is going on. The obvious way to enumerate functions-in-intension is by gnumbering the syntax, or the machines. That way, if i give you a number, you can examine it and see which function it is the gnumber of. If i do things that way it turns out that, for example, the set $\{n : \{n\}(0) \downarrow\}$ is indeed semidecidable in the operational sense—i can indeed verify membership in it of any actual member in finite time. Now suppose I compose that gnumbering with some extremely nasty uncomputable permutation—you can see what happens.

So an enumeration is going to be acceptable if it respects the structure of the syntax or of the family of machines (the two constraints will presumably turn out to be equivalent). If one tries to make this rigorous one will presumably find oneself exploiting the idea of a function from machines/syntax to \mathbb{N} defined by recursion on the recursive structure of the counted set of machines/wffs. Finally one will discover that any two enumerations defined in this way are mutually conjugate via some computable permutation of \mathbb{N} . And I think that is what the dfn of Rogers is saying.

We don't yet know what "computable" means.

From now on we are going to assume we have fixed an enumeration of register machines in this style, so that the m th machine is the machine with gnumber m . There is a convention of writing ' $\{e\}$ ' for the function computed by the e th machine/ e th program (we do not distinguish between machines and programs), and also writing

DEFINITION 11

- " $\{e\}(n) \downarrow = k$ " to mean that the e th machine halts with input n and outputs k ;
- " $\{e\}(n) \uparrow$ " means that the e th machine does not halt with input n . In these circumstances we say $\{e\}(n)$ **diverges**.

- “ $\{e\}_z(n)\downarrow$ ” to mean that the e th machine halts with input n in $\leq z$ steps.
- “ $\{e\}_z(n)\downarrow = x$ ” to mean that the e th machine halts with input n in $\leq z$ steps and will output x .

I am writing ‘ $\{e\}$ ’ for the function computed by the program with gnumber e (or the machine with model⁶ number e). But since there is a correspondence between machines and programs we will sometimes write ‘ $\{p\}$ ’ for the function-in-intension (program) with gnumber p .

The following notation is standard: ‘ W_e ’ for $\{n \in \mathbb{N} : \{e\}(n)\downarrow\}$. (The ‘ W ’ comes from the German *Wertebereich*, meaning *range of values*.)

One of the delights of the theory of computable functions is that we can equivocate over our data objects: it is of central importance that an object can be a number at one time and a computable function at another. Indeed, it can be both *at the same time*. This permanent possibility of equivocation makes for notational quicksand, so some explanation is in order.

When we equivocate on ‘ n ’ between a number and a function it is always between a number and a function-in-intension, not between a number and a function-in-extension. If we write ‘ n ’ *simpliciter* then we are thinking of n as a number. The point of the braces is that when we write ‘ $\{n\}$ ’ it is in order to disambiguate the equivocation, and to make it clear that it is the function that is meant. Further, if we want to make it crystal-clear that it is the function-in-extension that we mean not the function-in-intension then we can write ‘ $\text{Graph}(\{n\})$ ’. We do this (for example) in the proof of theorem 14.

In the spirit of this equivocation I should record that I shall sometimes write ‘ \mathfrak{M} ’ to denote the machine with gnumber m : thus \mathfrak{M} computes the function $\{m\}$.

DEFINITION 12

n is always a natural number;
 $\{n\}$ is the n th program;
 $\text{Graph}(\{n\})$ is the function-in-extension computed by $\{n\}$;
 \mathfrak{n} is the machine that computes $\{n\}$.

5.3 The μ -recursive functions are precisely those computed by register machines

An essential gadget is

⁶In earlier draughts I had “chassis number” here. That is of course wrong. Two cars of the same model (which do the same thing) have different *chassis* number but the same *model* number. The chassis number belongs to the *token* of the machine, whereas the part that matters to us pertains to the *type*.

DEFINITION 13 (Kleene's T function)

Input m and i and t , then output a list of t states of the m th machine started with input i , one for each time $t' < t$. (The state of a register machine is the tuple of contents of the registers and the current instruction.)

The output, $T(m, i, t)$, of Kleene's T -function is commonly called a **complete course of computation**. It is entirely plausible that T is computable since, as long as (1) the gnumbering is sensible in the sense that the gnumber of a machine is a *description* of it, and (2) the machines have standard architecture then, on being given a gnumber m , one can go away and build the machine described by m and then feed it input i and observe it for t steps. This is plausible because the machines have finite descriptions and are deterministic. Not only are they deterministic, but the answer to the question, "What state will it go to next?" can be found by looking merely at the machine and its present state, without consulting the positions of the planets or anything else that—however deterministic—is not internal to the machine. It is a lot less obvious that T is primitive recursive, but—as it so happens—it is. The proof is extremely laborious, but it relies merely on checking that all the functions involved in encoding and decoding are primitive recursive: nothing worse than exponentiation is required. (In fact, because of theorem 5 on p. 46 we can get by without even using exponentiation.) Observe, too, that the only searches we make are bounded searches, and bounded search is primitive recursive, as we saw earlier. (see theorem 7 p. 53.)

There is something to think about here. Kleene's T -function, properly understood, is really a hyperintensional object, something even more intensional than a function declaration. Really it has a secret extra parameter, which is the enumeration of machines: it's not a primitive recursive declaration in the same sense in which `mult 0 n = 0; mult (succ m) n = plus (n (mult (m, n)))` is a primitive recursive declaration of `mult`. The point is that the code for Kleene's T -function will not be what the CompScis call *self-validating*: that is to say that you can't tell of a primitive recursive declaration that it is a declaration of T merely by looking at it. There is a notion of *acceptable enumeration* lurking in the background.

Mind you, as Ben Millwood says, is this any more than the fact that all low level languages are (his word) *inscrutable*?

Another thought...the predicate " $o = T(m, i, t)$ " is primitive recursive. This will eventually give us an easy proof that the composition of two primitive recursive relations might not be primitive recursive.

This shows that

THEOREM 10 *The function $\{m\}$ computed by \mathfrak{M} , the m th machine, is μ -recursive.*

In other words, the machine with gnumber m computes the μ -recursive function: $i \mapsto$ the least k such that m started with i halts with output k .

Now for the converse.

THEOREM 11 *Every μ -recursive function can be computed by a register machine.*

Sketch of proof:

Consider the retype of functions built up from the initial functions (as in the declaration of primitive recursive functions) by means of composition, primitive recursion and μ -recursion. This class contains all sorts of functions that are undefined in nasty ways because it allows us to invert the results of inversions, and the result of inverting a function might not be total—as we have seen. Nevertheless, we can prove by induction on this datatype that for every declared function in it there is a register machine that computes it. That is, in the sense that whenever these declarations do not fall foul of common sense by attempting to invert functions that are not total, the machine that we build does indeed compute the function.

The details of how to glue together register machines for computing f and g into one that computes $f \circ g$ will be omitted, as will the details of how to compose register machines to cope with the primitive recursion constructor, and how to front-end something onto a register machine that computes $f(x, y, \vec{z})$ to get something that computes $\mu x.(f(x, y, \vec{z}) = k)$. ■

This completes the proof of the completeness theorem for computable functions.

5.3.1 A Universal Register Machine

Kleene's T -function is primitive recursive, so there is a machine that computes it. Any such machine can be tweaked⁷ into a **universal** or all-purpose machine: one that can simulate all others.

We need three auxiliary functions on memory-dumps:

`current_instruction(d)` and `register_0(d)`, which return respectively the current instruction and the contents of register 0;

`last` returns the last element of a list.

It is mechanical to check they are all primitive recursive. Once we have got those, we can build a machine that, on being given m and i , outputs:
`register_0(last(T(m, i, (μt)(current_instruction(last(T(m, i, t))) = HALT))))`
 which is what the m th machine does on being given i .

This machine is a **Universal Register Machine**.

5.4 Decidable and Semidecidable Sets

5.4.1 Zigzagging Autoparallelism: Volcanoes

Suppose X is the range of a computable function f and \mathfrak{M} is a machine that computes f . The idea of autoparallelism is that at stage n we run \mathfrak{M} with input `fst(n)` for `snd(n)` steps. When we do this with a machine the effect is that we keep trying the machine with all inputs, continually breaking off and revisiting

⁷By a process the computer scientists call *wrapping*.

| | | | | | | | | | |
|---|----|-----|----|-----|-----|-----|-----|-----|----|
| 5 | 15 | ... | : | ... | | | | | |
| | | ↘ | | | | | | | |
| 4 | 10 | | 16 | ... | | | | | |
| | | ↘ | | ⋮ | | | | | |
| 3 | 6 | | 11 | 17 | ... | | | | |
| | | ↘ | | ↘ | ⋮ | | | | |
| 2 | 3 | | 7 | 12 | 18 | ... | ... | | |
| | | ↘ | | ↘ | ↘ | ⋮ | | | |
| 1 | 1 | | 4 | 8 | 13 | | 19 | ... | |
| | | ↘ | | ↘ | ↘ | | ↘ | ⋮ | |
| 0 | 0 | | 2 | 5 | 9 | | 14 | | ⋮ |
| | | | | | | | | | 20 |
| | 1 | | 2 | 3 | 4 | 5 | 6 | | |

Volcanoes can come in more than one architecture. The crude architecture causes a volcano to keep revisiting computations that have already finished; there are more subtle volcano architectures that do not fall into this trap. Yet another (yet more subtle) style of volcano keeps track of the numbers it has

⁸The reader might be wondering whether or not the volcano, when (for example) it revisits the computation of $f(2)$ for 4 steps, is supposed to be able to remember the results of the computation of $f(2)$ it did earlier for 3 steps. That would mean that it only ever had to do one step of computation at each stage, and thereby speed things up a bit. To be able to do that it would have to have available to it an ever-increasing amount of memory. Ever-increasing, but always finite. It's a natural thing to wonder about, but reflection will show that it makes no difference one way or the other.

emitted, and never emits the same number twice: if one of its computations halts giving a value the volcano has already reported it passes over this event in tactful silence.

Volcanoes (of whatever architecture) can always be thought of as functions from time to \mathbb{N} . [might be a good idea to use the letter ‘ t ’ for inputs to volcanoes.]

EXERCISE 46 *Explain to your flatmate why, if we think of a volcano as a function $\mathbb{N} \rightarrow \mathbb{N}$ then that function is μ -recursive.*

Next explain why the cost function of a computable function-(in-intension) is computable. “Cost function”? A function-in-intension f is wlog a machine \mathfrak{M} and we define its cost function F by $F(n)$ = number of steps used by \mathfrak{M} on input n if $\mathfrak{M}(n) \downarrow$ and \uparrow o/w. Then show that every total computable function is (= has the same graph as) a volcano for a computable partial (in fact total) function. Suppose f is a total computable function $\mathbb{N} \rightarrow \mathbb{N}$, and F its cost function. Consider now the volcano for the function-in-intension f^* that, on being given input n , twiddles its thumbs for $\sum_{k < n} F(k)$ clock ticks, and then starts computing $f(n)$. The volcano for f^* will emit the values of f in the sequence $f(0), f(1), f(2) \dots$ (miniexercise) [each step that the volcano does in the computation of $f^*(1)$ is immediately after it does a step in the computation of $f^*(0)$ and—in computing $f^*(1)$ —it twiddles its thumbs until the computation of $f(0)$ has finished.]

5.4.2 Decidable and Semidecidable Sets

One of the intentions behind the invention of computable functions was to capture the idea of a decidable set. One exploits along the lines of something like “a set is decidable iff it is the range of a computable function” It turns out that that does not straightforwardly give us what we want. Suppose we want to know whether or not n is a member of a putatively decidable set, presented as $f^{-1}\mathbb{N}$, for some computable function f . If we use f ’s volcano then, if n is indeed a value of f , we will learn this sooner or later; but if it isn’t, this process will never tell us. However, this does at least give us a **verification procedure**: we can detect membership of $f^{-1}\mathbb{N}$ in these circumstances even though we are not promised an exclusion procedure. Thus the natural idea seems to be that of a *semidecidable* set: one for which membership can be confirmed in finite time. Perhaps nowadays one would be more likely to use words like ‘authenticate’ and ‘authentication’.

But is this the only way we can exploit computable functions to get a concept of semidecidable set? Being the range of a computable function seems a pretty good explication of the concept of a semidecidable set, but then being the set of arguments on which a computable function halts— $\{n : f(n) \downarrow\}$ —seems pretty good too. After all, if $f(n) \downarrow$, then we will certainly learn this in finite time. Fortunately for us, all obvious attempts to capture the concept of semidecidable set using these ideas give the same result.

REMARK 5 *The following conditions on a nonempty⁹ set $X \subseteq \mathbb{N}$ are equivalent:*

- (i) X is the range of a μ -recursive function;
- (ii) X is the set of naturals on which a μ -recursive function is defined;
- (iii) X is the range of a μ -recursive function that happens to be total.

Proof:

(i) \rightarrow (iii). Use volcanoes.

(The converse is obvious since (iii) is a special case of (i).)

Let g be the function that sends an input n to the n th thing emitted by \mathfrak{M} 's volcano. g is total, and clearly it outputs all and only the members of X . (I am ignoring the case where X is finite: it is a miniexercise to check for yourselves that it is in fact safe to ignore it!)

(i) \rightarrow (ii)

Given a machine \mathfrak{M} that outputs members of X , we can build a machine \mathfrak{M}' that, on being given a number n , runs \mathfrak{M} 's volcano until it produces the output n : \mathfrak{M}' then outputs 0, say (it does not matter). \mathfrak{M}' is then a machine that halts on members of X and on nothing else.

(ii) \rightarrow (i)

Given a machine \mathfrak{M} that halts on members of X , we can build a machine that outputs members of X by simply trapping the output of \mathfrak{M} and outputting the input instead of the output. ■

EXERCISE 47 *Show that every computable partial function has a computable (partial) right inverse.*

EXERCISE 48 *Show how to modify the volcano in part (iii) of remark 5 so that the total computable function that enumerates its emissions is one-to-one. (So it emits each member of X precisely once.)*

Incurable optimists might hope that volcanoes might give us a cure to the problem discussed on page 62 in section 4.3. After all, there is always the possibility of running g in parallel with itself. Will this help? Although that will turn up an input y to g s.t. $g(y, \vec{x}) = n$ if there is one, there is no reason to suppose it will turn up the smallest such y .

EXERCISE 49 *Cook up an example to show that sometimes it won't.*

Indeed, quite which one it turns up will depend on how we have implemented volcanoes, so even which functions turn out to be computable would depend on how we implement the algorithm! This is clearly intolerable.

We can now give a formal definition of 'semidecidable'.

⁹The empty set is obviously decidable!

DEFINITION 14

- (1) A set satisfying the conditions in remark 5 is **semi-decidable**¹⁰
 (2) A set X is **decidable** if X and $\mathbb{N} \setminus X$ are both semidecidable.

The original definition of decidable set as a set that is both semidecidable and the complement of a semidecidable set looks cumbersome and long-winded, and it might be felt that it would be more natural to define a set X to be decidable iff there is a total computable function $f : \mathbb{N} \rightarrow \{0, 1\}$ such that $X = f^{-1}\{1\}$. However if one starts with that definition it is much harder to motivate the concept of semidecidable set and the connection between the two ideas is less clear.

Observe that the graph of a computable function $\mathbb{N}^k \rightarrow \mathbb{N}$ is a semidecidable subset of \mathbb{N}^{k+1} and the graph of a total computable function $\mathbb{N}^k \rightarrow \mathbb{N}$ is a decidable subset of \mathbb{N}^{k+1} .

develop the parallels with
the graph of a p.r. function
being a p.r. set

Just as “computable function” is better than “recursive function” (because recursion is not always prominent in the declaration of a computable function) so “decidable set” is better than “recursive set” (the old terminology), since “recursive set” would suggest that there also ought to be “primitive recursive set”—you are one if you are the range of a primitive recursive function. But in fact

EXERCISE 50 (*)

- (1) Every nonempty semidecidable set is the range of a primitive recursive function. (Hint: Modify volcanoes by using Kleene’s T -function.)
 (2) Show that condition (i) of remark 5 is equivalent to “ X is $f^{-1}Y$ for some computable f and semidecidable $Y \subseteq \mathbb{N}$ ”.

EXERCISE 51 (*)

Give a primitive recursive relation ϕ for which the following fails:
 There is c_ϕ s.t., for all \vec{x} ,

$$(\exists y)(\phi(y, \vec{x})) \longleftrightarrow (\exists y < A(c_\phi, \max(\vec{x})))(\phi(y, \vec{x})).$$

OPEN QUESTION 1¹¹

Suppose $f : \mathbb{N} \rightarrow \mathbb{N}$ total computable with no odd cycles. Then there is $d : \mathbb{N} \rightarrow$

¹⁰The old terminology is ‘recursively enumerable’, which is gradually giving way (particularly across the pond) to ‘computably enumerable’ abbreviated to “c.e.”. That notation arises because any set of natural numbers can be enumerated (and *enumerable* or *denumerable* are old words for ‘countable’), but not necessarily by a computable function. If the set is enumerated by a *recursive* (or computable) function, it is *recursively* (or computably) enumerable. Bear in mind too that in some of the literature ‘semidecidable’ is used to mean ‘semidecidable and not decidable’.

¹¹This question was put to me years ago by my Ph.D. supervisor, Maurice Boffa. I don’t know if it is still open.

$\{0, 1\}$ with $(\forall n \in \mathbb{N})(d(f(n)) = 1 - d(n))$. Such a d is a discriminator for f . If f is computable must it have a computable discriminator?

EXERCISE 52 (*)

Check that, for all $A, B \subseteq \mathbb{N}$, the set $\{2n : n \in A\} \cup \{2n + 1 : n \in B\}$ is semidecidable iff both A and B are semidecidable.

DEFINITION 15 A infinite subset of \mathbb{N} set is **immune** if it has no infinite semidecidable subset.

‘Immune’ is a computable analogue of ‘infinite Dedekind-finite’.

EXERCISE 53 (Part III 2012 Paper 24 q 3—slightly modified)(*)

Prove that there is a semidecidable set $X \subseteq \mathbb{N}$ with $\mathbb{N} \setminus X$ infinite such that X meets every infinite semidecidable set. What is the asymptotic density of your X ?

See also exercise 78 in section 5.8.

Note the parallel between the idea of a *regular language*, which is the set of strings accepted by a finite-state machine, and the idea of a *semidecidable set*, which is the set of natural numbers on which a Turing machine will halt.

If X is semidecidable, it is $f^{-1}\mathbb{N}$ for some total computable f , so whenever $n \in X$ there is $k \in \mathbb{N}$ and a finite computation verifying that $f(k) = n$, so that $n \in X$. This finite computation should be thought of as a *proof* or *certificate* in the sense of the discussion on page 45, so a semidecidable set of naturals can be thought of as a subset of \mathbb{N} that happens to be a retype in its own right. Indeed, we can take this further: by means of gnumbering, every finitely presented retype can be thought of as a semidecidable set.

The following observation comes under the heading of soothing triviality. Not difficult but it makes you feel better. Actually we will need it later, in the proof of remark 8.

REMARK 6 Let X be a subset of \mathbb{N}^{k-1} . Then X is the projection of a decidable subset of \mathbb{N}^k iff it is semidecidable.

Proof:

left-to-right

Suppose X is $\{\vec{x} : (\exists n)(\vec{x}::n \in Y)\}$ where Y is a decidable subset of \mathbb{N}^k . Then to check, for any candidate $(k-1)$ -tuple \vec{x} , whether or not it is in X it suffices to find an n such that the k -tuple $\vec{x}::n$ is in Y . For each k -tuple \vec{x} the question “ $\vec{x}::n \in Y$?” can be answered mechanically in finite time, so if there is such an n we will find it in finite time by trying $n := 0, n := 1$ and so on. (No need for volcanoes.) But this is just to say that X is semidecidable.

right-to-left

Suppose X is semidecidable, so that $X = \text{dom}(\{m\})$ for some computable function $\{m\}$. Then

$$\begin{array}{ll}
\vec{x} \in X & \text{iff} \\
\{m\}(\vec{x}) \downarrow & \text{iff} \\
(\exists y)(\{m\}_y(\vec{x}) \downarrow) & \text{iff} \\
(\exists y)(\langle \vec{x}, y \rangle \in \{ \langle \vec{z}, y \rangle : \{m\}_y(\vec{z}) \downarrow \}) &
\end{array}$$

Observe that $\{ \langle \vec{x}, y \rangle : \{m\}_y(\vec{x}) \downarrow \}$ is clearly decidable, and that X is a projection of it.¹² ■

Observe that

- The left-to-right implication, above, is best possible. The projection of a decidable set is not always decidable: $\{ \langle n, m, k \rangle : \{n\}_k(m) \downarrow \}$ is a decidable set, but the halting set— $\{ \langle n, m \rangle : (\exists k)(\{n\}_k(m) \downarrow) \}$ —is a projection of it. We will see below (theorem 12) that the halting set is not decidable.
- The projection of a *semidecidable* subset of \mathbb{N}^k is likewise a semidecidable subset of \mathbb{N}^{k-1} . (This is because we can use **pair** to squash like quantifiers). This means that if X is the projection of a semidecidable set then it is the projection of a decidable set. Remark 8, below, of Craig is related to this.

So it's definitely all right to think of semidecidable sets as projections of decidable sets.

From now on we say “computable” instead of “ μ -recursive”. You may also hear people saying “general recursive” or “partial recursive”, which mean the same thing. Confusingly, you will also hear people talk about functions being *partial recursive* in contrast to being *total recursive*. (We would say ‘total computable’). A set is **decidable** if its **characteristic function**¹³ is total computable.

DEFINITION 16 *The characteristic function χ_A of $A \subseteq \mathbb{N}$ is*

$$\lambda x. \text{ if } x \in A \text{ then } 1 \text{ else } 0.$$

(The Greek letter ‘ χ ’ is the first letter of the Greek word for ‘character’.)

5.4.3 A Nice Illustration and a Digression

There is a natural example of an immune set, and it arises in a context of some independent interest.

It is natural to feel that the string 0^n (of n zeroes) is simple, in the sense that one can capture it by a description that has fewer than n characters.

What one wants to say is that a string σ is simple if there is a short string τ and a program f which outputs σ on being given τ . Well, of course there is: we

¹²Thanks to Philipp Kleppmann for this improvement on my original.

¹³In other traditions they are sometimes called **indicator functions**.

can simply hardcode σ into f . We need to work a little harder. What we want is a universal Turing machine U , which—for any computable f —will compute $f(\tau)$, by the following contrivance. U will associate—to each such f —a string ρ_f such that, for any τ , $f(\tau)$ is obtained as $U(\rho_f\tau)$. (Concatenation of strings is notated by juxtaposition, as with languages-and-automata). It is true that f can cheat, but he has to tell U how he did it, and that takes up bits.

We now say that $C(\sigma)$ is $|\tau|$ where τ is the shortest input¹⁴ on which U gives out σ . We say $\sigma \in \{0, 1\}^{<\omega}$ is incompressible if $C(\sigma) \geq |\sigma|/2$.

REMARK 7 $\{\sigma \in \{0, 1\}^{<\omega} : C(\sigma) \geq |\sigma|/2\}$ is immune



Proof:

Suppose this set had an infinite semidecidable subset, B , say. B is infinite and so must contain strings of arbitrary length. Since it is semidecidable there is a total function f whose range it is. Let h_n be the first string of length $\geq n$ that f puts into B . Then, by assumption, $C(h_n) \geq |h_n|/2 \geq n/2$. But manifestly the string h_n can be computed from n —by computing f . So this gives us $C(h_n) \leq C(n) +$ the (constant) length $|c_f|$ of the string c_f that U uses to compute f ¹⁵. $C(n) = \log_2(n)$ of course. This gives $n/2 \leq \log_2(n) + |c_f|$, and for n sufficiently large this is impossible; now B is infinite so we can take n as large as we like and obtain our contradiction. ■

5.4.4 “In finite time”—a warning

“In finite time” is a nice snappy expression, and it encapsulates a useful intuition. However one has to use it with care, since it can mislead. There are circumstances in which one is trying to construct a set X of natural numbers, by a process of length ω . At each stage one puts some stuff in and takes some other stuff out. So far so general. Suppose further that it is true of each $n \in \mathbb{N}$ that it only gets added or removed finitely often, so that it is determined “in finite time” whether or not n will be in X at close of play. This *sounds* as if X ought to be semidecidable or even decidable, but of course nothing of the sort can be guaranteed merely by the conditions outlined, since one might be unable to compute, for a number n , the stage $f(n)$ such that the final status of ‘ $n \in X$ ’ has been determined by stage $f(n)$. Of course, if there *is* such a computable f then X is decidable, but the “in finite time” thought does not guarantee that there should be.

I got into a tangle over this “in finite time” stuff, Dear Reader, and you might do too.

Consider the following example [which I only sketch here, co’s it’s best done at a board, and I will in fact do it at the board]. Suppose R and S are two decidable subsets of \mathbb{N}^2 which are wellorderings of \mathbb{N} of length ω . They are isomorphic, of course. Is the isomorphism a semidecidable set of ordered pairs?

¹⁴part of τ is of course the string c_f for the function f that U is calculating.

¹⁵As Ben Millwood says, the overhead is not *literally* c_f , but it’s at least a constant.

Another realistic case in point is the task of mentally reconstructing the proof of Friedberg-Muchnik, (this is theorem 25) once you have forgotten the details.

The following exercise might help/amuse you.

EXERCISE 54 (*)

A question from James Cranch, a real live Part III student in 2012/3. I can't remember what he needed it for. (Nor, it seems, can he)

Suppose $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ is total computable, and $(\forall x)(\exists y)(\forall z > y)(g(x, z) = g(x, y))$.

Then we can define $f : \mathbb{N} \rightarrow \mathbb{N}$ by $f(n) =$ the eventually constant value of $g(n, z)$ as $z \rightarrow \infty$.

There is of course no reason to suppose that f is going to be computable. What can we say about the graph of f ? $(\forall u, v)(\langle u, v \rangle \in f \iff (\exists y)(\forall z > y)(v = g(u, z)))$. In other words, the graph of f is an $\exists\forall$ set.

Cranch's question is: if we are given a total function $f : \mathbb{N} \rightarrow \mathbb{N}$ and told that its graph is an $\exists\forall$ set, can we find $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, total computable, such that $(\forall x)(\exists y)(\forall z > y)(g(x, z) = g(x, y))$, and $f : \mathbb{N} \rightarrow \mathbb{N}$ is defined by $f(n) =$ the eventually constant value of $g(n, z)$ as $z \rightarrow \infty$?

5.5 Decidable and semidecidable sets of other things

You will sometimes hear people talk about recursive or r.e. (or, as we say here, decidable and semidecidable) sets of—for example—computable functions, or formulæ. What they mean, of course, is a (semi-)decidable set of *indices* or *gnumbers* (of functions, or formulæ). You will even hear people say things like “A union of an r.e. set of r.e. sets is r.e.” [*sic*]. This is true, and so are some other things with the same kind of sound-bite.

EXERCISE 55

- (i) *A union of a semidecidable set of semidecidable sets is semidecidable;*
- (ii) *A union of a semidecidable set of decidable sets is decidable;*
- (iii) *A union of a semidecidable set of semidecidable sets is decidable;*
- (iv) *A union of a decidable set of decidable sets is decidable.*

In each case provide a proof or a counterexample.

In set theory we have the notion of a *setlike function*. If $f^{\text{“}x}$ is always a set whenever x is we say f is 1-setlike. Only “1”-setlike? If, additionally, $\{f^{\text{“}y} : y \in x\}$ is a set whenever x is a set¹⁶ we say f is 2-setlike. Similarly

¹⁶PTJ would have me write this as ‘ $f^{\text{“}x}$ ’!

3-setlike. A function that is n -setlike for every n is just plain *setlike*. In ZF we have the axiom scheme of replacement, and it tells us that every function-class is setlike, so one's attention is liable not to be drawn to this useful concept if one studies too much ZF.

There is a notion of “setlike” applicable also to theories that are not explicitly theories of sets. We can interpret a certain amount of second-order (and third-order and so on) arithmetic in the first-order theory of computable functions by encoding a semidecidable set of natural numbers as the gnumber of a function whose range it is. Thus in this context an externally visible set of things is a set from the point of view of computable function theory as long as its members are coded somehow as naturals, and the set itself is the range of a computable function defined on those naturals. Then we can repeat the trick, to represent (some!) sets of sets of naturals, and so on up.

In this setting it is natural to ask which functions $\mathbb{N}^k \rightarrow \mathbb{N}$ are setlike. Of course all computable functions are setlike. For example the nice primitive recursive pairing-and-unpairing gadget for natural numbers is *setlike* in the sense that if X and Y are (semi-)decidable subsets of \mathbb{N} , then so too are $X \times Y$, **fst** “ X ...

rewrite this section...S-m-n theorem

Because of the natural bijection between $(A \times B) \rightarrow C$ and $A \rightarrow (B \rightarrow C)$ one can think of a natural-number-valued function of two (natural number) variables either as a function $\mathbb{N}^2 \rightarrow \mathbb{N}$ or as a function $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$. We have a notion of primitive recursive function $\mathbb{N}^2 \rightarrow \mathbb{N}$ or as a function and a notion of a primitive recursive function $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$. One would hope that these are in some sense the same. What the primitive recursiveness of pairing/unpairing shows is that changing the way you think of a particular function of two natural-number variables from one of these ways to the other won't alter its primrec/non-primrec status.

5.5.1 Applications to Logic

We are now in a position to give a definition of *axiomatisable theory*. An axiomatisable theory is one with a set of axioms whose gnumbers form a semidecidable set. (It is assumed that the theory only has finitely many rules of inference. Without that condition, every theory in a countable language would be axiomatisable: take an empty set of axioms, and for each theorem have a nullary rule of inference whose conclusion is that theorem.) Take a moment to reflect on the significance of this notation: “axiomatisable” for “recursively axiomatisable”. If the set of axioms is not at least semidecidable then it fails of its purpose as an *axiomatisation*.

Have a look at all the theories you have met so far. Those with finitely many axioms are recursively axiomatisable; if you check you will see that all those with infinitely many axioms are recursively axiomatisable, too. Indeed most (but not all) of them have the stronger property that their axioms form a regular language. We will return to this in section 7.1.

There are theories whose obvious axiomatisation is semidecidable without

being decidable, but all the cases known to me are of the one same flavour. Here is one.

RCA_0 is a second-order theory of arithmetic. Two-sorted. It has Δ_1^0 comprehension (which says that $\{n \in \mathbb{N} : \phi\}$ exists as long as ϕ has no bound set variables) Σ_1^0 induction (which is the axiom scheme of mathematical induction for predicates that... erm ... Parameters are allowed.

A formula is Δ_1^0 if it is equivalent to both a Σ_1^0 formula and a Π_1^0 formula. So we want our axiom scheme of Δ_1^0 comprehension to be

If
 $\vdash (\forall x)(\theta(x) \longleftrightarrow (\exists y)(\phi(x, y)))$ and
 $\vdash (\forall x)(\theta(x) \longleftrightarrow (\forall y)(\psi(x, y)))$ then
 $(\exists y)(\forall x)(x \in y \longleftrightarrow \theta(x))$
 is an axiom.

This axiomatisation is clearly semidecidable and pretty clearly not decidable.

EXERCISE 56 (*)

Find a decidable axiomatisation of RCA_0 .

It turns out that the functions whose totality this theory can prove are precisely the primitive recursive functions.

REMARK 8 (Craig)

If T has a semidecidable set of axioms, then a decidable set of axioms can be found for it (in the same language).

Proof:

If T has a semidecidable set of axioms, then the set of its theorems is also semidecidable. This is because the relation of deducibility between (numbers of) formulæ is primitive recursive, so the set of theorems of T is the extension of a predicate $T(x)$ that informally says $(\exists y)(\exists z)(y \text{ is a run of a Turing machine that emits a list of (some) axioms of } T \text{ and } z \text{ is a derivation of } x \text{ from those axioms})$. Since the matrix of this formula is primitive recursive, and we can always code two naturals as one, this is of the form $(\exists w)(\phi)$, where ϕ is primitive recursive—and the extensions of all such predicates are semidecidable sets. So the set of theorems of T is a semidecidable set, just like the set of theorems of a theory with a *decidable* axiomatisation! This, indeed, is the burden of remark 6.

By assumption, T has a semidecidable axiomatisation. By remark 6, it follows that it has a semidecidable set of theorems. That is to say, there is a Turing machine \mathfrak{M} whose volcano emits theorems of T (as in remark 5). What we would like to do—when, for the n th time, \mathfrak{M} 's volcano emits a formula—is to label that n th formula with the natural number n , for then we would have a decidable set of ordered pairs of formulæ-with-numbers. The decision procedure for this set of ordered pairs is simplicity itself. Given a candidate pair, we look at the second component—which will be a natural number, n , say. We run \mathfrak{M} 's volcano until it has emitted n formulæ and we check to see whether the n th

get the definition straight

formula is the first component. If it is, we accept the pair; if it isn't, we reject it.

However, what we are after is not a decidable set of pairs $\langle \text{formula}, \text{natural number} \rangle$ but a decidable set of [gnumbers of] naked formulæ, indeed a decidable set of formulæ that axiomatises T . So we have to find a way of coding up pairs $\langle \text{formula}, \text{natural number} \rangle$ as formulæ. There are various ways of doing it, but the following is the best one I know.

Let ϕ_i be the i th formula emitted by \mathfrak{M} 's volcano. Then our n th axiom, Φ_n is

$$\phi_0 \rightarrow (\phi_1 \rightarrow (\phi_2 \rightarrow \dots \phi_n) \dots)^{17}$$

The set of all Φ_n is clearly a decidable set of axioms for T . Observe that the characteristic function of this set is not only total computable but actually primitive recursive.

Miniexercise: Is this axiomatisation *independent*? [See exercise 9 on sheet 2 of PTJ's set theory and Logic course in 2011-12]

update tis reference

■

If a theory has a semidecidable set of axioms then in some sense it has finite character, and remark 8 captures part of this sense by telling us it will have a decidable set of axioms. In both these descriptions the finite character is expressed in a metalanguage. The following remark tells us that this finite character can be expressed in a language for T .

REMARK 9 (Kleene, [38])

If T is a recursively axiomatisable theory in a language \mathcal{L} with only infinite models, then there is a language $\mathcal{L}' \supseteq \mathcal{L}$ and a theory T' in \mathcal{L}' and T' is finitely axiomatisable and is a conservative extension of T .

Proof: Omitted.

■¹⁸

They don't know what a conservative extension is, so you'll have to tell them.

(We can uniformly expand any \mathcal{L} -structure that is a model of T into a \mathcal{L}' -structure that is a model of T' .)

(The idea in the proof is to formalize the inductive clauses of the truth definition for T . The basic references are [38] and [14]. There is a very clear review of both papers by Makkai [41] that also provides a sketch of the proof.) You will have seen some examples of this phenomenon in Part II *Logic and Set Theory* last year. Bipartite graphs, algebraically closed fields... Another illustration of this process is afforded by the way in which the (pure) set theory ZF (which cannot be finitely axiomatised) corresponds to the class theory NBG, which can be finitely axiomatised. Why would one expect this to be true in general? A theory that is recursively axiomatisable is underpinned by a finite engine that generates all the axioms. It ought to be possible to hard-code this engine into the syntax, if necessary by enlarging the language. I have the feeling that it should be possible to do this without invoking truth-definitions.

Might it be easier to prove Kleene's theorem for automatic theories than for arbitrary recursively axiomatisable theories...?

¹⁷Brief reality check: if a formula is of this form, then we can recover the ϕ_i .

¹⁸I am omitting the proof since I cannot find a proof that doesn't use truth-definitions, and I haven't got time or space to go into them.

EXERCISE 57 Since propositional logic is decidable, the set of falsifiable propositional formulae over an alphabet is semidecidable, so it is a rectype. Give a presentation.¹⁹

EXERCISE 58

Suppose $f : \mathbb{N} \rightarrow \mathbb{N}$ is computable.

Show that there a computable partial function g s.t. $(\forall n \in \mathbb{N})(f(n) \downarrow \rightarrow f_{g(n)}(n) \downarrow)$.

Suppose further that $\{n : f(n) \downarrow\}$ is semidecidable but not decidable.

Show that there is no computable total function g such that $(\forall n \in \mathbb{N})(f(n) \downarrow \rightarrow f_{g(n)}(n) \downarrow)$.

5.6 The Undecidability of the Halting Problem

The set of register machine programs is countable because of the prime powers trick. The set of all subsets of \mathbb{N} is not, because of Cantor's theorem. There simply are not enough register machine programs to go round: inevitably some subsets of \mathbb{N} are going to be undecidable. In fact, *almost all* of them are, in the sense that there is the same number of subsets of \mathbb{N} as there are undecidable subsets. This argument is nonconstructive and does not actually exhibit a subset of \mathbb{N} that is not decidable, but we can do that too.

Suppose we had a machine \mathcal{M} that, on being given a natural number n , decoded it (using the primitive recursive unpairing functions alluded to on page 53) into **fst** n and **snd** n (n_1 and n_2 for short), and then $\downarrow = 0$ if the n_1 th machine halts when given input n_2 and $\downarrow = 1$ otherwise.

We can tweak this machine (by using something to trap the output) to get a machine \mathcal{M}^{\otimes} with the following behaviour: on being given n , it decodes it into n_1 and n_2 (**fst** and **snd** of n) and then $\downarrow = 1$ if the n_1 th machine diverges on input n_2 (just as before) but diverges if the n_1 th machine halts when given input n_2 .

Front-end onto *this* machine a machine that accepts an input x and outputs **pair**(x, x). We now have a machine $\mathcal{M}^{\otimes \otimes}$ with the following behaviour.

On being given n , it tests to see whether or not the n th machine halts with input n . If it does, it goes into an infinite loop (diverges); if not, it halts with output 1.

This machine is the n_0 th, say. What happens if we give it n_0 as input? Does it halt? Well, it halts iff the n_0 th machine loops when given input n_0 . But it is the n_0 th machine itself!

¹⁹I have to confess that I have no model answer to this! I thought I had a reference in the literature but that was to a paper which shows that the set of *negations* of tautologies is axiomatisable. But that's obvious!

wrapping again

Formally we can write $\{n_0\}(n_0)\downarrow$ iff (by definition of $\{n_0\}$) $\{n_0\}(n_0)\uparrow$. Notice the similarity with the proof of Cantor’s theorem (and, later, the incompleteness theorem, theorem 19)

What assumption can we discard to escape from this contradiction? Clearly we cannot discard the two steps that involve just trapping output and front-ending something innocent onto the hypothesised initial machine. The culprit can only be that hypothesised machine itself!

So we have proved

THEOREM 12 *The set of numbers $\{\text{pair}(p, d) : \{p\}(d)\downarrow\}$ is²⁰ not decidable.*



Though it is obviously *semidecidable*! ■

There is an aspect of this that often bothers beginners. We assumed that \mathfrak{M} solved the Halting problem and we then exhibited—on that assumption—one (only one!) instance of the halting problem that \mathfrak{M} couldn’t solve. One might think that all one had to do was modify \mathfrak{M} so that the first thing it did was check for that one case. That doesn’t work—because that modification changes \mathfrak{M} ’s gnumber: the target is not stationary! In fact any \mathfrak{M} that aspires to solve the halting problem must give *infinitely many* wrong answers. Any finite amount of tweaking can be hard-coded so if *per impossibile* we got our hands on a machine that made only finitely many mistakes we could (by wrapping) obtain one that made no mistakes at all. And that, as we have just shown, we can not have.

We saw earlier that many non-total functions can be encoded by primitive recursive functions, leaving open the possibility that all computable functions (even those that are not total) are in some sense primitive recursive. We saw that not every total computable function is primitive recursive. But might it still be the case that every computable partial function can be analogously encoded by a total computable function? No. The partial “evaluation” function $\langle x, y \rangle \mapsto \{x\}(y)$ cannot be encoded by any total computable function. If it were so encoded we would be able to solve the halting problem. The “evaluation” function is irreducibly and inescapably non-total.

5.6.1 Rice’s Theorem

Theorems 18 and 12 are manifestations of a general phenomenon, and in this section we examine that phenomenon. Its canonical expression is Rice’s theorem. (Though theorem 18 is not exactly a special case of Rice’s theorem but something a little bit more.) We prove a number of results *en route* to Rice’s theorem.

THEOREM 13 (*The “S-m-n theorem”*)

There is a computable total function S of two variables such that, for all e , b and a ,

$$\{e\}(b, a) = \{S(e, b)\}(a),$$

and so on for higher degrees (more parameters).

²⁰I trust the overloading of the curly brackets does not wrong-foot the reader ... (!)

(That is to say: currying, thought of as a function from gnumbers of functions to gnumber of functions, is computable.)

This is a corollary of the equality between μ -recursiveness and computability by register machines: one can easily tweak a machine for computing $\lambda ab.\{e\}(a, b)$ into a machine that, on being given a , outputs a description of a machine to compute $\lambda b.\{e\}(a, b)$.

In turn we get a corollary:



COROLLARY 2 (*The fixed point theorem*).

Let $h : \mathbb{N} \rightarrow \mathbb{N}$ be a total computable function. Then there is n such that $\{n\} = \{h(n)\}$ ²¹

Proof: Consider the map

$$\text{pair}(e, x) \mapsto \{h(S(e, e))\}(x).$$

This is computable and is therefore computed by the a th machine, for some a . Set $n = S(a, a)$. Then

$$\{n\}(x) =^1 \{S(a, a)\}(x) =^2 \{a\}(a, x) =^3 \{h(S(a, a))\}(x) =^4 \{h(n)\}(x)$$

- (1) holds because $n = S(a, a)$;
- (2) holds by definition of S ;
- (3) holds by definition of a and
- (4) holds by definition of n .

On Sun, 29 Apr 2012, Zhen Lin Low wrote:

Dear Dr Forster,

If I'm not mistaken, the proof of corollary 2 is almost exactly the recursion-theoretic translation of the Y combinator. To be precise, it corresponds to the combinator

$$Y' = [\lambda h. [\lambda xy. h(xx)y] [\lambda xy. h(xx)y]]$$

which η -reduces to the usual Y combinator.

Best wishes,

Zhen Lin

There is a powerful corollary of this that is a sort of omnibus undecidability theorem.



THEOREM 14 (*Rice's theorem*)

Let A be a nonempty proper subset of the set of graphs of all computable functions of one variable. That is to say: A is a set of functions-in-extension. Then $\{n : \text{Graph}(\{n\}) \in A\}$ is not decidable.

²¹In case you are wondering, we of course mean the *graphs* of these two functions are equal

Proof: Suppose χ , the characteristic function of $\{n : \text{Graph}(\{n\}) \in A\}$ is computable; we will deduce a contradiction.

Find naturals a and b so that $\text{Graph}(\{a\}) \in A$ and $\text{Graph}(\{b\}) \notin A$. If χ is computable the following function will also be computable:

$$g(n) := \text{if } \text{Graph}(\{n\}) \in A \text{ then } b \text{ else } a$$

(“wrong way round”!). By corollary 2 there must now be a number n such that $\{n\} = \{g(n)\}$. We also need to minute the fact that g swaps a and b .

Is $\text{Graph}(\{n\})$ in A ? Let’s assume it is, and derive a contradiction.

| | |
|--------------------------------|---|
| $\text{Graph}(\{n\}) \in A$ | 1) |
| $\text{Graph}(\{g(n)\}) \in A$ | 2) because $\{n\} = \{g(n)\}$ |
| $g(n) = b$ | 3) from (1) and definition of g |
| $g(g(n)) = b$ | 4) from (2) and definition of g |
| $g(b) = b$ | 5) from (4), substituting b for $g(n)$ (by (3)). |

...contradicting the fact that g swaps a and b .

But $\text{Graph}(\{n\}) \notin A$ also leads to contradiction:

| | |
|-----------------------------------|--|
| $\text{Graph}(\{n\}) \notin A$ | 6) |
| $\text{Graph}(\{g(n)\}) \notin A$ | 7) because $\{n\} = \{g(n)\}$ |
| $g(n) = a$ | 8) from (6) and definition of g |
| $g(g(n)) = a$ | 9) from (7) and definition of g |
| $g(a) = a$ | 10) from (9), substituting a for $g(n)$ (by (8)). |

...again contradicting the fact that g swaps a and b . ■

(We can probably leave out ‘Graph’ beco’s ‘ $\{n\} = \{m\}$ ’ means that the two functions-in-intension have the same graph, but i think it helps to leave it in. Rams the point home.)

This theorem is very deep and very important, but the moral it brings is very easy to grasp. It tells us that we can never find algorithms to answer questions about the *behaviour* of programs (“Does it halt on this input?”; “Does it always emit even numbers when it does halt?”) on the basis of information purely about the *syntax* of programs (“Every variable occurs an even number of times”). In general, if you want to know anything about the behaviour of a program, you may be lucky and succeed in the short term and in a small number of cases, but in the long run you cannot do better than by just running it.

In particular it has the consequence that it is not decidable whether or not two programs compute the same function(-in-extension). This makes it particularly important to bear in mind that the theory of computable functions

is in the first instance a study of function declarations (functions-in-intension) rather than function graphs.

One can also express this insight by saying: *syntax is intensional, behaviour is extensional. And extensions are undecidable.* (which sounds the wrong way round ...)

EXERCISE 59 Suppose that f is a μ -recursive function of two variables.

(i) Show that there is a μ -recursive function g of one variable such that for each m , if $(\exists n)(f(m, n) = 0)$, then $f(m, g(m)) = 0$.

(ii) Show that it is not always possible to choose g so that $g(m)$ is the least n with $f(m, n) = 0$.

5.7 Recursive Inseparability

Two disjoint sets X and Y are said to be *recursively inseparable* if there is no decidable set Z with $X \subseteq Z$ and $Z \cap Y = \emptyset$. (The idea of separable/inseparable comes from descriptive set theory).

REMARK 10 The two sets

$$A = \{e : \{e\}(e) \downarrow > 0\} \text{ and}$$

$$B = \{e : \{e\}(e) \downarrow = 0\}$$

are recursively inseparable. That is to say, if f is a total function with $f \restriction A = \{0\}$ and $f \restriction B = \{1\}$, then f is not computable.

Proof:

Consider any $n \in \mathbb{N}$; we will show that $\{n\}$ is not an f as in the statement of the remark.

If $\{n\}(n) \uparrow$ then clearly $\{n\} \neq f$, because f is total

If $\{n\}(n) \downarrow = 0$ then $n \in B$ so we must have $f(n) = 1$. But $\{n\}(n) = 0$ so clearly $\{n\} \neq f$.

If $\{n\}(n) \downarrow > 0$ then $n \in A$ so we must have $f(n) = 0$. But $\{n\}(n) > 0$ so, again, $\{n\} \neq f$.

■

And here is another result with wider ramifications.

DEFINITION 17 If \sim is an equivalence relation on a set A we say f is a **classifier** for \sim iff $(\forall x, y \in A)(f(x) = f(y) \iff x \sim y)$.

EXERCISE 60 Show that if \sim is a decidable equivalence relation on \mathbb{N} , then there is a computable classifier for it.

You might have expected that further if \sim is a decidable equivalence relation on a subset of \mathbb{N} (so that there is a computable two-valued function g defined on pairs from that subset such that if x and y are both in that subset then $g(x, y) = 0$ iff $x \sim y$) then there is a computable partial function f that classifies \sim . Remarkably this is not true.

THEOREM 15 *There is a decidable equivalence relation on a subset of \mathbb{N} that has no computable classifier.*



Proof:

Consider the relation R that is the reflexive symmetric closure of $\{\langle 3n, 3n+1 \rangle : n \in \mathbb{N}\} \cup \{\langle 3n, 3n+2 \rangle : n \in \mathbb{N}\}$. Its graph looks like lots of isolated paths of length 2. It's not transitive, but whenever A is a subset of \mathbb{N} s.t. $|A \cap \{3n, 3n+1, 3n+2\}| < 3$ for all n , then the restriction $R|_A$ is transitive, and is therefore an equivalence relation. We will cook up such an infinite set A .

For each $n \in \mathbb{N}$, A_n will be a one-or-two-membered subset of $\{3n, 3n+1, 3n+2\}$. A will be the union of the A_n , which we define as follows.

If the function $\{n\}^\dagger$ on any of $\{3n, 3n+1, 3n+2\}$ set A_n to be the singleton of the smallest such. Otherwise...

If $\{n\}(3n) \neq \{n\}(3n+1)$ set $A_n := \{3n, 3n+1\}$;

If $\{n\}(3n) = \{n\}(3n+1) \neq \{n\}(3n+2)$ set $A_n := \{3n, 3n+2\}$;

If $\{n\}(3n) = \{n\}(3n+1) = \{n\}(3n+2)$ set $A_n := \{3n+1, 3n+2\}$.

Evidently the restriction of R to A is an equivalence relation, and it's easy to see that this equivalence relation is decidable. However the construction of the A_n is a diagonal construction that ensures, for each n , that the function $\{n\}$ does not classify R .

Observe that A is not (apparently) semidecidable. What about $\mathbb{N} \setminus A$?

See [66], section 3.2.32. (pages 153–154).

Contrast this with the 2013 tripos question exercise 62 which follows.

5.8 Exercises

EXERCISE 61

The relational product (see p. 52) of two primitive recursive relations might not be a primitive recursive relation.

EXERCISE 62 (Part III Paper 20 2013)(*)

A transversal for a family \mathcal{X} of pairwise disjoint subsets of a set X is a subset X' of X s.t. $|X' \cap x| = 1$ for all $x \in \mathcal{X}$.

Let \sim be an equivalence relation on \mathbb{N} , of infinite index, whose complement is semidecidable (considered as a subset of $\mathbb{N} \times \mathbb{N}$). Show that there is a semidecidable transversal on the set of \sim -equivalence classes.

EXERCISE 63 (*)

Suppose $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is total computable and increasing: $f(\vec{x}) > \max(\vec{x})$. Show that there is a decidable $A \subseteq \mathbb{N}$ satisfying $f^{\omega} A = \mathbb{N} \setminus A$.

EXERCISE 64 (*)

Show that for any corruptible operating system there can be no program IS-SAFE that, when given program p and data d , says “yes” if p applied to d does not corrupt the operating system and “no” otherwise.

EXERCISE 65 (Mathematics Tripos IIA 1997 Paper 3 Question 8)(*)

Does there exist a computable function f such that, for all m and n , if the m th register machine program halts with input n , then it does so in at most $f(m, n)$ steps? Does the answer change if f is required to be total?

EXERCISE 66 (*)

Suppose that ϕ is a computable partial function of two arguments.

1. Show that there is a computable partial function ψ of one argument such that, for each m , if there are x with $\phi(x, m) = 0$, then $\phi(\psi(m), m) = 0$. If there are no such x , is your $\psi(m)$ defined?
2. Show that it is not always possible to take $\psi(m) = \mu x.(\phi(x, m) = 0)$.

EXERCISE 67

This is an old example sheet question from Professor Johnstone. I know nothing about it: use at your own risk. You may like to read the Wikipædia article on this subject.

The “programming language” **FRACTRAN**, invented by J. H. Conway, has “programs” which are finite lists (q_1, q_2, \dots, q_k) of positive rational numbers. Such a program accepts as input a positive integer n : a single step of the program replaces n by $q_i \cdot n$ for the least i such that $q_i \cdot n$ is an integer, if this is possible; if no $q_i \cdot n$ is an integer, the program terminates. [Thus, for example, the program $(\frac{2}{3}, \frac{3}{4}, 2)$ means ‘replace n by $2n/3$ if it’s a multiple of 3, by $3n/4$ if it’s a multiple of 4 but not of 3, and by $2n$ otherwise’; note that in this case, since the last number in the list is an integer, the program will run for ever.]

Show that, for any (unary) recursive function f , there is a **FRACTRAN** program which, given an input of the form 2^n , will reach $2^{f(n)}$ (if $f(n)$ is defined) before it reaches any other power of 2, and will never reach a power of 2 if $f(n)$ is undefined.

[Hint: show that the action of a register machine program can be simulated by a **FRACTRAN** program; you will need to use powers of distinct primes to represent not only the contents of the registers, but also the numbers of the states in the register machine program.]

Describe the behaviour of the **FRACTRAN** program

$$\left(\frac{3}{5}, \frac{1250}{567}, \frac{1}{81}, \frac{875}{297}, \frac{5}{27}, \frac{3025}{63}, \frac{125}{9}, \frac{25}{6}, \frac{625}{3}, 35\right).$$

[Explicitly, if the register machine program uses r registers and has $s + 1$ states including the terminal state, represent the situation when it is in state j and has n_i in R_i for each $i \leq r$ by the number $p_1^{n_1} p_2^{n_2} \cdots p_r^{n_r} p_{r+j}$, where p_t is the t^{th} prime number. The instruction $(i, +, k)$ corresponds to the fraction $p_i p_{r+k} / p_{r+j}$, and $(i, -, k, l)$ to the pair of fractions $p_{r+k} / p_i p_{r+j}, p_{r+l} / p_{r+j}$ (in that order), with slight modifications to cope with initial and terminal states. The given example uses a different coding, in which powers of the two primes 3 and 5 are used to represent the state numbers; it computes the function $n \mapsto 2^n$.]

EXERCISE 68 (*) For which of the following functions-in-intension are there computable functions-in-intension with the same extension?

1. λx . if there is somewhere in the decimal expansion of π a string of exactly x 7's, then 0, else 1;
2. λx . if there is somewhere in the decimal expansion of π a string of at least x 7's, then 0, else 1;
3. λk . the least n such that all but finitely many natural numbers are the sum of at most n k th powers.

EXERCISE 69

Show that the graph of a total computable function $f : \mathbb{N}^n \rightarrow \mathbb{N}$ is a decidable subset of \mathbb{N}^{n+1} . Is the graph of a partial computable function decidable?

EXERCISE 70

1. Let ψ be the partial computable function defined by $\psi(x) = \mu y. \{x\}_y(x) \downarrow$. Show that for any total computable function f there is an n with $f(n) < \psi(n)$. Deduce that ψ cannot be extended to a total computable function. You may assume that a coding of tuples is in use according to which any number coding a tuple is bigger than all of the numbers in the tuple that it codes.
2. Show directly in the manner of the proof of the undecidability of the Halting Problem that the following sets are not decidable:
 - (i) $\{e : |\{e\} \text{ "IN"}| = \aleph_0\}$;
 - (ii) $\{e : \{e\} \text{ "IN"} \text{ strictly contains } \text{Dom}(\{e\})\}$.

EXERCISE 71 (*)

A **box of tiles** is a set of rectangular tiles, all of the same size. The tiles have an orientation (top and bottom, left and right) and the edges have colours. The idea is to use the tiles in the box to tile the plane, subject to rules about which colours can be placed adjacent to which, and each box comes with such a set of rules. (Naturally every set of rules includes all the obvious things, like, a bottom edge can only go next to a top edge, and so on.) So of course the box has infinitely many tiles in it. Nevertheless, the tiles in each box come in only

finitely many varieties. (It is a bit like a scrabble set: only 27 letters but many tokens of each.)

With some boxes one can tile the plane; with some one cannot. Sketch how to number boxes and explain why the set of numbers of boxes that cannot tile the plane is a semidecidable set.

Hint: you will need König's Infinity Lemma.

EXERCISE 72

Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a strictly order-preserving total computable function. Construct a semidecidable subset A of \mathbb{N} such that

(i) for all e , if the domain $\text{Dom}(\{e\})$ of the e th partial computable function is infinite, then $\text{Dom}(\{e\}) \cap A \neq \emptyset$ and

(ii) there are at most e elements less than $f(e)$.

Deduce that there is a semidecidable set B such that $\mathbb{N} \setminus B$ is infinite and contains no infinite semidecidable subset.

EXERCISE 73

Is it possible to decide, given that $\{e\}$ is total, whether or not

1. $(\forall n)(\{e\}(n) = 0)$?

2. $(\exists n)(\{e\}(n) \leq \{e\}(n+1))$?

3. $(\exists n)(\{e\}(n) \geq \{e\}(n+1))$?

EXERCISE 74 ("A couple of weird applications of the fixed point or second recursion theorem" says Prof Hyland, the author of this question.)

Show that there is an $e \in \mathbb{N}$ such that $\text{Dom}(\{e\}) = \{x : x > e\}$.

Show that there is an $e \in \mathbb{N}$ such that $\text{Dom}(\{e\}) = \{x : \{x\}(e) \downarrow\}$.²² Any natural substitution function S will have $S(e, n) > e$ and $S(e, n) > n$ for all e and n . Deduce that, for any (partial) computable f , there are infinitely many e with $\{e\} = f$.

EXERCISE 75 Show that the following sets are not decidable.

(i) $\{e : \{e\} \text{ everywhere undefined}\}$. (ii) $\{e : \{e\} \text{ is total}\}$.

(iii) $\{e : \forall i < e. (\{e\}(i) \downarrow)\}$. (iv) $\{e : \forall i. (\{e\}(i) \downarrow \rightarrow i < e)\}$.

Which of the above sets R and their complements $\mathbb{N} \setminus R$ are semidecidable?

EXERCISE 76

Show that there is an $e \in \mathbb{N}$ such that $\text{dom}(\{e\}) = \{x : x > e\}$.

Show that there is an $e \in \mathbb{N}$ such that $\text{dom}(\{e\}) = \{x : \{x\}(e) \downarrow\}$.

In each case can one decide whether or not an index e is of the given kind?

EXERCISE 77 Suppose that $f, g : \mathbb{N}^2 \rightarrow \mathbb{N}$ are total computable. Show that there exist i, j with $\{i\} = \{f(i, j)\}$ and $\{j\} = \{g(i, j)\}$. [Hint: Show first that there is a total computable h with $\{h(i)\} = \{g(i, h(i))\}$.] (Hard)

²²My 2011/2 cohort tell me that this isn't a straightforward application of Rice's theorem, and neither they nor I know how to do it.

EXERCISE 78 (*)

1. Show that the range of an increasing total function $f : \mathbb{N} \rightarrow \mathbb{N}$ is a decidable set, and
2. conversely, that every decidable subset of \mathbb{N} is the range of an increasing total computable function $\mathbb{N} \rightarrow \mathbb{N}$.
3. What if f is merely nondecreasing (but still total)?
4. What if f is increasing but perhaps not everywhere defined? (i.e., $(\forall n)(\forall m)((n < m) \wedge f(n) \downarrow \wedge f(m) \downarrow) \rightarrow f(n) < f(m)$)?
5. What is the notion of “increasing function $\mathbb{N} \rightarrow \mathbb{N}^n$ ” that one would need were one to prove that every decidable subset of \mathbb{N}^n is the range of an increasing computable function $\mathbb{N} \rightarrow \mathbb{N}^n$?

EXERCISE 79 (*)

Show that every infinite semidecidable set has an infinite decidable subset.

EXERCISE 80 (*) (Part III Computability and Logic examination 2014)

Let $<_1$ and $<_2$ be recursive (decidable sets of ordered pairs) dense linear orderings of \mathbb{N} without endpoints. There are isomorphisms between $\langle \mathbb{N}, <_1 \rangle$ and $\langle \mathbb{N}, <_2 \rangle$. Are any of them recursive?

EXERCISE 81 (*)

Show that, for any partial computable function ψ not everywhere undefined, there is an index e with $\{e\} = \psi$ and such that, for some $n < e$, $\{e\}_e(n) \downarrow$.

Deduce that there is a recursive enumeration ψ_e of the partial computable functions such that $\psi_0 = \perp$ and such that for all $n > 0$, $\psi_n \neq \perp$. Why is this certainly not an acceptable enumeration?²³

By considering enumerations of the partial computable functions, find a partial computable function that cannot be extended to a total computable function.

EXERCISE 82 (*)

For a finite family (A_0, A_1, \dots, A_n) of subsets of \mathbb{N} , show that the following conditions are equivalent:

- (i) There exists a partial recursive function f of two variables such that $f(x, y) = 0$ whenever x and y belong to the same set A_i , but $f(x, y) = 1$ if $x \in A_i$ and $y \in A_j$ for some $i \neq j$.
- (ii) There exists a partial recursive function g of one variable, which takes distinct constant values on each of the A_i .

Such a family of sets is called recursively separated. Give an example of a recursively separated pair of sets (A_0, A_1) which cannot be separated (in either of the above senses) by a total recursive function (equivalently, such that there is no recursive set containing A_0 and disjoint from A_1).

²³See p 75 for definition of “acceptable”.

EXERCISE 83 [marked by PTJ as HARD]

A set $A \subseteq \mathbb{N}$ is called Diophantine if there exists a polynomial $p(x, y_1, \dots, y_n)$ with integer coefficients such that $x \in A$ if and only if there exist y_1, \dots, y_n such that $p(x, y_1, \dots, y_n) = 0$. Show that any Diophantine set is semi-recursive. [A famous result due to Yu. Matiyasevich asserts that the converse is true.] Show also that a set is Diophantine if and only if it is the set of non-negative values taken by some polynomial with integer coefficients.

EXERCISE 84 (*)

Explain what a model of a sentence is. If Φ is a sentence the **spectrum** of Φ is the set of $n \in \mathbb{N}$ such that Φ has a model of size n . Is every spectrum decidable? Use a diagonal argument to find a decidable set that is not a spectrum.

(I've mislaid my answer to this one!!!)

Chapter 6

Representability by λ -terms

Best local thing to read is [46]. I'm going to cover only as much λ -calculus as is needed to show the principal connections to computability. This is not a course on the λ calculus.

6.1 Some λ -calculus

Must say a bit about the Curry-Howard correspondence here. And typing.

β -reduction, α -conversion, η -reduction. An expression is in normal form if there are no β -redexes—a β -redex being something to which you can do a β -reduction.

The only uniformly definable function $A \rightarrow A$ is the identity $\mathbb{1}_A$. Any definable function $A \rightarrow A$ must commute with all permutations of A , and we all know that the centre of a symmetric group S is $\{\mathbb{1}_S\}$!

EXERCISE 85

By using Curry-Howard on a two-membered set B with a five membered superset A of it, or otherwise, show that Peirce's Law: $((A \rightarrow B) \rightarrow A) \rightarrow A$ is not a constructive thesis.

6.2 Arithmetic with Church Numerals

K and S . I know they sound like spymasters but they aren't. (*Karla and Smiley...*?)

0 is K of the identity. Iterators. They are all typed.

suc $\lambda n.\lambda f.\lambda x.f(nfx)$

plus $\lambda n.\lambda m\lambda f.\lambda x.mf(nfx)$

times $\lambda n.\lambda m\lambda f.\lambda x.m(nf)x$
 which η -reduces to
 $\lambda n.\lambda m\lambda f.m(nf)$

exp $\lambda n.\lambda m\lambda f.\lambda x.mnfx$
 which η -reduces first to
 $\lambda n.\lambda m\lambda f.mnf$
 and then to
 $\lambda n.\lambda m.mn$

In lambda-talk the Church numerals are often written with underlinings: \underline{n} . This overloads the notation on p. 42 but doesn't actually violate its spirit.

In this development we will trade heavily on the *aperçu* that a function $\mathbb{N}^k \rightarrow \mathbb{N}$ defined by recursion can always be thought of as a fixed point for a function $(\mathbb{N}^k \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^k \rightarrow \mathbb{N})$.

Fixed point combinators.

$$Y : \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)).$$

$$(\lambda x.f(xx))(\lambda x.f(xx)) \rightarrow f((\lambda x.f(xx))(\lambda x.f(xx))).$$

Somewhat to my surprise I learnt recently (from [28]) that the set of [gnumbers of] λ -terms that are fixed-point operators is semidecidable. This seems so striking that I supply a proof. [How can one possibly detect in finite time that, for all t , ϕt and $t(\phi t)$ have a common $\beta - \eta$ reduct??] The point is that ϕ is a fixed-point combinator iff ϕ and $\lambda x.x(\phi x)$ have a common $\beta - \eta$ reduct, and *that* fact can be detected in finite time. Presumably the set is not actually decidable, but I know no proof.

(Zhen Lin has pointed out to me that Y is lurking inside the proof of corollary 2. The moral of this is presumably that there are as many proofs of corollary 2 as there are fixed-point combinators.)

Talk a bit about the λ term for $(A \rightarrow B) \rightarrow A. \rightarrow (A \rightarrow B) \rightarrow B$.

Recall that the class of primitive recursive functions is closed under **if-then-else**, so we'd better have a lambda version of this construct.

$\lambda xy.x$ works like **true** and

$\lambda xy.y$ works like **false**.

EXERCISE 86 (*)

true is of type $A \rightarrow (B \rightarrow A)$ and **false** is of type $A \rightarrow (B \rightarrow B)$, so they are both of type $A \rightarrow (A \rightarrow A)$.

Show that they are the only definable functions of this type.

Now we can set: `if b then x else y` is $\lambda bxy.bxy$.

The point being that if b is a boolean it must have a normal form that is one of `{true, false}`... that is to say, one of $\{\lambda xy.x, \lambda xy.y\}$.

`iszero := $\lambda n.n(\lambda x.false)$ true`

`iszero x` evaluates to `true` or to `false` depending on whether or not x evaluates to (church numeral) 0:

Once we have `pred` (which is primitive recursive as we saw earlier) we can test for equality with other numerals.

We also need pairing and unpairing:

`pair := $\lambda xyf.fxy$`
`fst := $\lambda p.p \text{ true}$`
`snd := $\lambda p.p \text{ false}$`
`nil := $\lambda x.true$`

Check that `fst pair x y = x` and `snd pair x y = y`:

`fst pair x y =`
 $(\lambda p.p \text{ true})(\lambda f.fxy)$
 $(\lambda f.fxy)\text{true} =$
`true x y =`
 x

and `snd pair x y = y` similarly.

EXERCISE 87 *What are the types of these expressions? Discuss.*

Lists can be thought of as ordered pair of head and tail, so `fst` and `snd` double up as `hd` and `tl`; `pair x y` doubles up as $x::y$, and `nil` is the empty list.

`NULL := $\lambda p.p(\lambda xy.false)$`

`NULL` tests for the empty list. There is probably an EOF (end-of-file) character but we won't need it: it will be quite useful to us to have lists that are of infinite length. CompSci call things of this data type *streams*.

Another way of doing lists

Here is another way of thinking of [finite] lists. If I have

- (i) an α -list l ,
- (ii) a β and
- (iii) a function $f : \alpha \times \beta \rightarrow \beta$

then I can take the pair of the head of l and the β , and whack it with f to obtain another β —which I then pair with the second member of l , whack *that* with f to obtain a *third* β and so on until I exhaust the α -list. Thus anything that takes (i)–(iii) and gives back a β can be thought of as an α -list. So α -list is of type

$$(\forall \alpha \beta)(\beta \rightarrow (\alpha \times \beta \rightarrow \beta) \rightarrow \beta)$$

The empty list is thus $\lambda x \beta. \lambda f_{(\alpha \times \beta \rightarrow \beta)}. x$ which is to say: K .

`cons` and `hd` and `tl` will appear sooon. Watch this space.

A message from Toby Miller ...

A naïve approach to choosing a type for lists in the Polymorphic Lambda Calculus (hereafter *PLC*) would be to consider an α *list* as being an ordered pair $\alpha * (\alpha \text{ list})$. This doesn't work because *Nil* cannot be encoded like this, and also because we have just defined a recursive type, which isn't allowed in PLC.

Ordered pairs can be encoded in PLC thus:

$$\begin{aligned} \alpha * \beta &=_{\text{def}} \forall \gamma ((\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \gamma) \\ \text{makePair} &=_{\text{def}} \Lambda \alpha, \beta (\lambda x : \alpha, x' : \beta (\Lambda \gamma (\lambda f : \alpha \rightarrow \beta \rightarrow \gamma (f x x')))) \\ \text{first} &=_{\text{def}} \Lambda \alpha, \beta (\lambda p : \alpha * \beta (p (\lambda x : \alpha, x' : \beta (x)))) \\ \text{second} &=_{\text{def}} \Lambda \alpha, \beta (\lambda p : \alpha * \beta (p (\lambda x : \alpha, x' : \beta (x')))) \end{aligned}$$

This demonstrates how one can have a PLC term capable of returning multiple, differently typed, values by using a polymorphic return type (here γ), and passing in a function that chooses which value to return.

To make something that works for lists we need to address the two issues described earlier. First we need some way to encode *Nil* using the same type as a `cons` cell. Secondly we cannot use the type $\alpha \text{ list}$ in its own definition, as PLC has no support for recursive types. We address these by re-imagining the concept of a list as, rather than a head and a tail, an object on which an iteration can be performed. The functions *head* and *tail* become secondary to the principal operation performed on lists: *listIter*. This function accepts a cumulative function, which is performed recursively on the elements of the list. If we pass the function $\lambda a : \text{int}, b : \text{int} (a + b)$ then we get the sum of all the elements of the list. This is similar to the ML fold functions. Functions to get the head and tail of a list can be built on top of this fold, although as we shall see, *tail* is far from elegant.

The list object itself is a polymorphic function which takes a base case, and a fold function, returning the result of running the fold on each list item in turn. *Nil* simply returns the base case, while *Cons* recurses on the next item in the list, and then returns the result of applying the fold function to the result, and its own list item. The *listIter* function essentially just wraps a call to the list itself.

$$\begin{aligned}
\alpha \text{ list} &=_{\text{def}} \forall \beta (\beta \rightarrow (\alpha \rightarrow \beta \rightarrow \beta) \rightarrow \beta) \\
\text{Nil} &=_{\text{def}} \Lambda \alpha, \beta (\lambda x : \beta, f : \alpha \rightarrow \beta \rightarrow \beta (x)) \\
\text{Cons} &=_{\text{def}} \Lambda \alpha (\lambda x : \alpha, l : \alpha \text{ list} (\Lambda \beta (\lambda x' : \beta, f : \alpha \rightarrow \beta \rightarrow \beta (f x (l \beta x' f))))) \\
\text{listIter} &=_{\text{def}} \Lambda \alpha, \beta (\lambda x : \beta, f : \alpha \rightarrow \beta \rightarrow \beta (\lambda l : \alpha \text{ list} (l \beta x f)))
\end{aligned}$$

In PLC, traversing a list using *head* and *tail* is not very easy, or useful. Most often one would want to pass a cumulative function over the entire list. An example for summing a list of integers is presented below. I assume that 0, + and *int* are defined. In practice one would use Church Numerals.

$$\text{sum} =_{\text{def}} \lambda l : \text{int list} (\text{listIter int int 0} (\lambda x : \text{int}, x' : \text{int} (x + x')) l)$$

Although we have established that *head* and *tail* are not easy to use in PLC, we can define them anyway. The main problem we face is how to deal with the base case. *Nil* has neither a head nor a tail, but by the type we have assigned it, it must return something of the correct type in both cases. We can allow the user to provide a ‘null’ value, which is returned in the case of *Nil*, and leave it to them to deal with the difficulties this presents. Therefore both *head* and *tail* will take a ‘null’ value and a list.

The *tail* function presents something of a challenge, because we are required to submit for iteration a function which reassembles the list as it goes. We could submit *Nil* as the base case, and *Cons* as the cumulative function, but then we would get the entire list back, rather than just the tail. The method I use here instead returns an ordered pair of two lists, the first being the complete list up to the point in question, and the second being the list from one iteration prior. The base case returns (*Nil* * *x*) (where *x* is the ‘null’). Subsequent cases shift the first value into the second, and uses a cons cell of the new head with the previous list for the first. The tail function itself just returns the second item in the pair. The pair is used as a delay mechanism, so that the list can be assembled normally, but with the previous version of the list still available.

$$\begin{aligned}
\text{head} &=_{\text{def}} \Lambda \alpha (\lambda x : \alpha, l : \alpha \text{ list} (\text{listIter } \alpha \alpha x (\lambda x' : \alpha, x'' : \alpha (x')) l)) \\
\text{tailIter} &=_{\text{def}} \lambda x' : \alpha, l' : (\alpha \text{ list} * \alpha \text{ list}) (\text{makePair} (\text{Cons } x' (\text{first } l')) (\text{first } l')) \\
\text{tail} &=_{\text{def}} \Lambda \alpha (\lambda x : \alpha \text{ list}, l : \alpha \text{ list} (\text{second} (\text{listIter } \alpha (\alpha \text{ list} * \alpha \text{ list}) (\text{makeList Nil } x) \text{tailIter } l)))
\end{aligned}$$

end of message from Toby

6.3 Representing the μ operator in λ -calculus

Must prove Church-Rosser

So far we have pairing and unpairing, if-then-else, and fixed point combinators to give us recursion, so clearly we can capture all of primitive recursion. Lists

will enable us to describe certificates. In fact we can even do partial computable functions as well, because lists enable us to present Kleene's T -function.

[I wrote this section as an exercise from first principles, for my own improvement—without looking it up—so you may well find in the literature a better way of doing it. On the other hand you might—as did I—find that reinventing the wheel turns out to be a character-forming experience.]

THEOREM 16 *Every μ -recursive function can be represented by a λ -term.*

Proof:

We need a lambda term to do what ML calls 'map': when l is a list and f a function defined on l 's entries then `map f l` returns the list whose n th entry is the f of the n th entry of the list l . It has the recursive definition

$$\text{map } f \ x = \text{if null}(x) \text{ then nil else } (f(\text{fst}(x)) :: (\text{map } f (\text{snd}(x))))$$

so it will be a fixed point for

$$\lambda m f l. (\text{if null}(l) \text{ then nil else } (f(\text{fst}(l)) :: (m \ f \ (\text{snd}(l)))))$$

namely

$$Y(\lambda m f l. (\text{if null}(l) \text{ then nil else } (f(\text{fst}(l)) :: (m \ f \ \text{snd}(l)))))$$

Next we need the stream of naturals—which we may as well call ' \mathbb{N} '—and which is a fixed point for $\lambda l. (0 :: (\text{map suc } l))$. Then the stream of values of f is just `map f \mathbb{N}` .

To finally obtain μ all we need is a way (on being given \underline{n}) of searching through the stream of values of f until we find one that is \underline{n} . How do we do that? Well, I wouldn't start from here, I'd start instead from over there, where—rather than having the stream of values of f —we have the stream of values of the function $n \mapsto \text{pair}(n, f(n))$ instead. (That is to say, the graph of f tho'rt of as a list of pairs). We then do `miaow` to it.¹

$$\text{miaow } \underline{n} \ x = \text{if } (\text{snd}(\text{hd } x) = \underline{n}) \text{ then } (\text{fst } (\text{hd } x)) \text{ else } \text{miaow } n \ (\text{tl } x).$$

If we apply `miaow \underline{n}` to the stream of pairs `pair(m , $f(m)$)` it returns the least m such that $f(m) = \underline{n}$. Pleasingly, if there is no pair in the stream whose second component is \underline{n} then the computation does not halt.

The above declaration of `miaow` is recursive, so we can obtain a λ -term for it by using a fixpoint combinator as usual. ■

Thus: every μ -recursive function can be represented by a λ -term. To complete the picture we need to prove the converse to theorem 16, that is to say, that every function that can be λ -represented is computable. A rigorous proof would be extremely laborious but the idea is very simple. If f is represented by

¹Joke! Joke!!

a λ -term L then L applied to the Church numeral \underline{n} will β -reduce to the Church numeral for $f(n)$. But this β -reduction is a deterministic process and can be captured by a Turing or Minsky machine. ■

It's worth thinking a bit about this because our lambda terms are nasty things arising from Y and it is possible to β -reduce them in such a way that they do not terminate. However it so happens that if you β -reduce from the top level down (so that, when confronted with $\lambda x.MN$, you turn it into $M[N/x]$ rather than β -reducing M or N) then if there is a normal form you will find it.

6.4 Typed Lambda terms for computable functions

This presentation of computable functions in λ -calculus exploits fixpoint combinators and gives us λ -terms that are not typed. The list gadgetry that we relied on (for example) is not well-typed. However, initially (for addition, multiplication and exponentiation) we had λ -terms that *were* typed. If we work a lot harder we can get typed λ -terms for a lot more functions. Paulson [46] supplies this λ -term for the Ackermann function:

$$\lambda m.m(\lambda f n.nf(f\mathbf{1}))\mathbf{succ}$$

There is some interesting mathematics tied up in the question of how much of computable function theory can be reproduced in typed fragments of λ -calculus. The answer can depend sensitively on the small print of the definition of ‘typed’. Here is system- T as presented in [27]:

Does any enthusiast feel like providing me with λ -terms for any of the Péter functions from exercise 43?

```
datatype num = 0
           | S of num

datatype bit = T | F

fun R u v 0 = u
  | R u v (S n) = v (R u v n) n

fun D u _ T = u
  | D _ v F = v

fun NOT u = D F T u
fun AND u v = D v F u
fun OR u v = D T v u

fun ADD x y = R x (fn z => fn z' => S z) y
fun PRED n = R 0 (fn x => fn x' => x') n
```

```

fun SUB x y = R x (fn z => fn z' => PRED z) y
fun MUL x y = R 0 (fn z => fn z' => ADD x z) y
fun EXP x y = R (S 0) (fn z => fn z' => MUL x z) y

fun ISZERO x = R T (fn z => fn z' => F) x

fun ITER f n = R (f (S 0)) (fn z => fn z' => f z) n
fun ACK m = R S (fn x => fn z' => ITER x) m

val r = ADD (S(S(S 0))) (S(S 0))

fun num 0 = 0
  | num n = S(num (n-1))

fun mun 0 = 0
  | mun (S n) = (mun n) + 1

val a = mun (ADD (num 3) (num 3))
val b = mun (MUL (num 3) (num 3))
val c = mun (EXP (num 3) (num 3))
val d = mun (ACK (num 3) (num 3))
val e = ISZERO (num 3)
val f = ISZERO 0

```

6.4.1 Combinators??

Prove that every λ term is equivalent to a combinator word in K and S .

Chapter 7

Recursive and Automatic Structures

I assume you know what a *structure* is from Part II. Any notion of computability gives rise to a corresponding notion of a computable structure. A computable structure will be one the graphs of whose decorations (predicates, functions etc) are computable in the sense of that notion.

We have considered two computability paradigms in this course: finite state machines and Turing machines. There are others of course (linear bounded automata, pushdown automata) but only those two in any detail, and the two notions of computable structure that they give rise to are the only kinds of computable structure we will consider.

Structures that are computable in the finite state machine sense are **automatic**¹ and those that are computable in the wider Turing-machine sense are **recursive**. The concept of *computable structure* arising from Turing-machine computability has been around much longer, and the automatic structures of the immediately following section are very recent.

We deal with these two notions in the following two sections, the stricter one first.

7.1 Automatic Structures

You might think that this notion of computability is too restrictive to be useful but you'd be wrong ... tho' admittedly it is only relatively recently that structures computable in the regular-language sense have attracted interest. [36] surveys the possibilities of applying this kind of analysis more generally, but we will concern ourself primarily with its use in group theory, since that is where all the action seems to be currently.

¹It might seem natural to say that the structure is *regular* but *automatic* is the word that seems to be used.

First, a connection with Turing machines and complexity classes. Everything solvable in linear time and constant space is automatic. Ashley Montanaro tells me: “Yes, the languages decidable in constant space are just the regular languages. Interestingly, apparently even the class of languages decidable in $o(\log \log n)$ space is still just the regular languages! see [63].”

The nicest application currently known to me of ideas of computable-by-finite-state-machine is the application to Group theory.

An *automatic group* is a finitely generated group equipped with several finite-state automata that represent the Cayley graph of the group, i. e. they can tell if a given word representation of a group element is in a “canonical form” and can tell if two elements given in canonical words differ by a generator. More precisely...

DEFINITION 18

Let G be a group and A be a finite set of generators. Then an automatic structure of G with respect to A is a set of finite-state automata:

- *the word-acceptor, which accepts, for every element of G , at least one word in A^* representing it;*
- *multipliers, one for each $a \in A \cup \{1_G\}$, which accept a pair $\langle w_1, w_2 \rangle$, for words w_i accepted by the word-acceptor, precisely when $w_1 a = w_2 \in G$.*

The property of being automatic does not depend on the set of generators.

It turns out that many natural classes of groups are automatic. Braid groups for example, and [word] hyperbolic groups. “Hyperbolic”?

The reader is assumed to know what a Cayley graph of a group is.

We can put a metric on a Cayley graph by deeming every edge to have length 1, and by this means we can give lengths to paths in the graph. The distance $d(x, y)$ between two vertices is the least number that is the length of a path between x and y . A path between x and y of length $d(x, y)$ is a **geodesic**. A triangle (of paths) is a **geodesic triangle** if its three sides are geodesics. A (geodesic) triangle is δ -thin as long as, for all x on the triangle, there is y on an edge of the triangle other than that containing x s.t. $d(x, y) \leq \delta$.

DEFINITION 19

A geodesic space is said to be δ -hyperbolic (in the sense of Gromov) if δ is a positive real and, given any geodesic triangle and any point on any side, the [least] distance from that point to [any point in] the union of the other two sides is bounded by δ .²

*It is just plain **hyperbolic** if there is a δ such that it is δ -hyperbolic.*

*A group G is said to be **word hyperbolic** if the Cayley graph $\Gamma(G, A)$ is hyperbolic.*

²This is known in Group Theorists’ slang as *The Skinny Triangle Property* as in: a space is hyperbolic if there is δ s.t. every geodesic triangle is δ -skinny.

(It can be shown that if X and Y are geodesic spaces and $f : X \rightarrow Y$ is a pseudoisometry, X is hyperbolic iff Y is. So whether or not a group is hyperbolic does not depend on a choice of generators. So definition 19 is legitimate.)

Clearly for any finite Cayley graph there is δ such that it is δ -thin: this idea is interesting only for infinite groups.

It turns out that, for every group G , if the Cayley graph for G under some presentation is hyperbolic, then the Cayley graph for any other presentation is also hyperbolic. This means that hyperbolicity is a property of the group not of any particular presentation.

EXERCISE 88 *Show that if the Cayley graph for G under some presentation is hyperbolic, then the Cayley graph for any other presentation is also hyperbolic.*

A good place to start reading about automatic groups would be Baumslag's review [5] of [21]. There is also a Wikipædia article.

EXERCISE 89 *Show that if G under some presentation is automatic, then it is automatic under any presentation.*

Free groups of finite rank are hyperbolic.

7.1.1 Automatic ordinals

Delhommé [17] has proved that ω^ω is the least ordinal not the length of an automatic wellorder of \mathbb{N} . (What does this mean, exactly?). Mathias [43] p 5 wonders whether this is anything to do with the fact that ω^ω is not "suitable" for the set theory S_0 . My guess is: not, but one never knows.

7.1.2 Automatic theories

For example the theory of fields of characteristic 0 is clearly automatic. Careful! Khossainov says that it *isn't* automatic because you have to put in the brackets. Clearly the situation is complicated. What if you use the LISP convention that you can close any number of '(' with a single ']'? The set of [unary notations for] primes is not a regular language, but we take our characteristic-0 scheme to be the scheme $nx \neq 1$ over *all* n not just *prime* n . (Exercise 44 asked the reader to show that, for any base b , the set of base- b notations for natural numbers is a regular language.) Algebraically closed fields? The algebraically closed scheme seems to me to need a PDA.

These could give us lots of exercises

EXERCISE 90 (*) *Show that the intersection of two automatic theories is automatic.*

7.2 Recursive structures

The definition of recursive structure is to be interpreted liberally, in various ways...

- A **recursive ordinal** is going to be the ordinal of a wellordering $\langle \mathbb{N}, < \rangle$ where $<$ is a wellordering of \mathbb{N} whose graph is decidable.
- We will have a good notion of a **recursive partition** of the set $[\mathbb{N}]^k$ of unordered k -tuples of natural numbers, since a partition of a set can be canonically identified with an equivalence relation on that same set, and we have a good notion of decidable equivalence relation.
- The carrier set does not have to be *literally* \mathbb{N} , but it must at least be a set that can be gnumbered.

A countable ordinal is an ordinal that is the length of a wellordering of \mathbb{N} or of a subset of \mathbb{N} —it makes no difference. Cantor called the set of countable ordinals the *Second Number Class* (the first number class is \mathbb{N}). A *recursive* ordinal is an ordinal that is the length of a *recursive* [= decidable] wellordering of \mathbb{N} or of a *recursive* [decidable] wellordering of a decidable subset of \mathbb{N} —it makes no difference: either way it's a wellordering whose graph (set of ordered pairs of natural numbers) is a recursive (= decidable) set. A decidable relation on a decidable infinite subset of \mathbb{N} is isomorphic to a decidable relation on the whole of \mathbb{N} because the function enumerating the decidable subset is itself decidable. (This was exercise 50 on p. 82.)

There is a simple cardinality argument to the effect that not every countable ordinal is recursive. Rosser's extended axiom of counting (explain) tells us that the length of the wellordering of all the countable ordinals has uncountable length, so there are uncountably many (in fact \aleph_1) countable ordinals. However the set of recursive ordinals is a surjective image of the set of all machines, and that set is countable. Clearly every recursive ordinal is countable, so there must be countable ordinals that are not recursive.

DEFINITION 20

The sup of the recursive ordinals is the **Church-Kleene** ω_1 , aka ω_1^{CK} .

A standard application of countable choice tells us that every countable set of countable ordinals is bounded below ω_1 , so we know that ω_1^{CK} is actually a countable ordinal. But we can do much better than that, and without using the axiom of choice.

REMARK 11 *The family of recursive ordinals is a proper initial segment of the second number class.*

Proof:

Suppose $<_R$ is a wellordering of \mathbb{N} whose graph is a decidable subset of $\mathbb{N} \times \mathbb{N}$. That is to say that the length of $<_R$ is a recursive ordinal. Now

consider any ordinal α less than the length of R . This is the length of a proper initial segment of $<_R$ —the length of $<_R \upharpoonright \{m \in \mathbb{N} : m <_R n\}$ for some n , say—and this initial segment of $<_R$ is a decidable subset of $\mathbb{N} \times \mathbb{N}$ (it has the number n as a parameter) and its length is therefore a recursive ordinal. ■

This means that ω_1^{CK} is not merely the sup of the recursive ordinals but the least nonrecursive ordinal—and this is indeed how it is usually defined.

REMARK 12 *Every recursive limit ordinal has cofinality ω —recursively. That is to say: whenever R is a decidable binary relation on \mathbb{N} that wellorders \mathbb{N} to a length that is a limit ordinal there is a decidable $X \subseteq \mathbb{N}$ s.t. $\text{otp}(R \upharpoonright X) = \omega$.*

Proof: Recycle the usual proof that countable limit ordinals have cofinality ω . It works in this context.³ We enumerate the members of X in increasing order x_0, x_1, \dots . We set $x_0 := 0$. Thereafter x_{n+1} is the least natural number x such that $\langle x_n, x \rangle \in R$. There is always such an x and it is always decidable for any candidate whether or not the candidate passes. This ensures that X is a semidecidable set which can be enumerated in increasing order, and this makes it decidable (by exercise 78). ■

Observe that this proof is effective: there is a computable function which, on being given the number of a characteristic function of a wellordering of \mathbb{N} , returns the number of the characteristic function of an unbounded subset of length ω .

EXERCISE 91

The class of recursive ordinals is closed under the Doner-Tarski function f_α (see definition 8 p. 44) for every recursive ordinal α .⁴

Thus ω_1^{CK} is huge. This is contrast to the corresponding ordinal for automatic structures: the least ordinal not the ordertype of an automatic wellordering is ω^ω , see [17]

Something to be alert to. Do not confuse the concept of a recursive ordinal with the concept of a *recursive pseudowellordering* of \mathbb{N} . This would be a decidable binary relation R on \mathbb{N} which is a total order with the property that every decidable subset of \mathbb{N} has an R -least member.

When reasoning inside a formal system of arithmetic care is needed in approaching the concept of recursive ordinal. It's one thing to have a definable binary relation on \mathbb{N} , it is quite another to have a proof that this definable binary relation is a wellorder. Come to think of it, how on earth can a system of first-order arithmetic (such as Peano Arithmetic) ever prove that a binary relation is wellfounded? After all, to show that a relation is wellfounded one

³This was question 12 on Prof Johnstone's fourth sheet for Logic and Set Theory in Part II in 2011/12 so you might remember it.

⁴Come to think of it i'm not really entirely happy about this ... but Stan says it's obvious so it must be OK

has to be able to reason about all the subsets of its domain, and a first-order theory cannot reason about arbitrary subsets. The answer is that whenever T (being a first order theory of arithmetic) proves that a relation R on \mathbb{N} is a wellorder what is going on is that T proves all instances of R -recursion that can be expressed in the language of T .

EXERCISE 92 (*Jockusch*)(*)

Show that there is a decidable two-colouring of $[\mathbb{N}]^3$ such that any infinite set monochromatic for it can be used to solve the halting problem.

[Hint: consider the 3-place relation $\{p\}_z(d)\downarrow$.]

EXERCISE 93 (*)

Construct a recursive counterexample to König's Infinity lemma: a recursive finitely-branching tree with no infinite recursive path.

7.3 Tennenbaum's Theorem

O Tennenbaum, O Tennenbaum!

Wie treu sin' deine Blätter!!

See [7] (but not all editions), [62] and [35].

We know from compactness arguments that there must be nonstandard models of PA—models containing nonstandard naturals. In this section we explore what we can do with them.

You may recall from Question 11 on Example Sheet 3 of Professor Johnstone's Set Theory and Logic course in Part II in 2012/3 the relation $n E m$ on natural numbers, which holds when the n th bit of m (m considered as a bit-string) is 1. Clearly E is a decidable relation. It is useful to us because it holds out the possibility of using a natural number to code a set of natural numbers: for any n , $\{m : m E n\}$ is an actual set (a subset of the model) and it is coded by the element n of the model. There are other encoding schemes that pop up in the literature, and they are there because they are syntactically simpler. For example we can think of a number n as encoding the set $\{m : \text{the } m\text{th prime divides } n\}$.

Once we have erected a coding scheme we can use it to think of any nonstandard model \mathfrak{M} of PA as a structure for the language of second-order arithmetic, in the following way. \mathfrak{M} has a standard part, and any nonstandard element of M has the potential to encode sets that are unbounded in the standard part. According to the PTJ-example-sheet scheme no two (nonstandard) naturals can encode the same set, while according to the prime-number scheme distinct (nonstandard) naturals can encode the same set. However we are interested in the *standard part* of any set coded by a (nonstandard) natural, and—on this view—two distinct (nonstandard) naturals can encode the same set of standard naturals even under the PTJ-example-sheet scheme. The structure-for-the-language-of-second-order-arithmetic to which \mathfrak{M} corresponds has as its

carrier set the standard part of \mathfrak{M} . The range of the second order variables is the whole of M . Let us call this structure " \mathfrak{M}^* ".

We start with a simple observation.

REMARK 13 *In \mathfrak{M}^* , every decidable set of standard naturals is encoded by a [nonstandard] natural [of \mathfrak{M}].*

Proof:

Let's write ' $x \oplus y$ ' for the logical **or** of the two naturals x and y thought of as bit-strings. [There is nothing specific to \mathfrak{M}^* here: this is happening in PA]. Let $P()$ be a decidable predicate.

We will need the function f defined as follows.

$$\begin{aligned} f(0) &= 0; \\ f(n+1) &= \text{if } P(n) \text{ then } f(n) \oplus 2^n \text{ else } f(n). \end{aligned}$$

Of course, if we prefer coding with primes then we obtain a different definition.

Now we find, in \mathfrak{M} , that $\{n : n \text{ is standard} \wedge P(n)\}$ is encoded by $f(m)$ whenever m is nonstandard, and $f(m)$ is of course a set of \mathfrak{M}^* . That is to say that, in \mathfrak{M}^* , any decidable set of standard natural number is a set of \mathfrak{M}^* —as desired. ■

Are there any undecidable predicates we can encode? If there is, then we can derive a contradiction from the assumption that the graph of $+$ and \times in the model is decidable. This is

THEOREM 17 (*Tennenbaum*)

PA has no nonstandard model in which the graphs of $+$ and \times are decidable.

Proof:

Suppose some undecidable set A is encoded by a nonstandard natural m . Then—because the membership relation is decidable—it follows that membership in A becomes the relation " nEm " which is of course decidable. Now the obvious undecidable predicate is the halting set. You might think (as did I!) that one can encode it by the following ruse. Define f as follows:

$$\begin{aligned} f(0) &= 0; \\ f(n+1) &= \text{let } n+1 = \langle p, i, t \rangle \text{ in} \\ &\quad \text{if } \{p\}_t(i) \downarrow \text{ then } f(n) \oplus 2^{\langle p, i \rangle} \text{ else } f(n). \end{aligned}$$

The key to understanding what f does is to fix some pair $\langle p, i \rangle$ in one's mind and think about what happens to the $\langle p, i \rangle$ th bit of $f(n)$ as n gets bigger. It starts off clear, but gets set if $\{p\}(i)$ ever halts; and—once set—it remains set. The values of f are an ever-improving sequence of approximations to a numerical code for the halting set, $\{\langle p, i \rangle : \{p\}(i) \downarrow\}$.

So why doesn't $f(n)$ (for some—indeed any—nonstandard n) encode the halting set? The answer is that \mathfrak{M} might lie about whether or not $\{p\}(i)$ halts,

by saying that it halts when in fact it doesn't. If p is looking for an inconsistency proof for PA (so that $p(i) \downarrow = 1$ iff $p(i)$ is a proof of $\neg \text{Con}(\text{PA})$) then it will never find one in standard time. But it might find one in nonstandard time. *It simply isn't true that a standard p , applied to a standard i , halts in standard time if it halts at all.*

In fact we can indeed encode some undecidable sets—and we will need this fact—but one has to do rather more work.

Meanwhile, to be going on with, we have the following observation of Dana Scott's, which I think I will leave as an exercise.

WKL (“Weak König's lemma”) is the assertion that every rooted binary tree with infinitely many nodes has an infinite branch.

EXERCISE 94 (*Scott*)(*)

If \mathfrak{M} is a nonstandard model of PA and \mathfrak{M}^* the corresponding structure for second-order arithmetic, then $\mathfrak{M}^* \models \text{WKL}$.

We proceed to Tennenbaum's theorem.

We recall from remark 10 that there is a pair of semidecidable recursively inseparable sets. It doesn't matter what they are—any pair will do. Let them be $\mathfrak{A} = \{n : (\exists y)A(n, y)\}$ and $\mathfrak{B} = \{n : (\exists x)B(n, x)\}$. Because \mathfrak{A} and \mathfrak{B} are recursively inseparable we must have $(\forall n)(\forall y)(\forall x)(\neg A(n, y) \vee \neg B(n, x))$.

So the standard model believes

$$(\forall n < \underline{m})(\forall y < \underline{m})(\forall x < \underline{m})(\neg A(n, y) \vee \neg B(n, x)) \quad (1)$$

for any numeral \underline{m} . (Recall from p. 42 that \underline{m} is the string $\overbrace{S(S(S \dots 0 \dots))}^{m \text{ times}}$.)

Now observe that expression (1) is absolute and so must be true in any model of PA, and so—in particular—in our \mathfrak{M} . So \mathfrak{M} believes there is an [standard!] m such that

$$(\forall n < m)(\forall y < m)(\forall x < m)(\neg A(n, y) \vee \neg B(n, x)). \quad (C)$$

Indeed it believes this for *all* standard m . Since \mathfrak{M} does not know how to define standard-ness for its members, it must believe that there are nonstandard elements bearing the property C . This trick is known in the trade as an **overspill** argument. Let e be one such. Then we have:

$$\mathfrak{M} \models (\forall n < e)(\forall y < e)(\forall x < e)(\neg A(n, y) \vee \neg B(n, x)). \quad (D)$$

Now let X be the set of those naturals n [in the real world] satisfying $\mathfrak{M} \models (\exists y < e)A(y, \underline{n})$. We will show that X separates \mathfrak{A} and \mathfrak{B} .

$\mathfrak{A} \subseteq X$ holds because any member of \mathfrak{A} bears $A(,)$ to some genuine natural, and any such is less than e .

$\mathfrak{B} \cap X = \emptyset$ holds for similar reasons. Suppose $n \in \mathfrak{B}$. Then there is some m such that $B(n, m)$, whence $\mathfrak{M} \models B(\underline{n}, \underline{m})$, and this \underline{m} is certainly less than e . So $\mathfrak{M} \models (\exists m < e)B(\underline{n}, m)$. But then, by (D) above, $n \notin X$. ■

The final piece of the jigsaw is that if \mathfrak{M} is a model of PA in which even one undecidable set is encoded by a (nonstandard) element then it cannot be recursive. Since—as we have just shown—every nonstandard \mathfrak{M} has a model that encodes a set \mathfrak{A}' separating \mathfrak{A} and \mathfrak{B} , it will follow that every nonstandard model fails to be recursive.

The idea is that

if

- (i) there is an element \mathfrak{x} of the model that encodes an undecidable set \mathfrak{A}' , and
- (ii) the relations of the model are recursive,

then

the element \mathfrak{x} can be used as an oracle to answer questions about membership in \mathfrak{A}' —thereby rendering \mathfrak{A}' decidable.

Better supply the details

A funny Logic

Consider now the Logic \mathcal{L} of those sentences true in all *recursive* (decidable) structures. Let us write ' $T \models_R \phi$ ' to mean that every recursive structure that models T also models ϕ . Since the only recursive model of PA is the standard model, it follows that—according to this logic \mathcal{L} —PA is complete. We must have $\text{PA} \models_R \text{Con}(\text{PA})$ or $\text{PA} \models_R \neg\text{Con}(\text{PA})$.

The second is false, because PA has a model, so we infer $\text{PA} \models_R \text{Con}(\text{PA})$.

It's a pretty safe bet that \mathcal{L} is not axiomatisable!

7.4 Recursive Saturation

This section is a **stub**

The models of Pressburger that can be expanded to models of PA are precisely the recursively saturated ones.

Chapter 8

Incompleteness

8.1 Proofs of Totality

I emphasised on p 61 that concentrating on partial functions was the conceptual breakthrough: it was that that enabled us to prove the completeness theorem for computable partial functions. Quite how big a mess we would have got into if we had stuck with total functions is shown by the diagonal argument:

THEOREM 18 *The set of gnumbers of machines that compute total computable functions is not semidecidable.*

Proof: Suppose the set of gnumbers of machines that compute total functions were semidecidable. Then there would be a total computable function f whose values are precisely the gnumbers of machines that compute total functions. Now consider the function $\lambda n.\{f(n)\}(n) + 1$. This function is total computable and should therefore be $\{f(m)\}$ for some m . But it cannot be $\{f(m)\}$, because its value for argument m is $\{f(m)\}(m) + 1$ and not $\{f(m)\}(m)$. ■

We knew from Rice's theorem (theorem 14 p 92) that this set could not be decidable, but this claim is stronger. However, it should not come as a surprise. Ask yourself: if I am given the gnumber of a machine, can I confirm in finite time that the function computed by that machine is total? At the very least, it is obvious that there is no *straightforward* way of confirming it in finite time. So one should not be surprised that there is in fact no way at all of doing it—in finite time.

It can be hard to see whether or not a function specification specifies a function with the same graph as a recursive specification (Waring numbers). However it is mechanical to check whether or not a piece of syntax is literally a definition of a computable function.

The interesting hard part generally is establishing whether or not a palpably μ -recursive definition defines a *total* function.

It turns out that stronger theories of arithmetic can prove totality of more function declarations than weak theories can. This will lead us to a famous theorem of Gödel's.

8.2 A Theorem of Gödel's

A sound theory of arithmetic is one all of whose axioms are true. (Don't panic! 'unsound' does not imply 'inconsistent'. There are plenty of unsound consistent theories of arithmetic.)

Fix a theory T of arithmetic, with a semidecidable set of axioms.

We proved in theorem 18 p. 119 that the set of gnumbers of programs that compute total functions is not semidecidable. Observe however that, in contrast, the set of gnumbers of programs that compute functions *that T can prove to be total* definitely is semidecidable. It is obviously decidable whether or not a given proof is a proof that a given function is total. So, given a program, we can systematically examine all T -proofs to see whether or not any one of them is a proof that the program computes a total function.

Observe that this brings us some unwelcome news. If T is a recursively axiomatised system of arithmetic then the set of gnumbers of machines for which T can demonstrate that they compute total functions is a semidecidable set—unlike the set of gnumbers of machine that compute total functions. So these sets cannot be the same. So either T proves some function to be total when it isn't, or fails to prove total some function that happens to be total. This is bad enough, but a refined analysis will tell us more, and will explicitly provide a total function whose totality T cannot prove—if it is sound.

8.2.1 The T -bad function

Consider the machine \mathfrak{M} that tests, for each pair $\langle p, n \rangle$ of a T -proof p and a machine n , whether or not p is a T -proof that the function computed by n is total. Naturally we use a volcano for \mathfrak{M} . We modify the volcano to obtain a total function V which emits all pairs $\langle p, n \rangle$ where p is a T -proof that the function $\{n\}$ is total. For each $k \in \mathbb{N}$, we take $V(k)$, which will be a pair $\langle p, n \rangle$. We then compute $\{n\}(k) + 1$ and emit this as our output for input k .

Let us call this the **T -bad-function**. That is to say, the T -bad function is

$$\lambda k \in \mathbb{N}.(\text{let } \langle p, n \rangle = V(k) \text{ in } \{n\}(k) + 1)$$

Consider now the project of proving that the **T -bad-function** is total. Obviously we are not going to be able to do this in T ! So our refined analysis has already given us another nugget: a computable total function whose totality T cannot prove if it is sound.

A fruitful question to ask is: how can T fail to prove that the **T -bad-function** is total?

We have to prove the following:

For every n , if we take the n th program that T proves to be total, evaluate it at n and add 1 to the result, we get an answer, and this defines a total function (A)

Notice the difference between (A) and

For every n , if we take the n th total program, evaluate it at n and add 1 to the result, we get an answer, and this defines a total function. (B)

(B) is obviously going to be provable (in T or any halfway-sensible system we choose), but sadly it is not (B) we are attempting to prove in T but (A). That is to say: in T when we pick up an arbitrary n we are asking not *whether or not the n th program is total*, but *whether or not T proves that it is total*.

Observe that unless T is sound, (A) will not be true, let alone provable. (Think about what happens if we take the n th program that T proves to be total, but T is mistaken...?) Observe also that if T is merely *consistent* (never mind *sound*) it cannot prove (A), for were it to prove (A) it would both prove and not prove that the *T-bad-function* is total.

Thus we have established

THEOREM 19 *If T is a sound theory of arithmetic with a semidecidable set of axioms then T is incomplete.*

Indeed the proof explicitly exhibits an assertion that T cannot prove—namely the assertion that the *T-bad-function* is total. (This assertion is Π_2 , which is not best possible). Later, in chapter 11, we will see an example of a specific theory T and a specific function whose totality cannot be proved in T because it would imply the consistency of T .

Now suppose we add to T a rule of inference (the “*T-soundness*” rule) allowing us to infer ϕ from the fact that $T \vdash \phi$. Observe that in this system we can actually prove (A)—as follows...

The set $\{n \in \mathbb{N} : T \vdash \text{“}\{n\} \text{ is total”}\}$ is a semidecidable set, and is therefore f “ \mathbb{N} for some computable total function f ”.

Now let n be an arbitrary natural number, and consider the function $\{f(n)\}$. By the new rule of inference we infer that $\{f(n)\}$ is in fact genuinely a total function, so $\{f(n)\}(f(n)) + 1$ is defined; n was arbitrary, so the diagonal function $\lambda n.(\{f(n)\}(f(n)) + 1)$ is total. So we have proved (A).

COROLLARY 3

The T -soundness rule of inference cannot be a derived rule of inference for T .

■ expand this

Observe that corollary 3 doesn't say that the *T-soundness* rule of inference is *invalid* or *unsound*, merely that it is not a derived rule of inference for T .

Specifically there is nothing to prevent us adding it as a rule of inference should we feel like it.

The theory \mathcal{A} of truths of arithmetic is obviously complete and sound. From the foregoing it now follows that it cannot be recursively axiomatised. But the construction actually shows a bit more than that. If T is any sound recursively axiomatised theory of arithmetic the above construction shows how we can “diagonalise out of” T while remaining entirely within the set of arithmetic truths. What we have in our hands is an algorithm which, on being presented with a recursively axiomatisable $T \subseteq \mathcal{A}$ (such a T is a finite object and is a possible input to an algorithm), returns something in $\mathcal{A} \setminus T$. This property of \mathcal{A} is important enough to deserve a name...

DEFINITION 21 *Suppose $X \subseteq \mathbb{N}$ is not only not semidecidable but also comes equipped with a computable function f which “diagonalises out of” any semidecidable $\{n\}$ “ $\mathbb{N} \subseteq X$ in the sense that $f(n)$ is a member of $X \setminus \{n\}$ ” \mathbb{N} . Then we say X is **productive**.*

Clearer if we appeal to Gödel’s argument

So what we proved above can be stated as:

REMARK 14 *The set of (gnumbers of) arithmetic truths is productive.*

■

If I ever get round to it

We will see later that A is productive iff A is in some sense “at least as undecidable” as the halting set, the set $\{\langle p, i \rangle : \{p\} \downarrow (i)\}$.

EXERCISE 95 (*) *Jason Long said to me the other day that the complement of the Halting set is productive. Which version of the Halting set did he mean? And what did he mean by ‘complement’?*

EXERCISE 96 (*)

Stephan [64] says that both $\{e : |W_e| < \aleph_0\}$ and $\mathbb{N} \setminus \{e : |W_e| = \aleph_0\}$ are productive.

Although the set of arithmetic truths really is productive, and there really is an algorithm that will accept a decidable axiomatisation of a fragment of arithmetic and emit something unprovable in that fragment, the fact remains that the algorithm is a bit unwieldy. Producing actual arithmetic truths that are demonstrably unprovable in specific recursively axiomatisable fragments of arithmetic requires ingenuity. We will see an example in chapter 11.

8.3 Undecidability of Predicate Calculus

THEOREM 20 *The set of (gnumbers of) valid expressions of First-Order Logic is not decidable.*

Proof:

We have to be careful to state this properly. Monadic first-order logic without equality is decidable, as one can easily see once one notices that the language with n monadic predicate letters can distinguish only 2^n things and so any falsifiable formula has a finite countermodel which can be found by systematic exhaustive search. [In fact, what this shows is that every structure for monadic first-order-logic-without-equality has a finite elementary substructure].

We mean *sufficiently rich* [valid expressions of ...]. How rich is sufficiently rich? Rich enough to describe the working of a Turing machine. We know that truth-in-all-models is finitely detectable, we want to use Turing machines to show that falsifiability is not finitely detectable. So we have to show that if falsifiability is finitely detectable, then we can detect computations that will not HALT. So, given a computation of p with input i , we have to find a sentence in the appropriate language with a counter model. The sentence will be the one that says that p applied to i does not HALT.

We want to use the unsolvability of the Halting problem to prove the undecidability of First-Order-Logic. On being given a Turing machine \mathfrak{M} and an input i to \mathfrak{M} , we can compute a formula ϕ which has the property that every model of ϕ is a complete course of computation of \mathfrak{M} on input i and has a last stage at which \mathfrak{M} has HALTed. If the set of valid expressions of First-Order-Logic is decidable then we can determine whether or not this ϕ has a model. But ϕ has a model iff \mathfrak{M} halts on i , and *that*, we know, is not decidable. So the set of valid expressions of First-Order-Logic is not decidable either.

8.4 Trakhtenbrot's theorem

THEOREM 21 (*Trakhtenbrot*)

The set of sentences true in all finite structures is not semidecidable.

Proof: We show that if it were semidecidable we would be able to solve the halting problem. To this end what we want is a standard uniform method which, on being presented with an instance $\langle \mathfrak{M}, n \rangle$ of the halting problem, emits a formula ϕ of first-order logic with the property that:

$$\phi \text{ is true in all finite structures} \quad \text{iff} \quad \mathfrak{M}(n) \uparrow. \quad (\text{A})$$

What would ϕ be? The idea is that any finite model of ϕ will be a course-of-computation-of- \mathfrak{M} -applied-to- n (as in Kleene's T function) with the property that the state in the last snapshot in the list is not HALT. Loosely, a finite model of ϕ will be a computation-of- \mathfrak{M} -applied-to- n that is still running, so that if ϕ has arbitrarily large finite models then \mathfrak{M} applied to n never HALTs. So ϕ must say the following.

“There are these things called stages, and there is an order of succession on them. There is a first stage and every stage except the first has a unique predecessor. Every stage has at most one successor. Each stage s is a configuration of \mathfrak{M} (that is to say: an ordered pair

of a state of \mathfrak{M} and a state of \mathfrak{M} 's tape) and the stage succeeding s must be the configuration that arises from s as a result of the quintuples that represent \mathfrak{M} . Finally

if the first stage is $\langle \mathfrak{M}$ -in-its-start-state, n -on-the-tape \rangle **then** no stage has \mathfrak{M} -in-the-HALT-state as its first component."

This second clause is a conditional not a conjunction because the requirement on ϕ is that it should be true-in-all-finite-structures-(of the appropriate signature) iff $\{m\}(n)\uparrow$. So it must be true in any finite structure (of the appropriate signature) and typically such a finite structure is a description of a course of computation for a different machine or of the same machine on a different input.

We now have to check that ϕ obeys (A), that is: $\mathfrak{M}(n)\uparrow$ iff ϕ is true in every finite structure of the appropriate signature. What is the appropriate signature? It has apparatus for describing stages, and states of a machine and configuration of a tape. A structure \mathfrak{X} for $\mathcal{L}(\phi)$ has stages ordered like an initial segment of a model of arithmetic (so it is finite or is an ω -sequence followed by some number of $\omega^* + \omega$ sequences). It also contains a description of a Turing machine. There is nothing in being-a-structure-for- $\mathcal{L}(\phi)$ to say that the stages of \mathfrak{X} obey the transition rules for the machine, nor that the machine in question is \mathfrak{M} : for that \mathfrak{X} has to be a model of ϕ .

L \rightarrow R

Suppose \mathfrak{X} is a finite structure of the appropriate signature. If it doesn't encode a course of computation of \mathfrak{M} then it trivially satisfies ϕ by falsifying the antecedent. On the other hand, if it satisfies the antecedent then it really is a course of computation for $\mathfrak{M}(n)\uparrow$ and therefore—since $\mathfrak{M}(n)\uparrow$ —it will satisfy the conclusion.

R \rightarrow L

Suppose every finite \mathfrak{X} of the appropriate signature satisfies ϕ . Then $\mathfrak{M}(n)$ cannot halt. For if it did there would be a course of computation of \mathfrak{M} applied to n whose final snapshot showed \mathfrak{M} in the HALT state, and such a course of computation would be a finite structure $\mathfrak{X} \models \neg\phi$, contradicting assumption.

■

Note: I have been a *little* bit hand-wavy in the specification of ϕ . In order for an expression like ϕ to be finitary first-order we probably have to specify an upper bound for the number of states of \mathfrak{M} . However this does not cause us any difficulties: after all, we are cooking up ϕ on being given \mathfrak{M} , so we just take the bound to be $|\mathfrak{M}|$.

Something to think about:

EXERCISE 97 (*)

Will the same argument show that the set A of formulae with arbitrarily large finite models is not semidecidable? Or the set B of sentences true in all sufficiently large finite models?

What about the set of sentences true in all infinite models?

What about the set of sentences true both in all finite structures that have even cardinality and in all infinite structures?

Observe that, despite exercise 97, the set of sentences true in all infinite structures is axiomatisable, for it is the set of deductive consequences of the set “there are at least n things” for all n .

8.5 Refinements of theorem 19

Try doing all the constructions of this section using instead of “ $T \vdash \{n\}$ is total” something along the lines of “ $T \vdash \{n\}$ converges rapidly”.

Chapter 9

WQO theory, Kruskal's theorem and Friedman's Finite Form

Recall *stretching* from exercise 6.

The stretching relation on Q -lists is inductively defined as the \subseteq -smallest set of ordered pairs of Q -lists containing $\langle \text{nil}, \text{nil} \rangle$ and containing $\langle l_1, l_2 \rangle$ if it contains $\langle l_1, \text{tl}(l_2) \rangle$, or if $\text{hd}(l_1) \leq \text{hd}(l_2)$ and it contains $\langle \text{tl}(l_1), \text{tl}(l_2) \rangle$.

No list stretches into its tail: for all lists l over a quasiorder, $l >_l \text{tl}(l)$. $l \geq_l \text{tl}(l)$ by the second clause in the recursive definition of \leq_l , and we prove by induction that no list can \leq_l something strictly shorter than itself. In contrast to lists, a stream might stretch into its tail.

There are also finite trees over X , and we need to think how to lift quasiorders of X to trees over X .

There is of course an inductive definition for tree-stretching for finite trees.

DEFINITION 22 $T_a \leq_t T_b$ if

- Both are singleton trees $\{a\}$ and $\{b\}$ with $a \leq b$; or
- $T_a \leq_t$ some child of T_b ; or
- The root of $T_a \leq$ root of T_b and the list of children of $T_a \leq_l$ list of children of T_b .

PROPOSITION 1 If $\langle Q, \leq \rangle$ is a wellfounded quasiorder, then Q -lists are wellfounded under stretching.

Proof: Suppose not, and we had an infinite descending sequence of Q -lists under stretching. They can get shorter only finitely often, so without loss of generality we may assume that they are all the same length. But the entries at

each coefficient can get smaller only finitely often, so they must eventually be constant. ■

PROPOSITION 2 *If $\langle Q, \leq \rangle$ is a wellfounded quasiorder, then finite Q -trees are wellfounded under tree-embedding.*

Proof: Suppose $\langle Q, \leq \rangle$ is a wellfounded quasiorder and let $\langle t_i : i < \omega \rangle$ be a descending $>_t$ -sequence of Q -trees. We will derive a contradiction. The number of children of t_i is a nonincreasing function of i and must be eventually constant: indeed the trees will be of eventually constant shape, and we can delete the initial segment of the sequence where they are settling down. Because the shape is eventually constant there are unique maps at each stage, so for any one address the sequence of elements appearing at that address gets smaller as i gets bigger. ■

9.1 WQOs

DEFINITION 23 *$\langle Q, \leq \rangle$ is a **wellquasiorder** (hereafter **WQO**) iff whenever $f : \mathbb{N} \rightarrow Q$ is an infinite sequence of elements from Q then there are $i < j \in \mathbb{N}$ s.t. $f(i) \leq f(j)$.*

A natural example of a WQO is the set of (unordered) pairs of natural numbers with $\{x, y\}$ related to $\{n, m\}$ if $\max(x, y) \leq \min(n, m)$. It is the wellfoundedness of this quasiorder that ensures termination of Euclid's algorithm.

DEFINITION 24 *A **bad sequence** (over $\langle Q, \leq \rangle$) is a sequence $\langle x_i : i \in \mathbb{N} \rangle$ such that for no $i < j$ is it the case that $x_i \leq x_j$. A sequence that is not bad is **good**. A sequence $\langle x_n : n \in \mathbb{N} \rangle$ is **perfect** if $i \leq j \rightarrow x_i \leq x_j$.*

*Finite sequences $\langle x_i : i < k \in \mathbb{N} \rangle$ too will sometimes be said to be **bad** as long as they satisfy the remaining condition: $i < j < k \rightarrow x_i \not\leq x_j$.*

Thus a wellquasiorder is a quasiorder with no bad sequences. With the help of Ramsey's theorem we can prove that in a WQO not only is every sequence good but that it must have a perfect subsequence. (Notice that this is not the same as saying that in any quasiorder every good sequence has a perfect subsequence!)

LEMMA 1 *In a WQO every sequence has a perfect subsequence.*

Proof:

Although this theorem is very easy to prove, the usual (indeed only) proof using Ramsey's theorem is so natural and idiomatic, and so important *qua* prototype for so many other applications of Ramsey's theorem, that it is worth doing in full.

Let $\langle Q, \leq_Q \rangle$ be a WQO, and $f : \mathbb{N} \rightarrow Q$ a sequence. Partition $[\mathbb{N}]^2$ into the two¹ pieces $\{\{i < j\} : f(i) \leq_Q f(j)\}$ and $\{\{i < j\} : f(i) \not\leq_Q f(j)\}$. An infinite

¹We write ' $\{i < j\}$ ' for the unordered pair of two naturals i and j instead of ' $\{i, j\}$ ' when we wish to supply the information that $i < j$.

subset monochromatic for the first piece would give us a bad sequence, contradicting the assumption that $\langle Q, \leq_Q \rangle$ was a WQO, and the set monochromatic for the second piece is a perfect subsequence. ■

A quasiorder is a WQO iff the strict version of the corresponding partial order is wellfounded and has no infinite antichains. (miniexercise) Notice this does *not* mean that for each x in a WQO there are only finitely many things incomparable with x , nor even that there are only finitely many equivalence classes of things incomparable with x . What it does say is that if there are infinitely many things incomparable with x , some of them will be comparable with some others.

Now some basic facts about WQO's, some with an algebraic flavour.

PROPOSITION 3

- (i) *Substructures of WQOs are WQO;*
- (ii) *Homomorphic images of WQOs are WQO;*
- (iii) *The pointwise product of finitely many WQOs is WQO;*
- (iv) *The intersection of finitely many WQOs is WQO;*
- (v) *Disjoint unions of finitely many WQO are WQO;*
- (vi) *If \leq_1 and \leq_2 are both quasiorders of a set Q , and the graph of \leq_1 is a subset of the graph of \leq_2 , and \leq_1 is a WQO, then so is \leq_2 .*

Proof:

(i). Any bad sequence in a substructure is a bad sequence in the whole structure.

(ii). Suppose $f : \langle Q, \leq \rangle \rightarrow \langle X, \leq \rangle$ is a quasiorder homomorphism and $S : \mathbb{N} \rightarrow X$ a bad sequence of members of X . Consider $Q^\dagger = \{q \in Q : (\exists n \in \mathbb{N})(f(q) = S(n))\}$. (We are of course assuming that f is a surjective homomorphism). Let R be the binary relation $R(q, q')$ iff $(\exists n \in \mathbb{N})(S(n) = f(q) \wedge S(n+1) = f(q'))$. $R \subseteq (Q^\dagger \times Q^\dagger)$ satisfies the conditions for the application of *DC* (in that $(\forall q \in Q^\dagger)(\exists q' \in Q^\dagger)(R(q, q'))$), and the output sequence will be a bad sequence of members of Q .

For (iii) (iv) and (v) it is clearly sufficient to deal with the case of two WQOs. The proofs of all three use Ramsey's theorem with exponent 2, or the perfect subsequence lemma (lemma 1). For (iii) consider the product of two WQOs $\langle Q, \leq_Q \rangle$ and $\langle X, \leq_X \rangle$, and suppose we have a bad sequence $\langle \langle x_i, q_i \rangle : i \in \mathbb{N} \rangle$. By the perfect subsequence lemma there must be an infinite $I \subseteq \mathbb{N}$ such that for $i < j$ both in I we have $x_i \leq_X x_j$. Now consider the sequence of q_i for $i \in I$. This must be a good sequence, since $\langle Q, \leq_Q \rangle$ is WQO, so there are $i < j$ both in I with $q_i \leq_Q q_j$. So $\langle \langle x_i, q_i \rangle : i \in \mathbb{N} \rangle$ was not bad.

The proofs of (iv) and (v) are almost exactly the same. Notice that (iv) tells us in particular the if \leq_1 and \leq_2 are two WQOs of one carrier set, then the intersection of their graphs is the graph of a WQO of the same set.

Finally (vi) is obvious, but—since it will be generalised later—a bit of detail may be helpful. A quasiorder is a WQO if the complement of its graph does

not contain a copy of $\langle \mathbb{N}, <_{\mathbb{N}} \rangle$. This property of (the graph of) a relation is clearly preserved under superset. ■

9.1.1 The Minimal Bad Sequence construction

There is a well-defined notion of the *wellfounded part* of a quasiorder $\langle X, \leq_X \rangle$ that we saw in exercise 18. Sadly there is no good notion of the *WQO part* of a relation; however there is an ingenious construction which will do some of the work to which we would have put such a notion had there been one. Any quasiorder that is wellfounded but is not WQO has bad sequences, and—as we shall see—has some that are in some sense minimal. This “minimal bad sequence” is a key idea, and its significance for us here is that the set of things below such a minimal bad sequence in a wellfounded quasiorder $\langle X, \leq \rangle$ behaves in some ways as if it were the WQO part of $\langle X, \leq \rangle$. A precise definition will be given later²: for the moment our approach is a two pronged one: (i) How do we make one? (ii) What can it do for us once we have got it?

Let $\langle X, \leq_X \rangle$ be a wellfounded quasi order that is not WQO.

Let A be the set of proper initial segments of bad X -sequences, and let $R(s, t)$ hold if t is an end-extension of s by one element x , with x minimal so that s with x on the end is in A . Notice that $(\forall s \in A)(\exists t \in A)(R(s, t))$ so the conditions of DC apply.

Let x_0 be a minimal member of $\{x : \text{there is a bad sequence whose first member is } x\}$. Let x_{n+1} thereafter be a minimal member of $\{x : \text{there is a bad sequence whose first } n \text{ members are } \langle x_0 \dots x_{n-1} \rangle \text{ and whose } n+1 \text{th member is } x\}$. Let us say that a sequence constructed by this algorithm is an **MBS**.

The following remark is not needed just yet but crops up naturally here. If we topologise Q^ω in the usual way by giving Q the discrete topology and Q^ω the product topology, we find that if Q is a quasiorder that is not WQO then the set of bad sequences is a closed subset of Q^ω in the product topology. That is why the MBS algorithm—which is a greedy algorithm—works. The set of its outputs is a closed set.

REMARK 15 *If Q is a QO that is wellfounded but not WQO then the set of MBSs is a closed subset of Q^ω .*

Fortunately we do not need every subsequence of an MBS to be an MBS. We need every such subsequence to be bad, and we also need every sequence which is in some suitable sense below an MBS to be good.

LEMMA 2 *Let $\langle X, \leq \rangle$ be a wellfounded quasiorder that is not a WQO and $B = \langle b_0, b_1 \dots \rangle$ be an MBS. Let $X' = \{x \in X : (\exists n)(x < b_n)\}$. Then $\langle X', \leq \rangle$ is WQO.*

²By the reader!

Proof: Suppose $S = \langle s_0, s_1 \dots \rangle$ is a bad sequence from X' . We will prove by induction on \mathbb{N} that nothing in S is below b_n . This is clearly true for $n = 0$, as follows. If $s_i < b_0$ then the tail of S starting at s_i is a bad sequence beginning with something less than b_0 contradicting minimality of b_0 .

For the induction suppose that nothing in S is below any of $b_1 \dots b_n$. Suppose now that, *per impossibile*, there were $s_i < b_{n+1}$. Consider the sequence that begins $b_0 \dots b_n$ and continues $s_i, s_{i+1} \dots$. It can't be bad, because b_{n+1} was minimal among the set of elements that are the $n + 1$ th members of bad sequences beginning $b_0 \dots b_n$, and $s_i < b_{n+1}$ rules s_i out as a candidate. So this sequence contains a good pair. Both S and B are bad, so the good pair must be a $b_j \leq s_k$ with $j \leq n$ and $k \geq i$. Now consider s_k . It is in X' so there is a b_m in B with $s_k < b_m$. This m cannot be $\leq n$, by induction hypothesis, so we must have $m > n$. But then we have $j \leq n < m$ with $b_j \leq b_m$ (in fact $b_j < b_m$) contradicting badness of B . ■

9.2 Kruskal's theorem

Next we show that (finite) lists over a WQO are WQO.

LEMMA 3 *If $\langle X, \leq \rangle$ is a WQO, so is $\langle X^{<\omega}, \leq_l \rangle$.*

Proof: We use *reductio ad absurdum*. Suppose that $\langle X, \leq \rangle$ is a WQO but that $\langle X^{<\omega}, \leq_l \rangle$ is not. We know by now from proposition 1 that $\langle X^{<\omega}, \leq_l \rangle$ is wellfounded, so let us construct a minimal bad sequence $\langle a_i : i \in \mathbb{N} \rangle$ of lists. Look at the heads of the lists in the minimal bad sequence. These are WQO by hypothesis so (by lemma 1) there must be an infinite subsequence $\langle b_i : i \in \mathbb{N} \rangle$ of $\langle a_i : i \in \mathbb{N} \rangle$ such that for $i < j$, $\text{hd}(b_i) \leq \text{hd}(b_j)$. Throw away all the other lists in this bad sequence. We now have a bad sequence of lists whose heads, at least, form an increasing sequence. Now consider the tails. We want to show that the tails are WQO as well, for that will complete the proof for us by using the third clause of definition 9. We know from that $\text{tl}(l) <_l l$ always, so these tails belong to a collection of things *below* this minimal bad sequence, $\langle a_i : i \in \mathbb{N} \rangle$, in the sense of lemma 2. Therefore the sequence of tails of elements of $\langle b_i : i \in \mathbb{N} \rangle$ is not a bad sequence. So there are $i < j$ such that $\text{tl}(b_i) \leq_l \text{tl}(b_j)$. Therefore (by the third clause in the inductive definition of \leq_l) $b_i \leq b_j$, so $\langle b_i : i \in \mathbb{N} \rangle$ is not a bad sequence, and $\langle a_i : i \in \mathbb{N} \rangle$ is not bad either. ■

Now we can prove

THEOREM 22 (*Kruskal*) *Finite trees over a WQO are WQO.*

Proof: By wellfoundedness of $<_t$, if there is a bad sequence there is a minimal bad sequence, and let $\langle a_i : i \in \mathbb{N} \rangle$ be one. Look at the roots of the trees in this sequence. Since the roots are from a WQO there must be an increasing ω -subsequence $\langle b_i : i \in \mathbb{N} \rangle$ from $\langle a_i : i \in \mathbb{N} \rangle$ such that if $i < j$ then $(\text{root of } b_i) \leq (\text{root of } b_j)$ (this was lemma 1). Let l_i be the list of children of a_i .

We know that the roots of the a_i form a strictly increasing sequence. What we now have to look at is an ω -sequence of lists of children of the trees we started with. These trees form a collection of trees below (in the sense of lemma 2) the minimal bad sequence we started with. So, by lemma 2 they are WQO, so lists over them are WQO as well. Therefore there are $i < j$ with $l_i \leq_l l_j$, so (by the third clause in the definition of \leq_t) it follows that $a_i \leq_t a_j$. Thus $\langle a_i : i \in \mathbb{N} \rangle$ is not bad. ■

9.3 Some *bonnes bouches*

9.3.1 How to get some large ordinals

REMARK 16 Let $\langle X, \leq_X \rangle$ be a countable QO. Then the following are equivalent:

1. $\langle X, \leq_X \rangle$ is WQO;
2. The set $\{\alpha : \exists \text{ homomorphism } \pi : \langle X, \leq_X \rangle \rightarrow \langle \{\beta : \beta < \alpha\}, < \rangle\}$ is bounded below ω_1 .

Proof:

$\neg(\text{i}) \rightarrow \neg(\text{ii})$ is proved by the observation above about infinite antichains.

$(\text{i}) \rightarrow (\text{ii})$

Consider the (downward-branching) tree $\langle B, \succ \rangle$ of bad sequences from X ordered by reverse end-extension (each bad sequence is above all its bad end-extensions). $\langle X, \leq_X \rangle$ is a WQO (no infinite bad sequences) so $\langle B, \succ \rangle$ is well-founded and therefore has a rank. The carrier set X is countable so the set B is countable, and the rank of $\langle B, \succ \rangle$ is therefore countable too. We will show that this rank is an upper bound for $\{\alpha : \exists \text{ homomorphism } \pi : \langle X, R \rangle \rightarrow \langle \{\beta : \beta < \alpha\}, < \rangle\}$.

Let π be such a homomorphism. We quasi-order X by the relation $x \leq_\pi y$ iff $\pi(x) \leq \pi(y)$. Evidently (the graph of) \leq_π is a superset of (the graph of) \leq_X . Accordingly there are fewer bad sequences in $\langle X, \leq_\pi \rangle$ than there are in $\langle X, \leq_X \rangle$. Consider the tree of descending chains in $\langle X, <_\pi \rangle$ ordered by reverse end-extension in the style of $\langle B, \succ \rangle$.

It is easy to see that the rank—in this tree—of any descending sequence is simply π of its last member, so the tree must have rank α . It is also straightforward that any descending chain in $\langle X, <_\pi \rangle$ is a bad sequence in $\langle X, \leq_X \rangle$, so this tree is a subtree of $\langle B, \succ \rangle$, so $\langle B, \succ \rangle$ has rank at least α too. ■

The surjections here are, in contrast to the rank function from page 32, the precise opposite of parsimonious: the rank function tries to use as *few* ordinals as possible. Here we are trying to use as *many* ordinals as possible—while remaining surjective.

9.3.2 Friedman's Finite Form of Kruskal's Theorem

Consider the one-point WQO, and suppose there is a natural number k such that for all n there is a bad sequence of length n :

$$T_1^n, T_2^n, T_3^n \dots T_n^n$$

where T_i^n is a finite tree (over the one-point WQO) with $k + i$ nodes. Then there will be an infinite triangular matrix of trees (one row for each n):

Figure 9.1: An infinite triangle of trees: I

$$\begin{array}{cccc} T_1^1 & & & \\ T_1^2 & T_2^2 & & \\ T_1^3 & T_2^3 & T_3^3 & \\ T_1^4 & T_2^4 & T_3^4 & T_4^4 \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

Consider the first column of figure 9.1, the sequence $\langle T_1^n : n \in \mathbb{N} \rangle$. Each tree in this sequence has only $k + 1$ nodes, so only finitely many of them can be distinct. So some of them must be present with infinite multiplicity, and let us call ' T_1 ' the one that appears first. Now discard all the rows that do *not* begin with T_1 . Now consider the second column and obtain T_2 analogously. Iterate with all subsequent columns. Eventually we will have constructed an infinite bad sequence of trees. But this would contradict theorem 22. Therefore the initial assumption was wrong, so there is no such k , and we have proved

THEOREM 23 $\forall k \exists n$ if $T_1 \dots T_n$ is a list of trees where T_i has $k + i$ nodes, then there are $j < l \leq n$ s.t. $T_j \leq T_l$.

■

Chapter 10

Elementary Degree Theory

DEFINITION 25 *B is many-one reducible to A (written $B \leq_m A$) if there is a total computable f s.t. $(\forall n)(n \in B \iff f(n) \in A)$.*

The point being that if I can correctly answer “is it in A ?” then I can correctly answer “is it in B ?”. Observe that there is no requirement that f be surjective: we might not need the *whole* of A . Evidently \leq_m is a quasiorder of \mathbb{N} .

\mathbb{K} is the Halting set, $\{\langle n, x \rangle : \{n\}(x) \downarrow\}$

Some writers (e.g. [64]) define \mathbb{K} to be the diagonal halting set: $\{n : \{n\}(n) \downarrow\}$. The full halting problem is no harder than the diagonal halting problem. To ascertain whether or not $\{n\}(x) \downarrow$ using only the diagonal halting oracle it suffices to build the machine that computes the function with constant value $\{n\}(x)$. (That is: it suffices to be able to compute—from n and x —the e such that $\{e\}(k) = \{n\}(x)$ for all k .) ...and then apply it to its own index: $\{e\}(e) = \{n\}(x)$.

I’m not sure at this stage what is involved in choosing between these two sets, and (for the moment) whenever furnishing a proof of a theorem about the halting set I shall use that form of the definition that was used in the proof that I found.

The following triviality will help to clear the air.

REMARK 17 *$A \leq_m \mathbb{K}$ iff A is semidecidable.*

Proof:

Since \mathbb{K} is semidecidable it is the domain of a partial recursive g . If $A \leq_m \mathbb{K}$ in virtue of f then A is the domain of $g \circ f$, which makes A semidecidable.

For the other direction, suppose A is semidecidable. Define a binary partial (computable) function f by $f(e, x) =:$ if $e \in A$ then 1 else \uparrow . By theorem 13 (the S-m-n theorem, p. 91) there is now a computable function g such that, for all x and e , $\{g(e)\}(x) = f(e, x)$. From this we have $(\forall e)(\{g(e)\}(g(e)) \downarrow \iff$

$e \in A$). Thus $(\forall e)(e \in A \longleftrightarrow g(e) \in \mathbb{K})$. (Here we take \mathbb{K} to be the *diagonal* halting set). But now $A \leq_m \mathbb{K}$ in virtue of g . ■

This observation is very much in the spirit of *complete problem* from complexity theory. Indeed there is a connection between complete problems and many-one reducibility: NP-complete problems (to pick a salient example) are maximal among NP problems wrt many-one reducibility. The result here, remark 17, is telling us that the halting problem is complete for the class of semidecidable problems.

Despite the neatness of remark 17, the correct notion of reducibility (at least when we are considering maximal unboundedly-finite notions of computations as we are in this course) is *Turing-reducibility*. This is slightly more subtle, and it requires a bit of motivation.

10.1 Computation relative to an oracle.

We add an extra style of command to the language, as it were:

`consult-oracle \mathcal{O} , branch on the answer.`

That is to say: we spice up our machines so that as well as doing whatever it was they were doing already they can now ask an oracle “Is n in \mathcal{O} ?” [where \mathcal{O} is the set that the oracle knows about and we don’t] and branch on the answer, and they can do this as often as they like. We then say $A \leq_T B$ if the characteristic function χ_A (total version) for A can be computed by a machine of the new style that has access to an oracle for B .

In this setting, where we are considering recursion relative to an oracle, we let $\{e\}$ be the e th member of the set of functions-in-intension-that-call-oracles. Think of $\{e\}$ as code written in a language that allows invocations of oracles. Then $\{e\}^C$ is the function computed by $\{e\}$ when given access to the oracle C . The notation ‘ $\{e\}^C$ ’ doesn’t mean “the e th program that calls the oracle C ” [which is what I used to think].

It looks as if, in the first instance, “recursive-in” is defined between sets and functions. (A function is recursive in a set). However we can define what it is for f to be recursive in g .

Lerman [39] defines \leq_T between *functions* rather than between sets. And he does it by considering the set of functions obtained like the general recursive functions but containing g as an extra founder. Thus, f is recursive in g if f is a member of this set.

We can also do it as follows: the program for f is allowed to ask for $g(n)$, and it will be given either a value or the news that $g(n)\uparrow$.

Then we can say that A is recursive in B if χ_A is recursive in χ_B . Observe that if we do this it won’t matter whether we take the characteristic function for A to be $\lambda n.(\text{if } n \in A \text{ then } 1 \text{ else fail})$ or $\lambda n.(\text{if } n \in A \text{ then } 1 \text{ else } 0)$.

Observe that the quasiorder \leq_T is *prima facie* weaker (contains more ordered pairs) than the quasiorder \leq_m , so there can be $A \leq_T \mathbb{K}$ where A is not semidecidable. In particular $\mathbb{K} \leq_T (\mathbb{N} \setminus \mathbb{K})$ but $\mathbb{K} \not\leq_m (\mathbb{N} \setminus \mathbb{K})$.

Why?

Clearly we have $X \leq_T (\mathbb{N} \setminus X)$. So, in particular, $(\mathbb{N} \setminus \mathbb{K}) \leq_T \mathbb{K}$. If *per impossibile* $A \leq_T \mathbb{K}$ were sufficient for A to be semidecidable we would be able to infer that $\mathbb{N} \setminus \mathbb{K}$ were semidecidable, and thence that \mathbb{K} were decidable. But we know it isn't. This gives us a natural example— $\mathbb{N} \setminus \mathbb{K}$ —of a set that is $\leq_T \mathbb{K}$ but is not semidecidable.

Observe that this means that you can have two sets of the same degree of unsolvability (namely \mathbb{K} and $\mathbb{N} \setminus \mathbb{K}$) where one is semidecidable and the other isn't. Whether or not you are semidecidable might not be entirely determined by your Turing degree.

So we might be in with a chance of finding A, B satisfying $A \not\leq_T B \not\leq_T A$ because A and B are not semidecidable.

EXERCISE 98 (*)

Surely Turing reducibility and many-one reducibility must be the same? Surely when I have finitely many questions to ask the oracle I can bundle them up into a single question to ask the set-of-finite-subsets-of-the-oracle? So anything Turing-reducible to A is many-one-reducible to $\mathcal{P}_{\aleph_0}(A)$ (the set of finite subsets of A) and that is no more complex than A .

What has gone wrong?

We observed at the outset that \leq_T is a preorder: transitive and reflexive. The intersection of a preorder and its converse is an equivalence relation. In this case the equivalence classes are called (Turing) **degrees** or **degrees of reducibility**, and \leq induces a partial order of the degrees. The symbol ' \leq_T ' is often simplified (in context) by dropping the subscript. We also tend to equivocate between Turing-reducibility-between-subsets-of- \mathbb{N} and Turing-reducibility-between-degrees, and when we are asserting that \leq_T holds between two *degrees* we tend to use the lower case Roman letter '**d**' (written in **boldface**) to range over degrees. If $A \leq_T B$ we often say " A is recursive in B ".

$\mathbf{0}$ is the degree of decidable sets. \mathbf{d}' is the degree of the halting set for functions that call a set of degree \mathbf{d} .

Observe that the proof of the unsolvability of the halting problem relativises: we can think of theorem 12 as saying $\mathbf{0} <_T \mathbf{0}'$. Then the same proof will show $\mathbf{d} <_T \mathbf{d}'$ for any \mathbf{d} .

EXERCISE 99 (*)

An infinite set $X \subseteq \mathbb{N}$ is introreducible if it is recursive in all its infinite subsets. By considering a labelling of the perfect binary tree, or otherwise, show that every Turing degree contains an introreducible set.

There are various nice properties a poset can have. Look again at exercise 52. The technique there will show that the Turing-degrees-with- \leq_T form an upper semilattice (binary lubs exist) but that's the end of the good news. No other nice properties hold.

10.2 Kleene-Post, Friedberg-Muchnik, and Baker-Gill-Solovay

“Is the relation \leq_T a total order?” one might ask. It turns out that the answer is ‘no’. In fact:

THEOREM 24 *There are incomparable degrees even below $\mathbf{0}'$.*

Proof:

We prove this by a fairly straightforward diagonal construction. Let us start by enumerating the set $\{0, 1\}^{<\omega}$ of all finite strings from $\{0, 1\}$ as $\langle \eta_n : n \in \mathbb{N} \rangle$. Each string is thought of as a function from an initial segment of \mathbb{N} to $\{0, 1\}$.

When the superscript is an η_a —which of course is *finite*—it might happen that $\{e\}$ calls for the oracle to rule on an input at which η_a is not defined. In these circumstances $\{e\}^{\eta_a}(x) \uparrow$. [It seems to me that this means that $\{e\}^{\eta_a}(x)$ is a different notation from $\{e\}^B(x)$ because If B is finite then $\{e\}^B(x)$ will diverge—if at all—only because of e ; it always gets answers from $B \dots$ whereas $\{e\}^{\eta_a}(x)$ might diverge because it doesn't get an answer from the oracle.]

Observe that

- if $\{e\}^{\eta_a}(x) \downarrow$ then $\{e\}^C(x) \downarrow$ for every C extending η_a . (We are thinking here of C as an infinite sequence of 0s and 1s—as a characteristic function, in fact.)
- If $\{e\}^B(x) \downarrow$ then there is $a \in \mathbb{N}$ such that $\{e\}^{\eta_a}(x) \downarrow$

We will construct two sets A and B such that $A \not\leq_T B$ and $B \not\leq_T A$. There will be two sequences of binary strings:

$\langle \alpha_n : n \in \mathbb{N} \rangle$ such that α_{i+1} extends α_i and $\bigcup_{i \in \mathbb{N}} \alpha_i = \chi_A$; and $\langle \beta_n : n \in \mathbb{N} \rangle$ such that β_{i+1} extends β_i and $\bigcup_{i \in \mathbb{N}} \beta_i = \chi_B$.


We initialise $\alpha_0 = \beta_0 = 0$. Thereafter...

- Stage $2s + 1$. Let x be the first number not in the domain of α_{2s} . [That is to say, it is $\text{length}(\alpha_{2s})$ co's we start counting at 0.] If there are any η_a that are end-extensions of β_{2s} such that $\{s\}^{\eta_a}(x) \downarrow$ then use the least such a and set α_{2s+1} to be $\alpha_{2s} :: y$, where y is the least element of $\{0, 1\} \setminus \{\{s\}^{\eta_a}(x)\}$. [Beware overloading of braces]. And β_{2s+1} is set to be $\eta_a :: 0$. If there is no such η_a then set $\alpha_{2s+1} := \alpha_{2s} :: 0$ and $\beta_{2s+1} := \beta_{2s} :: 0$.
- Stage $2s + 2$. Let x be the first number not in the domain of β_{2s+1} . [That is to say, it is $\text{length}(\beta_{2s+1})$ co's we start counting at 0.] If there are



any η_b that are end-extensions of α_{2s+1} such that $\{s\}^{\eta_b}(x) \downarrow$ then use the least such b and set β_{2s+2} to be $\beta_{2s+1} :: y$, where y is the least element of $\{0, 1\} \setminus \{\{s\}^{\eta_b}(x)\}$. [Again, beware overloading of braces]. And α_{2s+2} is set to be $\eta_b :: 0$. If there is no such η_b then set $\alpha_{2s+2} := \alpha_{2s+1} :: 0$ and $\beta_{2s+2} := \beta_{2s+1} :: 0$.

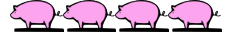
The idea of the construction is that at stage $2s$ (resp $2s+1$) you do something to ensure that $B \neq \{s\}^A \text{“}\mathbb{N}$ (resp. something to ensure that $A \neq \{s\}^B \text{“}\mathbb{N}$.) At the end of the construction the union of the α s is χ_A and the union of the β s is χ_B . ■

We have constructed A and B , neither Turing-reducible to the other, but we have done it by appealing to an oracle for the halting set (at the point on p 138 indicated by the ‘’ sign in the margin). The result is that they are \leq_T the halting set. However we haven’t ensured that they are \leq_m the halting set (remember that that \leq_T contains more ordered pairs than \leq_m , so that—for example— $\mathbb{N} \setminus \mathbb{K} \leq_T \mathbb{K}$). So the construction doesn’t guarantee that they are semidecidable. That is our next challenge.

10.2.1 Friedberg-Muchnik

THEOREM 25 *Friedberg-Muchnik*

There are $<_T$ -incomparable degrees of semidecidable sets.



Proof:

We are trying to build two sets $A, B \subseteq \mathbb{N}$ such that neither is recursive in the other. In particular neither of them can be recursive *tout court*—decidable. Both A and B are constructed as a union of finite approximants: $\langle A_i : i < \omega \rangle$ and $\langle B_i : i < \omega \rangle$ and this will make them semidecidable, which is clearly the best we can hope for. The fact that neither A nor B are decidable means that the A_i (resp. the B_i) cannot be ordered by end-extension, because that would mean A and B could be enumerated in increasing order and that would make them decidable. (See exercise 78.) However the A_i are totally ordered by \subseteq . Do they start off empty, with $A_0 = B_0 = \emptyset$? A useful thought is that it actually doesn’t matter a damn what finite sets A_0 and B_0 are. (In fact my guess is that we can even take A_0 and B_0 to be decidable moieties, like the odds and the evens.) We have countably many conditions all of which are incredibly easy to satisfy (= can be satisfied in infinitely many ways), and we can satisfy any finite bundle of them with our hands tied behind our back. The only hard part is to satisfy them all simultaneously. The reader might perhaps be reminded at this point of the Baire Category Theorem, in the form that a countable family of dense open sets has nonempty intersection. Managing to land inside any one dense open set is a piece of cake. Arranging for a “tail-event” like landing in all of them requires a bit of ingenuity.

We wish to ensure that for no e is $\{e\}^A$ the characteristic function for B nor is $\{e\}^B$ the characteristic function for A . The **requirements** are $A \neq \{e\}^B \text{“}\mathbb{N}$; $B \neq \{e\}^A \text{“}\mathbb{N}$, for each $e \in \mathbb{N}$

Each requirement $A \neq \{e\}^B$ “ \mathbb{N} is looking for a **witness**, an $x \in \mathbb{N}$ s.t. $\{e\}^B(x) = 0$ (which will say that x is not a member of the e th set computable from B) or $\{e\}^B(x) \uparrow$. When this happens we can put x into A . Thus x will be the desired witness to the fact that $\{e\}^B$ “ \mathbb{N} is not the characteristic function for A . (*Mutatis mutandis* swapping A and B .) [You might think that x could be a witness if $\{e\}^B(x) = 1$ (which will say that x is a member of the e th set recursive in B)—so that we then make sure never to put x into A —but we never put things into $\mathbb{N} \setminus A$, only into A .]

Each requirement has its own list of potential witnesses. The lists are disjoint, and written in increasing order. We make them disjoint so that no number is compelled to discharge more than one requirement. It’s not that no number can discharge more than one requirement, it’s that we can arrange things so that no number is called upon to, and it keeps things simple. *À propos* of the observation that A_0 and B_0 don’t have to be empty but could be decidable moieties . . . what we really do need is that all the lists of candidate witnesses for the a -requirements should be disjoint from A_0 (and the B -lists similarly of course).

Earlier requirements have higher priority than later requirements. At any stage a requirement has a **barrier** and a list of candidate witnesses.

At stage n you run the first n requirements for n steps, each processing the current head of its witness list, consulting oracles A_n and B_n . . . by which we mean the following. $\{e\}$ is allowed to ask about membership of $\{m : m \leq \sup(A_n)\}$ (*mutatis mutandis* $\{m : m \leq \sup(B_n)\}$). The point is that if $\{e\}$ halts on 17 (say) when consulting this oracle and gives 0 (so that we want to put 17 into B) it might have done so because it asked whether or not $3 \in \{m : m \leq \sup(A_n)\}$, and got the answer ‘no’, whereas 3 later got put into A .

Any requirement that asks for information outside that initial segment is told to crash (for that round). As long as it asks only about membership in $\{m : m \leq \sup(A_n)\}$ it gets an answer which will be **yes** or **no**.

A computation of $\{e\}^A(n)$ **merits attention** when it halts with output 0. Then we put n into B . (*mutatis mutandis* for B). Then the requirement is met. Every time a requirement is met it freezes (“Bank!!”) an initial segment of A (or B) which means that, for all requirements, it deletes—from the list of candidate witnesses for all requirements of lower priority—all the candidates below the barrier.

Clashes happen when a decision of a lower-priority requirement is overruled by a higher-priority requirement putting up a barrier that voids the computation it (the lower-priority requirement) has made.

A requirement might decide to put a number into A (or into B). When it freezes it thereby erases from lists-of-potential-witnesses for requirements of lower priority all witnesses that lie in the frozen area. This inevitably resets some computations.

A requirement r might be feeling happy, thinking it has found a witness. The purported witness is a witness as long as the initial segment frozen by r is indeed frozen. However a higher-priority requirement might come along and

write something into the frozen area, with result that—as it might be—17 is now a member of A when r has been acting on the assumption that it wasn't. So the witness that r had been banking on is no longer a witness, and r has to try another candidate. However this can happen only finitely often, and r has infinitely many candidates to play with.

Pin the following to the wall.

Once you put something into A (or into B) you never take it out again.
 You put things into A (or into B) but never into their complements.
 You do NOT add members in increasing order: A and B are NOT decidable.
 The n th requirement can be reset at most $2^n - 1$ times.
 You can prove by induction on the requirements that they are satisfied in the limit.

Here's another way in. Imagine you are the daemon whose job it is to look after the B -requirement $\{17\}^A \neq \chi_B$. When the bell strikes for the start of round n you look at the head of your list of potential candidates— x , say—and compute $\{17\}^A(x)$ for n steps. In the process you might be called upon to consult A . At this stage you only have access to A_n , but you consult it anyway.

- If you ask your membership-of- A question of something greater than $\sup(A_n)$ you crash. Smackie handy. Sit in the corner until the next round.
- If you halt and get output 1 then that's no use to you. Discard x and sit on your hands until the next round (when you will use the next thing after x in your list of candidate witnesses)
- If you halt and get output 0 you are pleased: we can put x into B_{n+1} and you are satisfied for the moment. The system managers then raise the barriers for all A -requirements of lower priority¹ so that none of them can write anything into A that undermines your reason for putting x into B . That is to say, for every A -requirement of lower priority, they erase—from that requirement's list of candidates—all the candidates that are smaller than $\sup(A_n)$.

Once you are satisfied you sit out subsequent rounds—unless something bad happens. Something bad?! Well, a daemon guarding a requirement of higher priority than you might put some a into A which is smaller than some of the things in A_n . This can be a problem because it could be that your decision that $\{17\}(x) \downarrow = 0$ happened because you asked the oracle if a was in A and it said **no**—whereas it now turns out that the correct answer is **yes**! So you are back to square one.

It's important to remember that we don't now delete x from A . We never delete anything! We leave it in. It no longer serves its original purpose but it isn't actually doing any harm.

¹Why stop there? Why not raise the barriers on *all* requirements? because you might end up going round in circles!

10.2.2 Omitting Types

I insert this here because people say that the priority method has deep connection with omitting types. I still don't understand why, but you might!

A *type* in a propositional language \mathcal{L} is a set of formulæ (a *countably infinite* set unless otherwise specified).

For T an \mathcal{L} -theory a *T -valuation* is an \mathcal{L} -valuation that satisfies T . A valuation v *realises* a type Σ if $v(\sigma) = \mathbf{true}$ for every $\sigma \in \Sigma$. Otherwise v *omits* Σ . We say a theory T *locally omits* a type Σ if, whenever ϕ is a formula such that T proves $\phi \rightarrow \sigma$ for every $\sigma \in \Sigma$, then $T \vdash \neg\phi$.

THEOREM 26 *The Omitting Types Theorem for Propositional Logic.*

Let T be a propositional theory, and $\Sigma \subseteq \mathcal{L}(T)$ a type. If T locally omits Σ then there is a T -valuation omitting Σ .

Proof:

By contraposition. Suppose there is no T -valuation omitting Σ . Then every formula in Σ is a theorem of T so there is an expression ϕ (namely ‘ \top ’) such that $T \vdash \phi \rightarrow \sigma$ for every $\sigma \in \Sigma$ but $T \not\vdash \neg\phi$. Contraposing, we infer that if $T \vdash \neg\phi$ for every ϕ such that $T \vdash \phi \rightarrow \sigma$ for every $\sigma \in \Sigma$ then there is a T -valuation omitting Σ . ■

However, we can prove something stronger.

THEOREM 27 *The Extended Omitting Types Theorem for Propositional Logic*

Let T be a propositional theory and, for each $i \in \mathbb{N}$, let $\Sigma_i \subseteq \mathcal{L}(T)$ be a type. If T locally omits every Σ_i then there is a T -valuation omitting all of the Σ_i .

Proof:

We will show that whenever $T \cup \{\neg A_1, \dots, \neg A_i\}$ is consistent, where $A_n \in \Sigma_n$ for each $n \leq i$, then we can find $A_{i+1} \in \Sigma_{i+1}$ such that $T \cup \{\neg A_1, \dots, \neg A_i, \neg A_{i+1}\}$ is consistent.

Suppose not, then $T \vdash (\bigwedge_{1 \leq j \leq i} \neg A_j) \rightarrow A_{i+1}$ for every $A_{i+1} \in \Sigma_{i+1}$. But, by assumption, T locally omits Σ_{i+1} , so we would have $T \vdash \neg \bigwedge_{1 \leq j \leq i} \neg A_j$ contradicting the assumption that $T \cup \{\neg A_1, \dots, \neg A_i\}$ is consistent.

Now, as long as there is an enumeration of the formulæ in $\mathcal{L}(T)$, we can run an iterative process where at each stage we pick for A_{i+1} the first formula in Σ_{i+1} such that $T \cup \{\neg A_1, \dots, \neg A_i, \neg A_{i+1}\}$ is consistent. This gives us a theory $T \cup \{\neg A_i : i \in \mathbb{N}\}$ which is consistent by compactness. Any model of $T \cup \{\neg A_i : i \in \mathbb{N}\}$ is a model of T that omits each Σ_i . ■

There is a version of this theorem for predicate (first-order) logic. (It deals with n -types for $n > 0$.) I doubt if we will get round to it. Look at [67], theorem 6.62.

10.2.3 Baker-Gill-Solovay and $P = NP$?

See [4].

THEOREM 28 *There are oracles with respect to which $P = NP$ and also oracles with respect to which $P \neq NP$.*

Proof:

The reason for the appearance of this result *here* is that the proof of the existence of an oracle \mathcal{O} such that—relative to \mathcal{O} — $P \neq NP$, uses a priority argument like that used in the proof of Friedberg-Muchnik. We will write “ $P^{\mathcal{O}} = NP^{\mathcal{O}}$ ” and “ $P^{\mathcal{O}} \neq NP^{\mathcal{O}}$ ”.

For the first part (an oracle \mathcal{O} such that $P^{\mathcal{O}} = NP^{\mathcal{O}}$) we show—and this bit sounds as if it should be easy—that if \mathcal{O} is an oracle that is *PSPACE*-complete (or *EXPTIME*-complete) then $P^{\mathcal{O}} = NP^{\mathcal{O}}$. I’m planning to leave this part to the reader.

The second part requires a bit more work.

First a few definitions

Recall from section 5.1 that $|x|$ is the length of the string x , and 1^n is the string of length n consisting entirely of 1s.

We need a new definition. For a language $B \subseteq \{0,1\}^*$ let $U_B \subseteq \mathbb{N}$ be $\{|x| : x \in B\}$.

We enumerate all oracle-consulting Turing-machines. Because of the way the story has come down to me the oracles that are consulted by the oracle-consulting Turing-machines are *languages*—namely infinite sets $\subseteq \{0,1\}^*$ —whereas the machines themselves have *numerals* (rather than strings) as input and output.

The idea is to cook up a language $B \subseteq \{0,1\}^*$ with the property that

$$U_B \notin P^B \text{ but } U_B \in NP^B$$

The second is easy: it is sufficient for a machine to guess a particular string of the desired length. The first requires a bit more work. The Turing machine is trying to establish, by interrogating B , whether or not B contains a string of length n . It’s allowed to ask about the membership of any particular string of course, but it can take only a polynomial amount of time, and there are exponentially many strings of length n that might be in B and about which it might have to ask.

We will construct B as a union of a \subseteq -increasing sequence $\langle B_i : i \in \mathbb{N} \rangle$ of finite B_i as usual. (This time the B_i are ordered by end-extension. There is no reason to try to stop B from being decidable: computability-in-polytime is a much stronger condition than decidability).

At stage i we obtain B_{i+1} from B_i by [possibly] putting something into B_i . The rôle of this something is to ensure that T_i , the i th Turing machine (with access to B_i) does not compute—in polytime—the characteristic function for

U_B . We need an input on which it gives an incorrect answer. This input will be n_i , a natural number chosen to be bigger than the length of any string in B_i . We fire up T_i with input n_i and allow it to consult B_i .

If T_i halts in time $< n_i^{n_i}$ and says **yes** we must be sure to put no strings of length n_i into B . We can secure that end by simply doing nothing. If T_i halts in time $< n_i^{n_i}$ and says **no** (thereby alleging that B does not contain any strings of length n_i) then we put 1^{n_i} into B_{i+1} . This will ensure that T_i (when consulting B) gives the wrong answer to the question of whether or not B contains a string of length n_i .

Is this enough? It's clearly the right *idea*. The problem is that if T_i doesn't give an answer to n_i in time $< n_i^{n_i}$ it might nevertheless give a correct answer in polytime for all inputs—by having a huge constant of proportionality or by having an exponent higher than n_i . If we only investigate it once we might find that on the one input n it is still running after n^n steps (so we do nothing) but it still eventually halts in polytime every time and gives the right answer. The key is to give us infinitely many opportunities to bugger up any given Turing machine. Let T be a Turing machine that computes a total function with range $\{\text{yes}, \text{no}\}$ and runs in time $k \cdot n^e$ (polytime in n). Then, for n suff large, $n^n > k \cdot n^e$, so if we sample the machine's behaviour for sufficiently large n we will be able to refute it.

Accordingly the plan is to enumerate the Turing machines in such a way that every Turing machine is visited infinitely often. At each stage we perform the action described two paragraphs above. That is, if the machine being examined does not halt we do nothing; if it halts and says **yes**, do nothing; if it halts and says **no** we put 1^{n_i} (but of course any string of length n_i will do) into B_{i+1} . If, at the end of time—after infinitely many visits—it still has not halted then it is evidently not a total function and we don't have to worry about it. (We will have taken no steps in regard to it but then none need to be taken)

This strategy will at least ensure that every Turing machine that halts in polynomial time on every input doesn't correctly answer membership questions about U_B . If it reliably halts in polytime then eventually we will reach an n_i bigger than the exponent and from that visit onwards it has enough time to halt if it wants to. ■

Ben Millwood writes:

Consider the oracle machine that does the following:

on input n , construct the string 1^n and consult the oracle for B , returning its result.

Clearly this machine always halts in polynomial time. Moreover, B that you construct in your B-G-S proof consists only of strings that are repeated ones, so the above machine decides U_B . Hence, $U_B \in P^B$, contrary to your proof.

I have a pretty good idea of what is going on here but the margin of time between now and my two (!) exams tomorrow is too narrow to contain a full explanation. I think you really have to pay attention to the strings that each

machine queries, so you don't end up changing your mind about what's in B (as it is, it seems that a machine can query 1^n with B_i which says no, but later you put 1^n into B_j for some $j > i$).

There's an alternative proof in the lecture notes for Computational Complexity <http://www.qi.damtp.cam.ac.uk/node/251> (page 34). I think it doesn't address what happens if your machine run doesn't halt in its allotted time, but I think if you just count that as a rejection then the proof goes through.

(Another conspicuous difference between your proof and that one is that the definitions of U_B are different. You have just $|x|$, which is a number, versus that proof's $1^{|x|}$, a string in a language. This matters because the size of the input is the parameter to the polynomial for which your machine is allowed to run, and $|x|$ can be given as an input in $\log|x|$ bits.)

Ben

Chapter 11

Ordinals, Fast-growing Functions, Consistency and Totality Proofs

11.1 The Ordinal ϵ_0 and the Consistency of Peano Arithmetic

Recall from definition9 that functions $\mathbb{N} \rightarrow \mathbb{N}$ can be preordered by *dominance* thus: $f < g$ if for all suff large n , $f(n) < g(n)$. It's pretty clear that $\mathbb{N}[X]$ (the polynomials in one variables with coefficients in \mathbb{N}) is linearly ordered by dominance in order type ω^ω . (just replace every occurrence of ' x ' by ' ω '). Now consider the slightly fatter set, containing functions of one variable $\mathbb{N} \rightarrow \mathbb{N}$ which is inductively generated from the set of polynomials in one variable with coefficients in \mathbb{N} by allowing x^f whenever we have f . Thus we have things like

$$x^{x^{x^9+3}+x^{x^2}+x^2+5} + x^{x^5+10} + x^{1000}$$

It can be easily seen analogously that if we order these by dominance we get ϵ_0 . However if we consider the somewhat larger inductively defined family of expressions that contains all the functions in (i) and is closed under exponentiation, so it contains things like

$$(x^{x^{x^9+3}+x^{x^2}+x^2+5} + x^{x^5+10})^{(x^{x^{x^9+3}+x^{x^2}+x^2+5}+x^{1000})}$$

then it is far from obvious that the set of [the functions denoted by] these expressions is totally ordered by dominance, let alone well-ordered by dominance, but as it happens it is. The order-type has not been computed, tho' some bounds are known. see [48], [20] and [40].

I will now sketch how to prove the consistency of Peano Arithmetic by transfinite induction. (I have lifted this from the *first* edition of [44]; this material has been removed from later editions.) The proof goes back to [54].

[message to reader: The material in the lectures can be found in the relevant chapters of www.dpmms.cam.ac.uk/~tf/chchlectures.pdf.]

We have a system of arithmetic with an ω -rule:

$$\frac{\Gamma \vdash F(1) \quad \Gamma \vdash F(2) \dots \Gamma \vdash F(n) \dots}{\Gamma \vdash (\forall n)(F(n))}$$

It also has a rule of cut:

$$\frac{\Gamma \vdash \Delta, A, \quad A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}$$

The assumptions (the leaves of the proof-trees in this system) are true atomic sentences of the kind ‘ $0 = 0$ ’, ‘ $0 \neq S(0)$ ’ and suchlike (no variables!)

Proofs in this system can be seen as countable trees (each node [inference] might have a countable infinity of premisses). Clearly we are not going to be interested in proofs that have infinite paths—after all, any formula whatever can be supplied with a proof with an infinite path. We are interested only in proofs whose corresponding trees have no infinite paths. Such a proof can be decorated with ordinals in the standard manner from chapter 3. How large a countable ordinal might one need to decorate a tree of a proof in this system? There are only countably many formulæ in the language of arithmetic so each node can have only countably many immediate predecessors, and a sup of countably many countable ordinals is countable. This means that the rank of a proof must be countable¹. This invites us to consider, for a limit ordinal $\alpha < \omega_1^{CK}$, the collection $T(\alpha)$ of those formulæ that have proofs whose trees have rank $< \alpha$. For suitable α , $T(\alpha)$ might be closed under the finitary rules of inference and thereby be a set deductively closed in the usual sense, to wit: a *theory*.² Theories arising from countable ordinals in this sense have the potential to be very interesting ... particularly if they are consistent! Mendelson [44] says that $T(\omega_1)$ is the first-order theory of the standard model.

When do we know that such a theory is consistent? One way of detecting that a theory is consistent is to prove cut-elimination for it. This is because there is no cut-free proof of the **false**.

This is roughly the point of departure for the analysis in [44]. The labellings of the trees that he uses there differ slightly from the rank function on a naked tree but the idea is the same. Decorate the proof tree by labelling endpoints with ‘0’, and the rank of a node is the sup of rank + 1 of the nodes above it—unless the node corresponds to a structural rule, in which case the rank is the same as the rank of its predecessor.

If we eliminate cuts from a proof of a certain rank, what can we say about the rank of the cut-free proof we obtain? We get blowup in the case where the cut formula was introduced by an application of the ω -rule. It turns out that

Beware—this cut rule is cut as in Mendelson not cut as in Gentzen.

¹There is enough structure around for us not to need countable choice to prove this.

²I shall equivocate between thinking of $T(\alpha)$ as a theory and thinking of it as a body of proofs.

any proof of rank β with a cut C on a formula of length l can be transformed into a proof of potentially higher rank, but with the desirable feature that the cut C has been replaced by cut(s) on formulæ of length $< l$. The rank of the new proof may be greater, but at all events no greater than 2^β .

Suppose we can show that if we eliminate the cuts from proofs in $T(\alpha)$ we obtain a proof in $T(\beta)$ with $\beta > \alpha$. We will have gone to great effort (we will done transfinite induction up to β to prove termination of the process of elimination of cuts) and all we have to show for it is a proof that $T(\alpha)$ is consistent if $T(\beta)$ is consistent—and that of course was obvious all along, since $\beta > \alpha$ whence $T(\alpha) \subseteq T(\beta)$. However if $\beta = \alpha$ (so that if we take a proof in $T(\alpha)$ and we remove the cuts from it the resulting proof is still in $T(\alpha)$) then our work will not be in vain, for it will have shown that $T(\alpha)$ is consistent—using only transfinite induction up to α —rather than beyond it as far as β .

(What had been worrying me here is that if the proof is infinite there may be no cut of greatest rank, so how can we prove that the process halts? The point is that all proofs that arise from embedding proofs of finitary arithmetic in this system are finitary and have finite cut degree.)

EXERCISE 100 ω is the first solution to the equation $\alpha = 2^\alpha$. What is the next solution?

DEFINITION 26 An ordinal α is an epsilon number iff it is a solution to $\alpha = \omega^\alpha$, or equivalently iff the ordinals below it are closed under exponentiation.

The foregoing means that, for any ϵ -number ϵ , we can prove by transfinite induction on ‘ α ’ that

if $\alpha < \epsilon$ then every formula that has a proof of rank $\leq \alpha$ has a cut-free proof of rank below ϵ

Thus

REMARK 18

If α is an ϵ -number and we can induct as far as α (i.e., we have a wellordering of length α) then we can recursively eliminate cuts from proofs in $T(\alpha)$ while remaining inside $T(\epsilon)$ thereby proving $T(\alpha)$ consistent.

In particular, if we can induct as far as ϵ_0 , then this will show that $T(\epsilon_0)$ is consistent. So: what do we know about this system $T(\epsilon_0)$ whose consistency we can prove if we can induct as far as ϵ_0 ? It turns out that $T(\epsilon_0)$ is at least Peano Arithmetic.

11.2 The Goodstein function

The Goodstein function, known as G (for obvious reasons) is an example of a function that is manifestly computable but very far-from-manifestly total. To discover what $G(x)$ is to be, we first express x as a sum of powers of 2, and then

express the *exponents* as sums of powers of two, and so on recursively. Thus, if we do this to—say—37, we get

$$\begin{aligned} 32 + 4 + 1 = \\ 2^5 + 2^2 + 1 = \\ 2^{4+1} + 2^2 + 1 = \\ 2^{2^2+1} + 2^2 + 1 \end{aligned}$$

This is the *extended* base 2 representation of a number. I have written the ‘2’s in **boldface** to remind us that this expression is in extended base 2.³ Now replace all the 2’s by 3’s and subtract 1. This gives us $3^{3^3+1} + 3^3$. The result is still in extended base 3. Now replace all ‘3’s by ‘4’s

$$4^{4^4+1} + 4^4$$

and subtract 1 to get

$$4^{4^4+1} + 4^4 - 1$$

But this is not in extended base 4 representation because of the minus sign, and we have to express $4^4 - 1$ as a sum of powers of 4 with a few 1’s left over, thus

$$4^{1+1+1} + 4^{1+1+1} + 4^{1+1+1} + 4^{1+1} + 4^{1+1} + 4^{1+1} + 4 + 4 + 4 + 1 + 1 + 1$$

so the whole thing is

$$4^{4^4+1} + 4^{1+1+1} + 4^{1+1+1} + 4^{1+1+1} + 4^{1+1+1} + 4^{1+1} + 4^{1+1} + 4^{1+1} + 4 + 4 + 4 + 1 + 1 + 1$$

(The ‘4’s are still in boldface to remind us that this number is being written in extended base 4.)

Then we can replace all ‘4’s by ‘5’s, subtract 1 and continue. How long can we continue doing this? These numbers seem to go on getting bigger and bigger!

However, if we try it on 2, the process stops: 2 becomes $3 - 1$ which in extended base 3 is $1 + 1$ becomes 1 becomes 0. If we try it on 3 we get $2^1 + 1$ becomes 3^1 becomes $4^1 - 1 = 1 + 1 + 1$ which will decay to 0 as before. we are now in a position to announce a definition:

$G(x)$ is the length of the sequence of terms generated in this way (if it is defined).

Thus the Goodstein function is actually a cost function for the computable function $\mathbb{N} \rightarrow \{0\}$ defined by

```

INPUT  $n$ 
write  $n$  in extended base 2
 $i =: 3$ 
REPEAT
```

³there is a reason for the choice of a Greek font for the first letter of ‘extended’.

```

 $n$  =: replace  $i + 1/i$  in representation of  $n$ 
rewrite in extended base  $i + 1$  representation;
subtract 1
UNTIL
 $n = 0$ 
PRINT  $n$ 

```

and this definition makes it clear that G is μ -recursive.

Thus $G(2) = 4$ and $G(3) = 5$. $G(4)$ is quite large but can be computed by hand. One might think that for at least some larger numbers the sequence goes on for ever; remarkably⁴, this is not so: G is total computable.

THEOREM 29

If there is a wellordering of length ϵ_0 then $G(n)$ is defined for all $n \in \mathbb{N}$.

Proof

The key to the proof is to spot the trick that the conjuror is playing on you. Your attention is being directed to the apparently inexorably increasing sequence of numbers, so that you don't notice the thing that is actually decreasing.

Start with a number in extended base 2 representation. Consider the ordinal in Cantor normal form obtained from this expression by replacing every '2' by an ' ω '.⁵ In our first example above (37), this would be $\omega^{\omega^{\omega}+1} + \omega^{\omega} + 1$, since the extended base 2 representation of 37 was $2^{2^2+1} + 2^2 + 1$.

To every number in the sequence we are building (whose length will be $G(n)$) we will make correspond an ordinal in precisely this way—(That was why I wrote the base in **boldface** so that we can say:—simply replace the boldface number by ω . Numerals not written in boldface are *not* replaced by ' ω '. Thus for each i the i th member of the sequence (on the left) will correspond to the ordinal to its right:⁶

⁴Try proving by induction on ' n ' that $G(n)$ is defined; you will get nowhere.

⁵For these purposes we take the Cantor Normal Form of an ordinal to be the wordy, verbose version that does not allow multiplication by naturals, so that an ordinal is a sum of powers of ω .

⁶I have reverted to the style of Cantor normal form that allows multiplication by naturals in order to save space!

| | |
|---|---|
| $2^{2^2+1} + 2^2 + 1$ | $\omega^{\omega^\omega+1} + \omega^\omega + 1$ |
| $3^{3^3+1} + 3^3$ | $\omega^{\omega^\omega+1} + \omega^\omega$ |
| $4^{4^4+1} + 3 \cdot 4^3 + 3 \cdot 4^2 + 3 \cdot 4 + 3$ | $\omega^{\omega^\omega+1} + \omega^3 \cdot 3 + \omega^2 \cdot 3 + \omega \cdot 3 + 3$ |
| $5^{5^5+1} + 3 \cdot 5^3 + 3 \cdot 5^2 + 3 \cdot 5 + 2$ | $\omega^{\omega^\omega+1} + \omega^3 \cdot 3 + \omega^2 \cdot 3 + \omega \cdot 3 + 2$ |
| $6^{6^6+1} + 3 \cdot 6^3 + 3 \cdot 6^2 + 3 \cdot 6 + 1$ | $\omega^{\omega^\omega+1} + \omega^3 \cdot 3 + \omega^2 \cdot 3 + \omega \cdot 3 + 1$ |
| $7^{7^7+1} + 3 \cdot 7^3 + 3 \cdot 7^2 + 3 \cdot 7$ | $\omega^{\omega^\omega+1} + \omega^3 \cdot 3 + \omega^2 \cdot 3 + \omega \cdot 3$ |
| $8^{8^8+1} + 3 \cdot 8^3 + 3 \cdot 8^2 + 2 \cdot 8 + 7$ | $\omega^{\omega^\omega+1} + \omega^3 \cdot 3 + \omega^2 \cdot 3 + \omega \cdot 2 + 7$ |
| \vdots | \vdots |
| $15^{15^{15}+1} + 3 \cdot 15^3 + 3 \cdot 15^2 + 2 \cdot 15$ | $\omega^{\omega^\omega+1} + \omega^3 \cdot 3 + \omega^2 \cdot 3 + \omega \cdot 2$ |
| $16^{16^{16}+1} + 3 \cdot 16^3 + 3 \cdot 16^2 + 16 + 15$ | $\omega^{\omega^\omega+1} + \omega^3 \cdot 3 + \omega^2 \cdot 3 + \omega + 15$ |
| \vdots | \vdots |
| \vdots | \vdots |
| \vdots | \vdots |

So the length of the sequence we are building will be the same length as a particular decreasing sequence of ordinals. Why is it decreasing? The entries in the left-hand column keep increasing as long as there are boldface numerals around, because we increase each boldface numeral by one at each stage. In the short term, this more than compensates for the 1 that we keep subtracting. In contrast the entries on the right have ω instead of a boldface numeral, and we do not increase the ω , so there is nothing to counteract the slow attrition of subtraction of 1.

Any decreasing sequence of ordinals must be finite, so the original sequence of numbers was finite, so $G(n)$ is defined. In this case the ordinals we are using are all below ϵ_0 , so it will suffice to have a wellordering of that length. ■

Once you understand the proof of theorem 29 you can see immediately that from the same assumption used above—namely that the set of ordinals below ϵ_0 is available to us, along with its ordering, and the information that that ordering is wellfounded—we can prove not only the totality of G but also the totality of any function computed like G but with the tweak that we are not required to decrement *every single time* we increase the base, as long as we promise, when we find ourselves at a nonzero number, to decrement at some point. Consider what one might call the *Nondeterministic Goodstein Function* where at each stage in the computation of $G(n)$ one makes a random choice about whether to decrement or not. Clearly an analysis analogous to the analysis above will establish that any nonterminating computation of the Nondeterministic Goodstein function has only finitely many decrements. Let us minute this fact.

REMARK 19 *If there is a wellordering of length ϵ_0 then the nondeterministic $G(n)$ is defined for all $n \in \mathbb{N}$.*

Why the odd title?

Goodstein's paper was entitled "On the Restricted Ordinal Theorem"; "The restricted ordinal theorem" was the name current at that time for the allegation usually expressed nowadays by the form of words "the ordinals below ϵ_0 are wellordered". This is loose talk: ϵ_0 is an ordinal, and for any ordinal α the ordinals below α are wellordered: that's a complete triviality and cannot be used to prove anything. The bit that does the work is the assumption that *there is a wellordering of length α* . For consider how the proof would proceed in a formal system: for each input to G we define a decreasing function from \mathbb{N} to the ordinals below ϵ_0 , and we need the range of that function to be a set, so we need that collection to be a set, and we need the ordering on it to be a wellordering. One might suspect that Goodstein's purpose in devising this rather odd function was to exhibit a computable total function whose totality is not demonstrable in Peano arithmetic, precisely because the totality relies on an induction that is not available in PA. However⁷ the reason is more likely to do with the view—current around that time—that ϵ_0 was the supremum of those ordinals that had a finite description. That in turn may be something to do with the fact that the ordinals below ϵ_0 are closed under $+$, \times and exponentiation, and that those three operations are the only operations in the Doner-Tarski sequence that correspond to actual operations on wellorderings. In case you didn't know, α^β is the order type of the set of functions $B \rightarrow A$ which are 0 at all but finitely many places, ordered colex—where $\text{otp}(\langle A, <_A \rangle) = \alpha$ and $\text{otp}(\langle B, <_B \rangle) = \beta$. The next operation— f_s , the "tower of exponents"—has no concrete representation of this kind. This is because it grows faster than $n \mapsto \beth_k(n)$ for any $k \in \mathbb{N}$, so we cannot find any expression $R(x, y)$ in the languages of set theory such that $|y| = |f_3(x, x)|$. Actually it's not the next one after exponentiation that explodes the type hierarchy but a slightly later one.

Worth spelling out in some detail

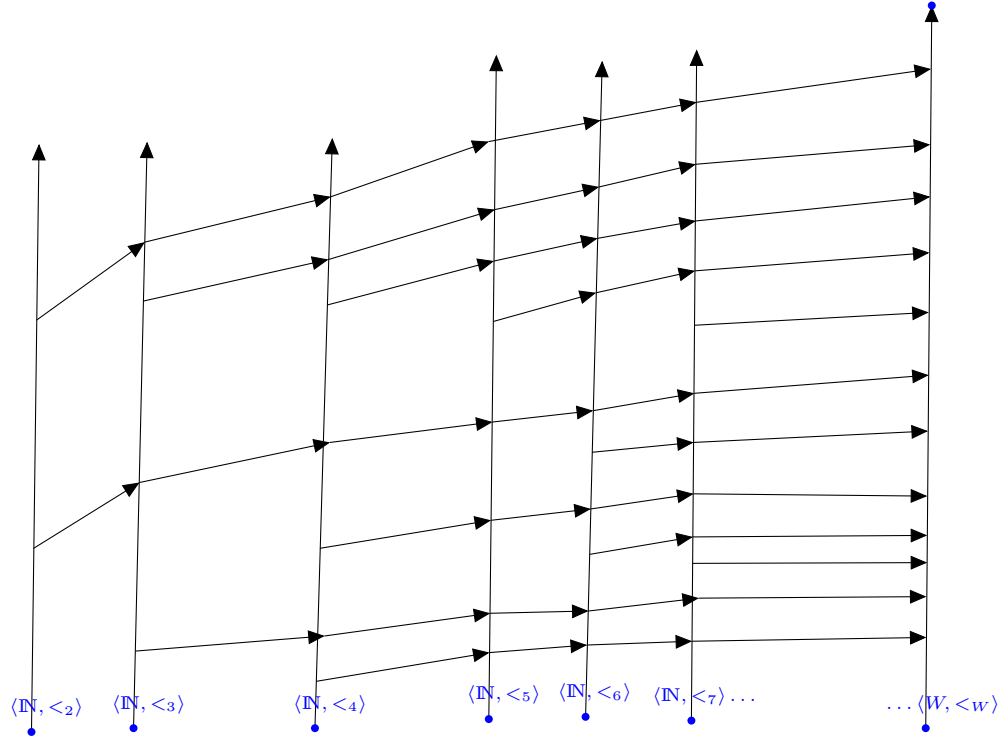
I do not know if this consideration is explicit in the literature of the 1930's and 40's ... it could be worth checking.

We'd better complete Goodstein's [putative] project by showing a converse to 19, namely that if the nondeterministic Goodstein function is total then PA is consistent.

REMARK 20 *There is a definable total ordering of \mathbb{N} with the property that it is of length ϵ_0 if every nonterminating run of the nondeterministic Goodstein function has only finitely many decrements.*

Proof:

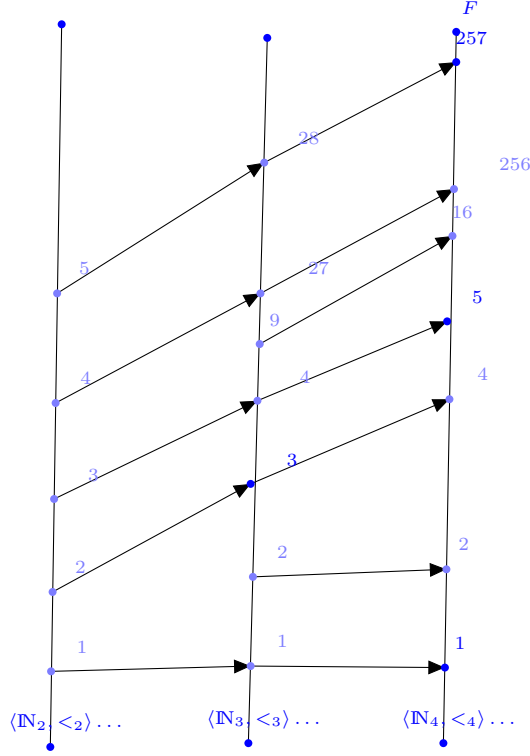
⁷Thank you Stan Wainer!



We construct the total order as a direct limit. In fact it will be not merely definable but also actually decidable. And we can exhibit it without the assumption that the nondeterministic Goodstein functions is total; we don't need that assumption until we attempt to prove that the ordering is a *wellordering*.

Let $\langle \langle \mathbb{N}_n, <_n \rangle : 2 \leq n < \omega \rangle$ be a family of copies of $\langle \mathbb{N}, <_{\mathbb{N}} \rangle$. Define, for each $2 \leq n < \omega$, an injective homomorphism $i_n : \mathbb{N}_n \hookrightarrow \mathbb{N}_{n+1}$ as follows. Given $y \in \mathbb{N}_n$, think of it as written in extended base n . Then replace every ' n ' by ' $n+1$ '; this number is to be our value of $i_n(y)$. Now consider the direct limit ("colimit") of this system, which we will call $\langle W, \leq_w \rangle$. Every element of the direct limit $\langle W, \leq_w \rangle$ is an ω -sequence of natural numbers. Indeed any such sequence is a *computable* function, and thus a natural number, so W is clearly countable. (Actually—assuming countable choice—a direct limit of countably many countable structures is always countable, but never mind). However we will continue to think of elements of W as functions.

Thus, as *per* the slightly more detailed figure that follows, i_2 sends 1 to 1, sends 2 to 3, sends 3 ($= 2+1$) to 4 ($= 3+1$), sends 4 ($= 2^2$) to 27 ($= 3^3$), sends 5 ($= 2^2 + 1$) to 28 ($= 3^3 + 1$) and so on. Similarly i_3 sends 1 to 1, 2 ($= 1+1$) to 2, sends 3 to 4, sends 4 to 5, sends 9 ($= 3^{1+1}$) to 16 ...



Each member of W is a function from a terminal segment of \mathbb{N} , typically a *proper* terminal segment. Consider \mathbb{N}_3 for example. The only numbers in \mathbb{N}_3 that are in the range of i_2 are sums of distinct powers of 3 (since they arise by replacing ‘2’ by ‘3’ in the extended base 2 representation of something). The sequence that is to become the finite ordinal n in $\langle W, <_W \rangle$ is a sequence that starts at \mathbb{N}_{n+1} .

If $<_W$ is illfounded there will be a descending ω -sequence. (We do not need DC for this, since the carrier set is wellordered, being a subset of \mathbb{N} .)

Suppose $f_i : i < \omega$ is a descending sequence in $\langle W, <_W \rangle$. Every f_i , being a member of W , is an ω -sequence, and it starts at \mathbb{N}_i for some i . Without loss of generality we can pass to a subsequence of f so that the sequence of i s s.t. members of the sequence start at \mathbb{N}_i form an increasing sequence. Recall that we are thinking of the f_i as functions on terminal proper segments of \mathbb{N} , so that $f_i(n)$ is not defined if f_i first appears at \mathbb{N}_j with $j > i$. Given this family $f_i : i < \omega$ consider the evaluation sequence g for the modified G function (so that $g(n) \in \mathbb{N}_n$ for all n) defined as follows. Set $g(0)$ to be some natural large enough to ensure that $g(j) > f_1(j)$, where \mathbb{N}_j is the copy of \mathbb{N} where f_1 first appears. The idea is that g decrements only when a new f_i appears. That is to say, $g(n+1) =: i_n(g(n))$ unless \mathbb{N}_{n+1} is one of those copies of \mathbb{N} at which a new f_j starts, in which case $g(n+1) =: i_n(g(n)) - 1$.

It may well be that, for all f_n , it happens that for sufficiently large values of m we have $g(m) <_m f_n(m)$, but the values of m for which this first happens increase monotonically with n . This means that any function f in W that lies entirely $<_W$ -below all the f_i must lie $<_W$ -below g . But, by assumption on g , f now must be the zero element of W .

Finally we have to check that the order type of $<_W$ is indeed ϵ_0 . To do this, we have to find, for any ordinal $\alpha < \epsilon_0$, a sequence which is a member of W to which it corresponds. Every ordinal $\alpha < \epsilon_0$ has a Cantor normal form $\mathfrak{C}(\alpha)$, which is a finite string of characters, so there is an upper bound a on the natural numbers that appear in $\mathfrak{C}(\alpha)$. The ω -sequence that will correspond to α starts in \mathbb{N}_a .

■

Recall at this point the results of chapter 8.2, (for example theorem 19) where we saw how natural assertions that certain functions are total can turn out to be unprovable. What remark 20 gives us is a specific function whose totality implies the consistency of PA.

It seems pretty obvious that the Goodstein function is monotone increasing. However we have to open a can of worms if we want to prove it. This introduces a new topic.

11.3 Hierarchies of fast-growing functions

Need the concept of *predecessor* function; $P_n(\alpha)$

Look at the picture on page 151. P_n is the function you need if you are to obtain the $n + 1$ th ordinal in the right-hand column from the n th ordinal in the right-hand column: P_n of the n th ordinal in the right-hand column is the $n + 1$ th ordinal in the right-hand column. To be precise:

$$\begin{aligned} P_n(0) &:= 0; \\ P_n(\alpha + 1) &:= \alpha; \\ P_n(\lambda) &:= P_n(\lambda_n). \end{aligned}$$

... where λ_n is the n th member of the fundamental sequence for λ . Fundamental sequences (see for example Q 10 on Professor Leader's second example sheet from Part II Logic and Set Theory in 2015) go back to Hardy [30], an article rediscovered by Kreisel, Löb and Wainer. 'P' for predecessor. We need another auxilliary function:

$$\begin{aligned} H_0(n) &:= n; \\ H_\alpha(n) &:= H_{P_n(\alpha)}(n + 1); \\ H_\lambda(n) &:= H_{\lambda_n}(n). \end{aligned}$$

and a function $\text{ord}: \mathbb{N} \times \mathbb{N} \rightarrow \omega_1$ defined so that $\text{ord}(n, m)$ is the ordinal you obtain by writing m in extended base n and then replacing all the ' n 's by ' ω '.

We'd better check that if we replace H_0 by any strictly increasing function $f : \mathbb{N} \rightarrow \mathbb{N}$ with $(\exists k \in \mathbb{N})(\forall n \in \mathbb{N})(f(n) < n \cdot k)$ then we get the same dominance behaviour. This could make an exercise.

The significance of H is as follows:

Evaluate $H_{\text{ord}(k,2)}(2)$. First step gives $H_{P_2(\text{ord}(k,2))}(3)$; then we get, successively:

$$\begin{aligned} & H_{P_3(P_2(\text{ord}(k,2)))}(4); \\ & H_{P_4(P_3(P_2(\text{ord}(k,2))))}(5); \\ & H_{P_5(P_4(P_3(P_2(\text{ord}(k,2)))))}(6); \\ & \vdots \end{aligned}$$

and this continues until we reach an n such that $P_n(P_{n-1}(\dots(\text{ord}(k,2))(n+1)\dots)) = 0$, at which point we return the answer $n+1$. The ' n ' works like a kind of **count** variable that records the length of the evaluation sequence so far. Thus $H_{\text{ord}(k,2)}(2)$ is the length of the descending sequence of ordinals in the right-hand column, starting with $\text{ord}(k,2)$, which is to say, it is $G(k)$. Hang on to this fact: it's useful!

$$G(k) = H_{\text{ord}(k,2)}(2).$$

REMARK 21 *If G is total, so too is H_α for every $\alpha < \epsilon_0$.*

Proof: We prove by induction on \mathbb{N} that $(\forall \alpha < \epsilon_0)(H_\alpha(n) \downarrow)$.

Assume G is total. That is to say $H_\alpha(2) \downarrow$, for all $\alpha < \epsilon_0$. That takes care of the base case, $n = 2$.

Induction step: Suppose true for all $\alpha < \epsilon_0$ that $H_\alpha(n) \downarrow$; we will show by UG on ' α ' that the same goes for $n+1$. Let α be arbitrary. We want $H_\alpha(n+1) \downarrow$. But $H_\alpha(n+1) = H_{\alpha+1}(n)$ and the RHS is defined by induction hypothesis on ' n '. ■

This is clear enough, but it involves reasoning explicitly about ordinals. What are the chances of reproducing this proof (or anything like it) in a theory of natural numbers? Well, instead of ordinals-below- ϵ_0 , we can reason about (gnumbers of) *character strings* for ordinals-below- ϵ_0 . It is simple enough to define a set of natural numbers that are codes for ordinals-below- ϵ_0 , and it is clear that this set will be decidable. We can even define an order $<'$ on the codes which (seen from outside) orders them like the ordinals below ϵ_0 . The tricky part is justifying induction on $<'$. That is to say, the challenge is to prove all instances of

$$(\forall n)[(\forall m <' n)(\phi(m)) \rightarrow \phi(n)] \rightarrow (\forall n)(\phi(n))$$

How might we prove this? One naturally expects to use induction of some sort. The only kind of induction that we have straightforwardly available is

mathematical induction. It is true that transfinite induction over \mathbb{N}^2 can be simulated by a nested induction (“inner loop”) as in the second proof of totality of Ackermann (theorem 8) but that technique offers hope only up to ordinals below ω^ω .

We cannot in fact do this in Peano Arithmetic, and the reason is that transfinite induction up to ϵ_0 enables us to prove the consistency of Peano Arithmetic.

The Hardy hierarchy is a hierarchy of functions $\mathbb{N} \rightarrow \mathbb{N}$ each one dominating all previous ones. There is also . . .

DEFINITION 27 *The Fast-Growing hierarchy.*

$$\begin{aligned} F_0(x) &:= x + 1; \\ F_{\alpha+1}(x) &:= F_\alpha^{x+1}(x); \\ F_\lambda(x) &= F_{(\mathcal{F}_\lambda x)}(x). \end{aligned}$$

(I shall use capital ‘ F ’ rather than lower-case ‘ f ’ to forstall confusion with the Doner-Tarski hierarchy from p. 44.) The fast-growing hierarchy with *finite* subscripts is the **Grzegorzcyk** hierarchy from [29].⁸

It turns out that

REMARK 22 $(\forall \alpha)(F_\alpha = H_{\omega^\alpha})$

EXERCISE 101 (*)

Think of the fast-growing hierarchy as a function F from the second number class to Baire space, $\mathbb{N}^{\mathbb{N}}$. Both these spaces have natural topologies: the second number class has the order topology and $\mathbb{N}^{\mathbb{N}}$ can be thought of as the product (with the product topology) of countably many copies of \mathbb{N} (with the discrete topology).

Is F continuous with respect to these topologies?

There is an obvious possibility of proving by induction on the ordinal subscript that every H_α is total. What one has to think about is the formal system in which such a proof might be couched.

Just to reassure myself that I am in familiar surroundings I shall prove

REMARK 23 *For $\alpha < \omega$, F_α is primitive recursive.*

Proof: Clearly true for $\alpha = 0$. Define **iter** g so that **iter** $(g, n) : m \mapsto (g^n(m))$ by means of the following declaration:

$$\mathbf{iter}(f, 0) \, m := m; \quad \mathbf{iter}(f, (n + 1)) \, m := f(\mathbf{iter}(f, n) \, m)$$

⁸I want a medal for spelling this name correctly. Craig McKay (my first Logic teacher) told me that Grzegorzcyk was usually known in the West as ‘G’—not because he was a spymaster but merely in order to sidestep the challenge to which I have just risen.

we see that $\text{iter}(g, n)$ is primitive recursive as long as g is. Then

$$F_{\alpha+1} : n \mapsto \text{iter}(F_\alpha, n+1) n$$

is primitive recursive as long as F_α is. ■

Indeed there is even a converse: we can show—by analogy with the proof that the Ackermann function dominates all primitive recursive functions—that every primitive recursive function is dominated by an F_n with $n < \omega$.

EXERCISE 102 *Complete this proof sketch from Stan Wainer.*

“For the primitive recursive bounding, you can show that if $f(0, a) = g(a)$ and $f(x+1, a) = h(x, a, f(x, a))$ where both g and h are assumed to be bounded by F_n , then $f(x, a) < F_n(F_n(F_n \dots (F_n(a+x) \dots))$ with $x+1$ iterates of F_n (or something like this). Then you get $< F_n F_n F_n \dots F_n F_n (\max\{x, a\})$ with one extra iterate, since $F_n(b) > 2b$ for $n > 0$.

Since $F_{n+1}(x) = F_n$ iterated $x+1$ times on x , this yields $f(x, a) < F_{n+1}(\max\{x, a\}) < F_\omega(\max\{x, a\})$ for $\max\{x, a\} > n$. F_ω is a version of Ackermann, as can be shown fairly easily by comparison with the original.”

The Goodstein function is roughly F_{ϵ_0} . The modified version where you use base 2 not extended base 2 (so you leave the exponents alone) corresponds to F_{ω^ω} .

11.3.1 Good behaviour of the F_α , and the Schmidt conditions

We would like to establish that every F_α is strictly increasing and F_α dominates F_β whenever $\alpha > \beta$. However this is actually quite tricky, and the attempt to secure it gives rise to very subtle conditions on fundamental sequences. It turns out that—for ordinals below ϵ_0 —all the conditions one needs are in fact satisfied by the “obvious” system of fundamental sequences.

EXERCISE 103 *For α an ordinal, let α' be the least ordinal that is the length of a terminal segment of a wellordering of length α . Prove that α' is always a power of ω .*

Might it be a good idea to think of a family of fundamental sequences as a three-place relation on the ordinals?

EXERCISE 104 (*)

1. Characterise the “obvious” system of fundamental sequences for ordinals below ϵ_0 .
2. Establish that, using those fundamental sequences, F_α is strictly increasing and F_α dominates F_β whenever $\beta < \alpha < \epsilon_0$.

This will lead us to the Schmidt conditions from [55].

11.3.2 Schmidt-coherence

Now we return to the endeavour of showing that a sequence of functions defined in the style of definition ?? will be monotone increasing with each function dominating all earlier ones. The idea is to prove by induction on α that f_α is monotone increasing and dominates all earlier f_β . Given the induction hypothesis it's easy to prove that f_α dominates all earlier f_β . Suppose f_{α_i} is strictly increasing for each $i \in \mathbb{N}$ and later f s dominate earlier f s. If f_α is $\lambda n. f_{\alpha_n}(n)$ then it dominates every α_i . Why isn't strict monotonicity obvious too? If f_α is strictly increasing so is $f_{\alpha+1}$. The hard case is that of limit ordinals.

We want

$$f_\lambda \ n < f_\lambda(n+1).$$

This holds iff

$$f_{\lambda_n} \ n < f_{\lambda_{n+1}}(n+1).$$

But

$$f_{\lambda_n} \ n < f_{\lambda_n}(n+1)$$

because f_{λ_n} is strictly increasing by induction hypothesis. Then to complete the proof it will suffice to show

$$f_{\lambda_n}(n+1) < f_{\lambda_{n+1}}(n+1)$$

which will follow if $(\forall \lambda \forall n)(S(\lambda_n, \lambda_{n+1}))$ where $S(\alpha, \beta)$ is:

$$\alpha < \beta \rightarrow (\forall m)(f_\alpha \ m < f_\beta \ m).$$

Now this clearly isn't going to happen: otherwise what could $f_\omega(0)$ possibly be? Duh! What one can ask for is that $f_{\lambda_{n+1}}$ has overtaken f_{λ_n} by the time argument $n+1$ comes along. This we can bring about by controlling our choices of λ_n .

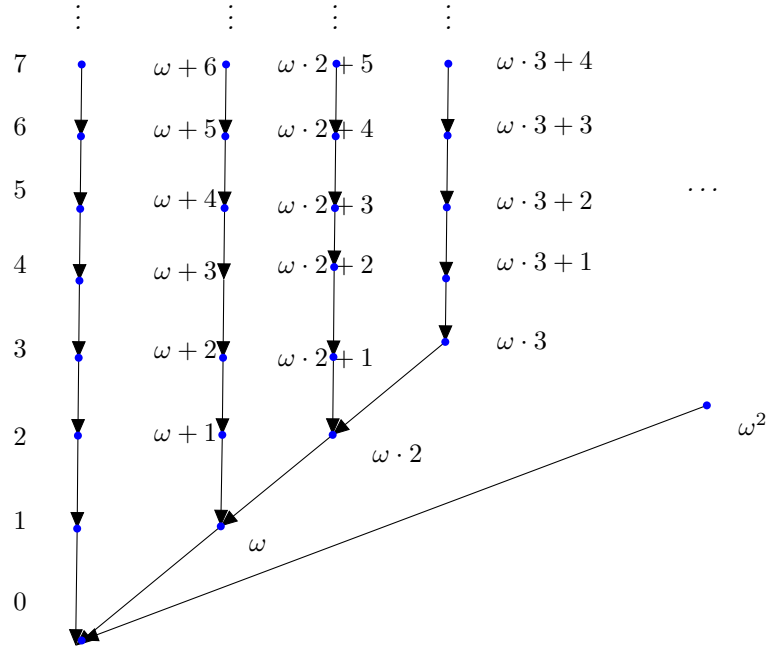
The construction of the f_α s ensures that $S(\alpha, \beta)$ holds if $\beta = \alpha + 1$ or if β is limit and $\alpha = \beta_0$. To be sure of $S(\alpha, \beta)$ when $\alpha < \beta$ are members of a fundamental sequence we need to specify that they are related by the transitive closure of the union of these two relations. A family of fundamental sequences satisfying this condition is **Schmidt-coherent**.

Formally:

DEFINITION 28 Let $\mathcal{F} : \Delta \rightarrow \Delta^\omega$ be an assignment of fundamental sequences to an initial segment Δ of the second number class. Define the **step-down** function $f : \Delta \rightarrow \Delta$ by

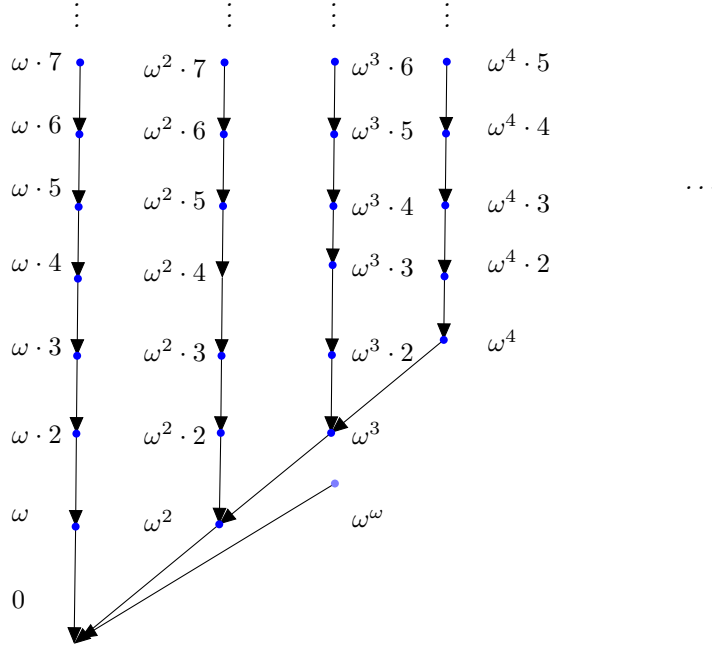
$$\text{if } \beta = \alpha + 1 \text{ then } \alpha \text{ else } \mathcal{F}\beta \ 0$$

and $f(0)$ is of course undefined.



We can think of f as a digraph, in which all paths lead to 0. If we do this then we can see that it is actually a tree, and a tree with no infinite descending paths. Any digraph like that is of course also the graph of a transitive relation, and if we reverse the arrows we obtain the Hasse diagram of a wellfounded (strict) partial order.

If we zoom out a bit and show only the branches consisting entirely of limit ordinals we get



[Is every wellfounded partial order on a set an intersection of two wellorderings of that set?]

This order is written $<_{\mathcal{F}}$ by Schmidt [55]—who calls it the step-down relation of \mathcal{F} .

Then

\mathcal{F} is **Schmidt-coherent** iff

$$(\forall \lambda \in \Delta)(\lambda \text{ limit} \rightarrow (\forall n \in \mathbb{N})((\mathcal{F} \restriction \lambda \cdot n) <_{\mathcal{F}} (\mathcal{F} \restriction \lambda \cdot (n+1)))).$$

Equivalently: every fundamental sequence lies entirely within one branch of the tree.

Here we really need some pictures. First picture is $<_{\mathcal{F}}$ restricted to ordinals below ω . The ordinals in this picture are a fundamental sequence for the first ordinal not so far seen, which is ω , so we put ω on the end of a new sprout coming off 0, to its right. We can now put in the ordinals below $\omega \cdot 2$, as in the second picture. We add infinite paths corresponding to fundamental sequences for $\omega \cdot n$ for all finite n . Third picture: ω^2 ; fourth picture: ω^ω ; fourth picture: ω^{ω^ω} ; fifth picture: ϵ_0

Is there an identifiable wellfounded stree such that a family of fundamental sequences is just a decoration of this tree? branches thru' the tree are limit

ordinals. This give a topology on the limit ordinals. Is this the same as the order topology?

EXERCISE 105 In exercise 104 you defined the natural assignment of fundamental sequences to ordinals below ϵ_0 : check that it is Schmidt-coherent.

Now define a natural assignment of fundamental sequences to the ordinals below Γ_0 , and check that that, too, is Schmidt-coherent.

We are now in a position to prove

THEOREM 30 (Schmidt, [55] theorem 1)

Consider the conditions:

- (a) F_0 is strictly monotonic;
- (b) if F_α is strictly monotonic so is $F_{\alpha+1}$, and $F_\alpha(0) \leq F_{\alpha+1}(0)$, and $F_\alpha(x) \leq F_{\alpha+1}(x)$;
- (c) $F_\lambda(n) = F_{\lambda_n}(n)$ when λ is limit.

If the system \mathcal{F} defined on the initial segment Δ satisfies conditions (a), (b) and (c) and is Schmidt-coherent then, for each $\alpha \in \Delta$,

- (i) F_α is strictly monotonic, and
- (ii) if α is a limit ordinal then

$$F_{\alpha_n}(0) < F_{\alpha_{n+1}}(0)$$

and

$$F_{\alpha_n}(x) \leq F_{\alpha_{n+1}}(x).$$

Proof:

We show by transfinite induction on α that, for each $\alpha \in \Delta$,

- (i) holds and
 - (iii) $\beta <_{\mathcal{F}} \alpha \rightarrow F_\beta(0) \leq F_\alpha(0)$; $\beta <_{\mathcal{F}} \alpha \rightarrow F_\beta(x) < F_\alpha(x)$
- [(ii) follows from (iii) because \mathcal{F} is Schmidt-coherent.]

The case $\alpha = 0$ is easy.

Suppose $\alpha = \gamma + 1$: By induction hypothesis, (i) holds for γ ; hence, by (b), it also holds for α .

By (b), (iii) holds if $\beta = \gamma$; but $\beta <_{\mathcal{F}} \alpha$ iff $\beta = \gamma \vee \beta \leq_{\mathcal{F}} \gamma$; hence, by (iii) of the induction hypothesis, (iii) holds for all $\beta <_{\mathcal{F}} \alpha$.

α a limit ordinal:

For each $x \in \mathbb{N}$ $F_\alpha(x) = F_{\alpha_x}(x) \leq F_{\alpha_{x+1}}(x)$ by (iii) of the induction hypothesis

$$< F_{\alpha_{x+1}}(x+1) \text{ by (i) of the induction hypothesis} \\ = F_\alpha(x+1).$$

Hence (i) holds. Moreover, if $0 < x < \omega$, $F_{\alpha_0}(x) < F_{\alpha_x}(x) = F_\alpha(x)$, by (iii) of the induction hypothesis, since \mathcal{F} is Schmidt-coherent; but $\beta <_{\mathcal{F}} \alpha \longleftrightarrow \beta =$

$\alpha_0 \vee \beta <_{\mathcal{F}} \alpha_0$; hence—by (iii) of the induction hypothesis— $\beta <_{\mathcal{F}} \alpha \rightarrow F_{\beta}(x) < F_{\alpha}(x)$.

Also, $\beta <_{\mathcal{F}} \alpha \rightarrow \beta <_{\mathcal{F}} \alpha_0 \vee \beta = \alpha_0$ which implies $F_{\beta}(0) \leq F_{\alpha_0}(0) = F_{\alpha}(0)$, by (iii) of the induction hypothesis.

Thus (iii) holds for a. ■

The following is from [55], but the proof is due to Nathan Bowler.

THEOREM 31 *For every proper initial segment Δ of the second number class there is a [are uncountably many, in fact] Schmidt-coherent system of fundamental sequences for the limit ordinals in Δ .*

Proof:

Let f be a bijection $\mathbb{N} \longleftrightarrow \{\beta : \beta < \alpha\}$ for some countable ordinal α , satisfying $f(0) = 0$. Suppose that $f(k)$ is a limit ordinal. We define a sequence $\langle s_n^k : n \in \mathbb{N} \rangle$ as follows

- s_0^k is that element of $\{i < k : f(i) < f(k)\}$ on which the value of f is maximal.
- s_{n+1}^k is the minimal element of $\{i \in \mathbb{N} : f(s_n^k) < f(i) < f(k)\}$.

It follows that

- (a) $s_0^k < k$;
- (b) For any i with $f(s_0^k) < f(i) < f(k)$, we have $i > k$;
- (c) For any $n \in \mathbb{N}$ and any i with $f(s_n^k) < f(i) < f(k)$, we have $i > s_n^k$;
- (d) The sequence $\langle f(s_n^k) : n \in \mathbb{N} \rangle$ is strictly increasing with limit $f(k)$.

(d) says that $\langle f(s_n^k) : n \in \mathbb{N} \rangle$ is a fundamental sequence for $f(k)$. We take these sequences as the elements of our system of fundamental sequences for the limit ordinals below α . Let σ be the corresponding step-down function, and define $\delta : \mathbb{N} \rightarrow \mathbb{N}$ so that $\sigma \cdot f = f \cdot \delta$. Thus when $k \in \mathbb{N}$ is such that $f(k)$ is limit we have $\delta(k) = s_0^k$. We must show that, for any limit ordinal $\beta < \alpha$ and any γ in the fundamental sequence for β , the sequence $\langle \sigma^n(\gamma) : n \in \mathbb{N} \rangle$ run through all lower members of that fundamental sequence. To establish this, it will suffice to prove the following

LEMMA 4 *Let k be such that $f(k)$ is a limit ordinal, let $n \in \mathbb{N}$ and $i \in \mathbb{N}$ be such that $f(s_n^k) < f(i) < f(k)$. Then $\delta(s_n^k) \leq \sigma(f(i))$.*

Proof:

This is immediate if $f(i)$ is successor, so suppose it is limit. So $\sigma(f(i)) = f(\delta(i)) = f(s_0^i)$. By (c) above we have $s_n^k < i$, and by assumption we have $f(s_n^k) < f(i)$, so by definition of s_0^i we have $f(s_n^k) \leq f(s_0^i) = \sigma(f(i))$. ■

REMARK 24 *There is no definable family of fundamental sequences for all $\alpha < \omega_1$.*

Proof: Suppose \mathcal{F} were such a family. We then define by recursion on ω_1 a sequence $\langle W_\alpha : \alpha < \omega_1 \rangle$ of wellorderings of \mathbb{N} (so each is a subset of $\mathbb{N} \times \mathbb{N}$). 0 is easy, successor steps are easy; at a limit λ use the fundamental sequence $\mathcal{F}\lambda$, to get the codes $W_{\mathcal{F}\lambda n}$ you have already formed for each $\mathcal{F}\lambda n$ and then piece them all together one after the other to get a wellordering of $\mathbb{N} \times \mathbb{N}$. Use a bijection $\mathbb{N} \times \mathbb{N} \longleftrightarrow \mathbb{N}$ to turn this into a code for $\Sigma_{n \in \mathbb{N}} \mathcal{F}\lambda n$ —which may have overshoot the mark, so take the right initial segment and you have a code for λ . (The sum of a sequence of ordinals might be bigger than its sup). None of this uses any AC.

This shows that if we have a function assigning a fundamental sequence to every countable ordinal, then we have a function assigning to each countable ordinal a wellordering of $\mathbb{N} \times \mathbb{N}$. But any wellordering of $\mathbb{N} \times \mathbb{N}$ is coded by a real number so this implies $\aleph_1 \leq 2^{\aleph_0}$. It is known that this is independent of ZF. ■

(I think that when (in [30]) Hardy introduced the Hardy Hierarchy—of which more later—he was trying to solve the continuum problem)

Suppose there is a function $g : \mathbb{N} \rightarrow \mathbb{N}$ that dominates all f_α . Then, for each $n \in \mathbb{N}$, let $h(n)$ be the sup of the α s such that g has permanently overtaken f_α by stage n . h is clearly nondecreasing. For every α there is $n \in \mathbb{N}$ s.t. $h(n) \geq \alpha$, so h is unbounded below ω_1 , and is an ω -sequence of countable ordinals whose sup is ω_1 , contradicting countable choice.

This shows that if we assume countable choice (or merely that ω_1 is regular) then there cannot be a Schmidt-coherent system of fundamental sequences for the whole of the second number class.

Rose says that theorem ?? is best possible, and credits [3] I'm sceptical about this because he also says that Schmidt, too, proves that it is best possible—and she doesn't!

If it really is best possible, it's presumably because a Schmidt-coherent family for all countable ordinals would give us an embedding of ω_1 into the reals, or something like that. There can be long sequences ($\geq \omega_1$) of functions with each function dominating all earlier functions, but they don't increase as fast as Wainer-Buchholtz.

11.4 Preposterously Large Countable Ordinals

Notes of Countable Ordinals Reading Group meeting on 16/v/2014

(look also at taranovsky's ordinalnotations.ps in my assorted-paper-archive folder)

Under the guidance of Jeroen van der Meeren and Michael Rathjen I finally began to get the first glimmers of an understanding of the use of a large ordinal in describing initial segments of the countable ordinals. What follows is my notes of the discussion of this topic at the meeting of the ordinals reading group on 16/v. Present were: your humble correspondent, Professors Leader and Dawar,

Arno Pauly, Philipp Kleppmann and an unidentified Ph.D. student from the Lab. We put our heads together and made some progress, and this file records my understanding of that progress.

Key word lurking in the background is **impredicativity**.

The following gadgetry goes back to Bachmann. [need a ref]

It's probably a good idea for the reader to start off by keeping in mind the Veblen picture of rows and rows of ordinals. The top row consists of powers of ω , written in increasing order left-to-right. Going down the page, each subsequent successor row consists of the fixed points in the enumeration of the row immediately above it; at limit stages the row is the intersection of all the rows above it. We assume that the reader is familiar with this picture.

For ordinals α and ζ we define a set $C(\alpha, \zeta)$ of ordinals and a function $\vartheta : On \rightarrow On$, by a *simultaneous* recursion on On^2 . The thing we are really interested in is the function ϑ ; the $C(\alpha, \zeta)$ are mere scaffolding, and they play no rôle in the system of notations with which the ϑ gadgetry will eventually furnish us.

To construct $C(\alpha, \zeta)$ you start with a set containing 0 and Ω , all the ordinals less than ζ , and $\vartheta(\gamma)$ for all $\gamma < \alpha$; you then close under $+$ and $\alpha \mapsto \omega^\alpha$. Our first stab at the definition of $\vartheta(\alpha)$ is: the least ζ such that $\zeta \notin C(\alpha, \zeta)$. Bear in mind that $\vartheta(\alpha)$ is **not** defined as the least thing not in $C(\alpha, \zeta)$. For one thing, it would need *two* arguments— $\vartheta(\alpha, \zeta)$ —not one. It's a complex diagonalisation and you need to read the definition carefully. Bind the ' ζ ' somehow, and “the least ζ such that $\zeta \notin C(\alpha, \zeta)$ ” sounds sensible. **However** we add a clause so that $\vartheta(\alpha)$ is not the first ζ s.t. $\zeta \notin C(\alpha, \zeta)$ but rather the first ζ s.t. $\zeta \notin C(\alpha, \zeta) \wedge \alpha \in C(\alpha, \zeta)$. It will become clear later what purpose is served by this extra $\alpha \in C(\alpha, \zeta)$ clause, but you should not expect it to be clear at this stage.

Here is something that threw me and it might throw you. It's pretty clear that the function $\alpha, \zeta \mapsto C(\alpha, \zeta)$ is \subseteq -increasing in both arguments, but you mustn't jump to the conclusion that ϑ is strictly increasing—it isn't, as we shall see. The best way to understand what is going on is to fix a small α and consider $C(\alpha, 0)$, $C(\alpha, 1)$ and so on, so let's do some of these by hand to calm our nerves. We will see that the first few values of ϑ are the first few ϵ -numbers.

$C(0, 0)$ contains 0 and Ω . We don't have to put any values of ϑ into it co's the first argument is 0. We then close under addition and $\beta \mapsto \omega^\beta$. Pretty clearly it is going to contain everything less than ϵ_0 . It *won't* contain ϵ_0 itself (how could it, after all?) but it does contain a lot of stuff beyond Ω . We will see later [*much* later] what that stuff does. For the moment it does nothing.

What about $C(0, 1)$? It's just going to be the same set. $C(0, \omega)$ is going to be the same set, too. Observe that if $\zeta < \epsilon_0$ then $\zeta \in C(0, \zeta)$, so *all* the $C(0, \zeta)$ are going to be the same set all the way through all the ordinals less than ϵ_0 . Indeed even $C(0, \epsilon_0)$ is the same (tho' $C(0, \epsilon_0 + 1)$ is bigger).

The first ζ such that $\zeta \notin C(0, \zeta)$ is therefore ϵ_0 . The second (\otimes) condition on candidates for $\vartheta(\alpha)$ (the condition that requires that $\alpha \in C(\alpha, \zeta)$) is satisfied—all it requires in this case is that $0 \in C(0, 0)$ —so we conclude that $\vartheta(0)$ is ϵ_0 .

Notice that there is never any need for us to compute $C(0, \zeta)$ for any $\zeta > \vartheta(0)$; since the only purpose served by the $C(\alpha, \zeta)$ is to enable us to calculate $\vartheta(\alpha)$, once that is done we lose interest.

How about $C(1, 0)$? It's like $C(0, 0)$ except that we put $\vartheta(0)$ (which is ϵ_0) into it before closing under the operations. This means that we get everything less than ϵ_1 (think: Cantor Normal Forms for ordinals $< \epsilon_1$). As we run through the $\zeta < \epsilon_1$ we get nothing new in $C(1, \zeta)$ until we reach ϵ_1 itself, so we conclude that $\vartheta(1) = \epsilon_1$. As before, the (\otimes) condition on ζ does nothing because all it requires is that $C(1, \zeta)$ should contain 1, and we already know it contains everything below ϵ_1 .

Similarly we conclude that $\vartheta(n) = \epsilon_n$ for $n < \omega$. A picture emerges in which, for small arguments, ϑ enumerates the ϵ numbers. In fact Jeroen tells me that ϑ is injective and all its values are ϵ -numbers.

Fixed point ϵ numbers are sometimes called κ -numbers, so that κ_0 is the least solution to $\kappa = \epsilon_\kappa$. Let us think a bit about what $\vartheta(\kappa_0)$ might be. We start with $C(\kappa_0, 0)$. This set contains Ω and all the ϵ -numbers below κ_0 , and is closed under $+$ and $\zeta \mapsto \omega^\zeta$. Now, recalling what we know about Cantor Normal Forms, we can see that this act of closure will put into $C(\kappa_0, 0)$ every ordinal below κ_0 (plus a lot of big rubbish beyond Ω). This immediately tells us that the sets $C(\kappa_0, \zeta)$ for $\zeta \leq \kappa_0$ are all going to be the same set as $C(\kappa_0, 0)$. We observe that $\kappa_0 \notin C(\kappa_0, \kappa_0)$ so we might expect that we then declare $\vartheta(\kappa_0)$ to be κ_0 . However note that κ_0 is not only the *second* argument at this stage, but also the *first*, so we look at the \otimes condition—“ $\alpha \in C(\alpha, \zeta)$ ”—and we see that it is not satisfied! So we have to look at a few more $C(\kappa_0, \zeta)$ before we can say we have reached $\vartheta(\kappa_0)$. In fact we have to go as far as $C(\kappa_0, \epsilon_{\kappa_0+1})$.

The picture I now have is that, for $\alpha < \Omega$, ϑ enumerates the ϵ -numbers less than Ω —except that it misses out the fixed points (that is what the \otimes condition is doing). Another way of putting this is that it enumerates those ordinals in the first row that do not appear in the second row; yet another way of putting it is to say that the purpose of the \otimes clause is to prevent ϑ from having fixed points.

That was what one might call the *first pass*. I am assured by Jeroen that $\vartheta(\Omega)$ is the first fixed-point ϵ -number (the first κ -number)—aka $\phi(2, 0)$ —and that $\vartheta(\Omega + 1)$ is the second fixed-point ϵ -number.

OK, so: thus emboldened, let us check these allegation for ourselves and start by thinking about what $\vartheta(\Omega)$ might be. We obtain $C(\Omega, \zeta)$ by starting with $\{\vartheta(\alpha) : \alpha < \Omega\}$ and all the ordinals less than ζ and closing under $\beta \mapsto \omega^\beta$ and $+$. If I was right earlier, then we have all the ϵ numbers less than the first κ number. So $C(\Omega, \kappa_0)$ contains Ω but does not contain κ_0 so $\vartheta(\Omega)$ is going to be κ_0 as foretold. Observe that we have now reached a stage where all the stuff $\geq \Omega$ that we always put into the $C(\alpha, \zeta)$ s starts doing something.

This is consonant with what the preceding paragraph is telling us, namely us that, in the *second pass*, ϑ goes back and enumerates those ordinals in the

second row that do not appear in the *third* row. Indeed one has the impression that in the α th pass ϑ enumerates in increasing order those ordinals in the α th row of the Veblen table that do not appear in the $\alpha + 1$ th row. Jeroen and Michael tell me that $\vartheta(\Omega^2) = \Gamma_0$. This would appear to confirm what i have just been saying, because, after all, once one has made Ω passes (and thereby reached $\vartheta(\Omega^2)$) one should have hit every power of ω below Γ_0 .

Stuff to sort out

There now follow some observations from Jeroen and Michael that I am reassured to find plausible but which I can't at this stage actually prove.

Jeroen also sez $\alpha < \Omega \rightarrow \alpha < \vartheta(\alpha)$.

All values of ϑ are less than Ω .

ϑ is injective.

[These last two observations cannot both be true! What did he mean?]

The values of ϑ do not depend on the choice of Ω . You can even take Ω to be ω_1^{CK} .

Every ϵ -number below the Bachmann-Howard ordinal is a value of ϑ .

$C(\epsilon_{\Omega+1}, 0) \mid \Omega$ is the ordinals below the Bachmann-Howard ordinal.

If $\alpha < \epsilon_{\Omega+1}$ then α has a CNF with base Ω . That much is obvious. Let $K(\alpha)$ be the set of ordinals that appear in the CNF for α , and let $\alpha^* = \max(K(\alpha))$. Then we can say

$$\begin{aligned} \vartheta(\alpha) < \vartheta(\beta) \text{ iff either } & \alpha < \beta \wedge \alpha^* < \vartheta(\beta) \\ \text{or} & \alpha > \beta \wedge \vartheta(\alpha) \leq \beta^* \end{aligned}$$

Then

$$\vartheta(\alpha) = \min\{\zeta \in E : \alpha^* < \zeta \wedge (\forall \beta < \alpha)(\beta^* < \zeta \rightarrow \vartheta(\beta) < \zeta)\}$$

where $\zeta \in E$ means that ζ is an ϵ -number.

All this machinery presumably supports a notational system. There is a binary $\phi(-, -)$ function that we can use to denote ordinals in sufficiently early levels of the Veblen table. I would like to understand that properly.

Should say something about why all these ordinals described by this Bachmann gadgetry are recursive. Anuj says that the ordering on the ordinals denoted by these notations is decidable. So, for any of these ordinals— α , say—the set of [gnumbers of] notations for ordinals below α gives a wellordering of \mathbb{N} . [but why is this set of notations for ordinals below α a decidable set? Why isn't it merely r.e...?]

Apparently it's straightforward to show that $C(\Omega, \beta)$ never exhausts all the ordinals, so that $\vartheta(\alpha)$ is well-defined. Should find something to say about this.

11.4.1 Cantor normal form using $\omega \uparrow\uparrow \alpha$

One obvious generalisation of CNF replaces the base ω by a different base. We exploited earlier the fact that ordinals above ϵ_0 but below ϵ_1 can be notated by a CNF with base ϵ_0 . How else can we generalise?

Something that has always puzzled me is why the discovery that Cantor Normal form sometimes gives uninformative answers (think: ϵ_0) did not prompt the reflection that one should use the normal function after exponentiation as the gadget for a system of ordinal notations. After all, CNF uses exponentiation to base ω as a normal function the drives a “division algorithm”, so why not just use the next normal function in the Doner-Tarski hierarchy? Let’s try this and see what happens; perhaps we shall learn from this exercise why Veblen and co^y escalated the struggle to notate ordinals by using this new gadget of enumerating fixed points rather than do what seems the obvious thing.

The next Doner-Tarski operation beyond exponentiation is declared by

$$\begin{aligned}\beta \uparrow\uparrow 0 &= \beta; \\ \beta \uparrow\uparrow (\alpha + 1) &= (\beta \uparrow\uparrow \alpha)^\beta; \\ &\text{taking sups at limits.}\end{aligned}$$

Thus $x \uparrow\uparrow 1 = x^x$; $x \uparrow\uparrow 2 = (x^x)^x = x^{x^2}$; $x \uparrow\uparrow 3 = (x^{x^2})^x = x^{x^3}$; and presumably $x \uparrow\uparrow n = x^{x^n}$ for $n \in \mathbb{N}$.

And, when $\beta = \omega$.

$$\begin{aligned}\omega \uparrow\uparrow 0 &= \omega; \\ \omega \uparrow\uparrow (\alpha + 1) &= (\omega \uparrow\uparrow \alpha)^\omega; \\ &\text{taking sups at limits.}\end{aligned}$$

It’s worth noting that if you get it the other way round, so that the successor step is

$$\omega \uparrow\uparrow (\alpha + 1) = \omega^{(\omega \uparrow\uparrow \alpha)}$$

—which looks more natural—you find that $\omega \uparrow\uparrow \omega = \epsilon_0$ and $\omega \uparrow\uparrow (\omega + 1) = \omega^{\omega \uparrow\uparrow \omega} = \omega^{\epsilon_0} = \epsilon_0$ so $\beta \mapsto \omega \uparrow\uparrow \beta$ grinds to a shuddering halt, and is not strictly increasing, let alone normal.

Then when we do the CNF thing we get... Give me an ordinal α . Let β_0 be maximal such that

$$\omega \uparrow\uparrow \beta_0 \leq \alpha < \omega \uparrow\uparrow (\beta_0 + 1) = (\omega \uparrow\uparrow \beta_0)^\omega.$$

Now let n_0 be maximal such that...

$$(\omega \uparrow\uparrow \beta_0)^{n_0} \leq \alpha < (\omega \uparrow\uparrow \beta_0)^{n_0+1} = (\omega \uparrow\uparrow \beta_0)^{n_0} \cdot (\omega \uparrow\uparrow \beta_0)$$

Now let β_1 be maximal such that...

$$(\omega \uparrow\uparrow \beta_0)^{n_0} \cdot (\omega \uparrow\uparrow \beta_1) \leq \alpha < (\omega \uparrow\uparrow \beta_0)^{n_0} \cdot (\omega \uparrow\uparrow (\beta_1 + 1)) = (\omega \uparrow\uparrow \beta_0)^{n_0} \cdot (\omega \uparrow\uparrow \beta_1)^\omega$$

Then we find n_1 s.t.

$$(\omega \uparrow\uparrow \beta_0)^{n_0} \cdot (\omega \uparrow\uparrow \beta_1)^{n_1} \leq \alpha < (\omega \uparrow\uparrow \beta_0)^{n_0} \cdot (\omega \uparrow\uparrow (\beta_1+1)) = (\omega \uparrow\uparrow \beta_0)^{n_0} \cdot (\omega \uparrow\uparrow \beta)^{n_1+1}$$

The next question is: why do people not do this analysis? What is the least fixed point $\alpha = \omega \uparrow\uparrow \alpha$?

$$\omega \uparrow\uparrow 1 = \omega^\omega$$

$$\omega \uparrow\uparrow 2 = (\omega \uparrow\uparrow 1)^\omega = (\omega^\omega)^\omega = \omega^{\omega^2}$$

$$\omega \uparrow\uparrow 3 = (\omega \uparrow\uparrow 2)^\omega = (\omega^{\omega^2})^\omega = \omega^{\omega^3}$$

So presumably

$$\omega \uparrow\uparrow \omega = \omega^{\omega^\omega}$$

and

$$\omega \uparrow\uparrow (\omega + 1) = (\omega^{\omega^\omega})^\omega = \omega^{\omega^\omega + 1}$$

so it's looking as if the least fixed point $\alpha = \omega \uparrow\uparrow \alpha$ is ϵ_0 . If that's the case then that might help to explain why Veblen and co^y went straight to the device of enumerating fixed points. What rather bothers me is that the literature nowhere seems to explain why the tradition took the step it did. If the reason why it moved straight to Veblen ϕ s is that using $\uparrow\uparrow$, $\uparrow\uparrow\uparrow$ and do on does nothing for us, then why was this never spelled out?

Connect this with Paris-Harrington and other connections between typing disciplines and strength.

What was Zachiri saying about Paris-Harrington?

Ackermann a fixed point for DT..?

Want to show that every function in the DT hierarchy is normal in its second argument

<http://www.math.ucsb.edu/~doner/articles/>.

Chapter 12

Constructive Mathematics

The classical concept of *nonempty set* multifurcates into lots of constructively distinct properties. Constructively x is **nonempty** if $\neg(\forall y)(y \notin x)$; x is **inhabited** if $(\exists y)(y \in x)$, and these two properties are distinct constructively: the implication $\neg\forall\phi \rightarrow \exists\neg\phi$ is not good in general.

A is **decidable** iff $(\forall x)(x \in A \vee x \notin A)$. $A \subseteq B$ is a **detachable** subset of B iff $(\forall x \in B)(x \in A \vee x \notin A)$.

Linton-Johnstone. Negative interpretation. Heyting Arithmetic.

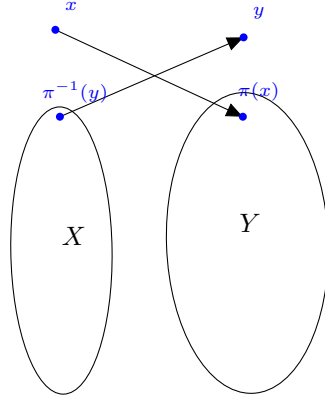
Every Kfinite set is notnot Nfinite

Every subset of an Nfinite set is Nfinite

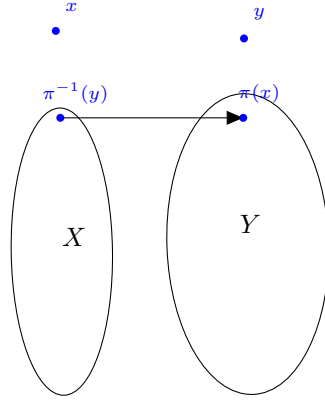
Heyting naturals the cardinals of N-finite sets. Decidable¹ whether or not two Nfinite sets are in bijection. We prove by induction on X that $(\forall Y)(\text{Nfinite}(Y) \rightarrow (X \sim Y \vee \neg(X \sim Y)))$. Clearly true for X empty. Sse true for X , and consider $X \cup \{x\}$, with $x \notin X$. If Y is Nfinite then its either \emptyset in which case the answer is ‘no’ or it’s $Y' \cup \{y\}$ with $y \notin Y'$. By induction hyp on X we have $X \sim Y' \vee \neg(X \sim Y')$. If the first then $X \sim Y$. If $\neg(X \sim Y')$ then we can’t have $X \cup \{x\} \sim Y' \cup \{y\}$. This is because any bijection $\pi : X \cup \{x\} \longleftrightarrow Y' \cup \{y\}$ will give rise to a bijection $X \sim Y'$, namely $\pi \setminus \{\langle \pi^{-1}(y), y \rangle, \langle x, \pi(x) \rangle\} \cup \{\langle \pi^{-1}(y), \pi(x) \rangle\}$.

π (before)

¹Different meaning of this word here!!



$$\pi \setminus \{ \langle \pi^{-1}(y), y \rangle, \langle x, \pi(x) \rangle \} \cup \{ \langle \pi^{-1}(y), \pi(x) \rangle \} \text{ (after)}$$



Then we prove trichotomy for Nfinite cardinals: given any two Nfinite sets one injects into the other.

LEMMA 5 $(\forall X)(Nfin(X) \rightarrow (\forall Y)(Nfin(Y) \rightarrow X \hookrightarrow Y \vee Y \hookrightarrow X))$

Proof:

By induction on X . Base case (X empty) is easy.

Suppose true for X . Want it to be true for $X \cup \{x\}$. Let Y be Nfinite. By induction hypothesis either $Y \hookrightarrow X$ (in which case $Y \hookrightarrow X \cup \{x\}$) or $X \hookrightarrow Y$. Y is non empty so it is $Y' \cup \{y\}$. By induction hypothesis either $Y' \text{ inj } X$ (in which case Y (which is $Y' \cup \{y\}$) $\hookrightarrow X \cup \{x\}$). On the other horn $X \hookrightarrow Y'$ which gives $X \cup \{x\} \hookrightarrow Y' \cup \{y\} = Y$. (NB for this to work we need both X and Y to be Nfinite not merely Kfinite). ■

Then we prove

LEMMA 6 *Every nonempty Kfinite set of naturals has a least member.*

Proof:

Suppose $X \cup \{n\}$ is a set of naturals with X kfinite. By induction hypothesis X has a least member, x , say. But then we have $n \leq m \vee m \leq m$ and either way we have a least element of $X \cup \{n\}$. ■

Then

LEMMA 7 *Every natural number has a prime factor.*

The set of nontrivial (> 1) factors of a natural number n is an Nfinite set, being a subset of the numbers below n . It has a least element. That least element is a prime factor of n . ■

LEMMA 8 (Euclid) $(\forall X)(X \text{ an Nfinite set of primes} \rightarrow (\exists)(p \text{ a prime not in } X))$.

Proof: If X is an Nfinite set of primes then $\prod X + 1$ is Nfinite and has prime factors, by lemma 7. All of them lie outside X , for the usual reasons. Therefore there are primes not in X . ■

Notice that this strategy doesn't work to show that there are infinitely many primes $\equiv 3 \pmod{4}$, because the only thing the classical argument tells us in that case is that not all the prime factors of $\prod X + 1$ can be in X .

I do not know if the infinitude of the set of primes congruent to 3 mod 4 has a constructive proof.

12.1 Diaconescu: the Axiom of Choice implies Excluded Middle

Clearly if every family of nonempty sets is to have a choice function then if x is nonempty we can find something in it. This would imply that every nonempty set is inhabited. We shall not resort to such cheating. If we are to refrain from cheating we will have to adopt AC in the form that every set of *inhabited* sets has a choice function.

Let us assume AC in this form, and deduce excluded middle. Let p be an arbitrary expression; we will deduce $p \vee \neg p$. Consider the set $\{0, 1\}$, and the equivalence relation \sim defined by $x \sim y$ iff $x = y \vee p$. Next consider the quotient $\{0, 1\} / \sim$. (The suspicious might wish to be told that this set is $\{x : (\exists y)((y = 0 \vee y = 1) \wedge (\forall z)(z \in x \longleftrightarrow z \sim y))\}$). This is an inhabited set of inhabited sets. Its members are the equivalence classes $[0]$ and $[1]$ —which admittedly may or may not be the same thing—but they are at any rate inhabited. Since the quotient is an inhabited set of inhabited sets, it has a selection function f . We know that $[0] \subseteq \{0, 1\}$ so certainly $(\forall x)(x \in [0] \rightarrow x = 0 \vee x = 1)$. Analogously

we know that $[1] \subseteq \{0, 1\}$ so certainly $(\forall x)(x \in [1] \rightarrow x = 0 \vee x = 1)$. So certainly $f([0]) = 0 \vee f([0]) = 1$ and $f([1]) = 0 \vee f([1]) = 1$. This gives us four possible combinations. $f([0]) = 1$ and $f([1]) = 0$ both imply $1 \sim 0$ and therefore p . That takes care of three possibilities; the remaining possibility is $f([0]) = 0 \wedge f([1]) = 1$. Since f is a function this tells us that $[0] \neq [1]$ so in this case $\neg p$. So we conclude $p \vee \neg p$. ■

This proof is what Douglas Bridges and I reconstructed from memory, given only the fact that the result is correct. There is no reason to suppose that is the original proof due to Diaconescu, [18]—tho' it could be, for all i know.

12.1.1 Least Number Principle Implies Excluded Middle

The least number principle says that every inhabited set of naturals has a least member.

LEMMA 9 *The Least Number Principle Implies Excluded Middle*

Proof: Let p be any proposition, and consider $A = \{n \in \mathbb{N} : n = 1 \vee (n = 0 \wedge p)\}$.

A is inhabited (since 1 is a member of it) so, by LNP, it has a least member. Every member of A is 0 or 1. If this least member is 0 then we must have p . If it is 1 we must have $\neg p$. ■

Observe that A is not Kfinite. (We saw in lemma 6 that every Kfinite set of naturals has a least member).

“Fishy” Sets

The two proofs we have just seen involve ...

$$A = \{n \in \mathbb{N} : n = 1 \vee (n = 0 \wedge p)\}.$$

$x \sim y$ iff $x = y \vee p$. Classically \sim is either the identity relation or the universal relation—neither of them things involving ‘ p ’.

Classically we have two infinite distributive laws:

$$p \vee (\forall x)(A(x)) \text{ is equivalent to } (\forall x)(p \vee A(x))$$

and

$$p \wedge (\exists x)(A(x)) \text{ is equivalent to } (\exists x)(p \wedge A(x))$$

so we can “export” from the scope of a quantifier any subformula not containing any occurrence of a bound variable. This does not work constructively (constructively $p \vee (\forall x)(A(x))$ does not follow from $(\forall x)(p \vee A(x))$ —though the converse is good) and where we have failures of exportation we find these “fishy” sets that—as we have seen—turn up in proofs that certain set-theoretic principles imply excluded middle.

12.1.2 Linton-Johnstone and Markov's Principle

LEMMA 10 *Let X be Kuratowski-finite. Then*

$$(\forall y \in X)(\neg\neg\phi(y)) \rightarrow \neg\neg(\forall y \in X)(\phi(y)).$$

Proof:

Naturally we use induction on K-finite sets.

The proposition certainly holds when $X = \emptyset$. Now assume it holds for X , and assume also that $(\forall y \in X \cup \{x\})(\neg\neg\phi(y))$. This last assumption implies both

- (i): $(\forall y \in X)(\neg\neg\phi(y))$ and
- (ii) $(\forall y \in \{x\})(\neg\neg\phi(y))$,

and (ii) of course implies $\neg\neg\phi(x)$. By induction hypothesis (i) implies

$$(ii)': \neg\neg(\forall y \in X)(\phi(y)).$$

Now $(\forall y \in X)(\phi(y))$ and $\phi(x)$ together imply

$$(iii) (\forall y \in X \cup \{x\})(\phi(y))$$

so the conjunction of their double negations will imply the double negation of (iii), namely:

$$\neg\neg(\forall y \in X \cup \{x\})(\phi(y))$$

as desired.

Markov's principle is a kind of dual to Linton-Johnstone. If you attempt to prove the dual to L-J, namely

$$\neg\neg(\exists y \in X)(\phi(y)) \rightarrow (\exists y \in X)(\neg\neg\phi(y))$$

you will find that the proof breaks down because $\neg\neg$ does not distribute over \vee (tho' it does over \wedge). Markov's principle asserts that nevertheless in some circumstances $\neg\neg$ *does* distribute over \vee . Suppose ϕ is a decidable predicate of naturals, so that $(\forall n \in \mathbb{N})(\phi(n) \vee \neg\phi(n))$. Markov's principle then tells us that

$$\neg\neg(\exists n \in \mathbb{N})(\phi(n)) \rightarrow (\exists n \in \mathbb{N})(\phi(n))$$

12.2 Proof theory, Curry-Howard and Realizability

EXERCISE 106 (*)

Find a normal proof of $(p \longleftrightarrow (p \rightarrow q)) \rightarrow q$.

12.2.1 Proof Theory

Do we prove cut-elimination here? or keep it for chapter 11?

12.2.2 Curry-Howard

Read the relevant chapter of www.dpmms.cam.ac.uk/~tf/chchlectures.pdf here.

We can define other connectives in terms of \rightarrow if we are allowed infinitary intersections:

$A \vee B$

$$A \vee B \text{ is } \bigwedge_C ((A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C))$$

Evidently one can infer C from $A \vee B$, $A \rightarrow C$ and $B \rightarrow C$, as desired.

$$\frac{\frac{A \rightarrow C \quad (A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C)}{(B \rightarrow C) \rightarrow C} \rightarrow\text{-elim} \quad B \rightarrow C}{C} \rightarrow\text{-elim} \quad (12.1)$$

And one can infer $A \vee B$ from A :

$$\frac{\frac{A \quad [A \rightarrow C]^1}{C} \rightarrow\text{-elim} \quad \frac{[B \rightarrow C]^2}{C} \text{ identity rule}}{(B \rightarrow C) \rightarrow C} \rightarrow\text{-int (2)} \quad (12.2)$$

$$\frac{(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C)}{(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C)} \rightarrow\text{-int (1)}$$

... and from B :

$$\frac{\frac{B \quad [B \rightarrow C]^1}{C} \rightarrow\text{-elim} \quad (B \rightarrow C) \rightarrow C}{(B \rightarrow C) \rightarrow C} \rightarrow\text{-int (1)} \quad (12.3)$$

$$\frac{(B \rightarrow C) \rightarrow C \quad [A \rightarrow C]^2}{(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C)} \text{ identity rule}$$

$$\frac{(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C)}{(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow C)} \rightarrow\text{-int (2)}$$

$A \wedge B$

$$A \wedge B \text{ is } \dots \bigwedge_C ((A \rightarrow (B \rightarrow C)) \rightarrow C)$$

$\bigwedge_C ((A \rightarrow (B \rightarrow C)) \rightarrow C)$ can be deduced from the two assumptions A and B :

$$\frac{\frac{A \quad [A \rightarrow (B \rightarrow C)]^1}{B \rightarrow C} \rightarrow\text{-elim} \quad B}{C} \rightarrow\text{-elim} \quad (12.4)$$

$$\frac{C}{(A \rightarrow (B \rightarrow C)) \rightarrow C} \rightarrow\text{-int (1)}$$

For the other direction—inferring both A and B from $\bigwedge_C ((A \rightarrow (B \rightarrow C)) \rightarrow C)$ —observe that we are allowed to specialise ‘ C ’ to ‘ A ’ or to ‘ B ’, giving both

$$(A \rightarrow (B \rightarrow A)) \rightarrow A \quad \text{and} \quad (A \rightarrow (B \rightarrow B)) \rightarrow B.$$

Both these formulæ have antecedents that are valid, so we can infer both A and B .

I'm now a bit worried by this. What is to be gained by defining finite conjunction in terms of infinitary conjunction? And the $\bigwedge_C(\dots)$ really is a conjunction over *all* formulæ not just all formulæ of lower rank, or whatever.

EXERCISE 107 (*) (*Part III Computability and Logic Examination 2014*)

Find a natural deduction proof for

$$((((A \rightarrow B) \rightarrow A) \rightarrow A) \rightarrow B) \rightarrow B$$

using only the two rules for \rightarrow and the “identity rule”:

$$\frac{A \quad B}{A}$$

Then decorate your proof appropriately with λ -terms.

Sort out the duplication

12.2.3 The Axiom of Choice in Constructive Set Theory: notes consequent to a lecture given by Sol Feferman on 10/i/2014, written up partly to amuse Valeria Paiva

AC

... I am trying to explain to myself why the implication

$$(\forall x \in A)(\exists y)(\phi(x, y)) \rightarrow (\exists f)(\forall x \in A)(\phi(x, f(x))) \quad (\text{S})$$

is suspect. (Hence the ‘S’!)

A certificate for S is a function that takes certificates for $(\forall x \in A)(\exists y)(\phi(x, y))$ and gives back certificates for $(\exists f)(\forall x \in A)(\phi(x, f(x)))$.

Very well, so: what is a certificate for $(\forall x \in A)(\exists y)(\phi(x, y))$? Write it out in full as $(\forall x)(x \in A \rightarrow (\exists y)(\phi(x, y)))$.

A certificate for $(\forall x)(x \in A \rightarrow (\exists y)(\phi(x, y)))$ is a function that takes an x and returns a certificate for $x \in A \rightarrow (\exists y)(\phi(x, y))$.

A certificate for $x \in A \rightarrow (\exists y)(\phi(x, y))$ is a function that takes a certificate for $x \in A$ and gives back a certificate for $(\exists y)(\phi(x, y))$.

A certificate for $(\exists y)(\phi(x, y))$ is a pair-of-a- y -and-a-certificate-that- $\phi(x, y)$.

So, with the help of a bit of currying (which is constructive!) we can think of a certificate \mathcal{C} for the antecedent of S as a function that takes x and a certificate that $x \in A$ and returns a pair-of-a- y -and-a-certificate-that- $\phi(x, y)$. Actually we can think of the input to \mathcal{C} as a certificate that $x \in A$ *simpliciter*—rather than

the-certificate-plus- x —since presumably we can recover x from the certificate. But nothing hangs on it. What emphatically *does* matter that the y given by the certificate depends not just on x but on the certificate that $x \in A$.

Now! Can we turn such a certificate \mathcal{C} into a certificate for $(\exists f)(\forall x \in A)(\phi(x, f(x)))$? A certificate for $(\exists f)(\forall x \in A)(\phi(x, f(x)))$ must be a pair-of-a-function- f -with-a-certificate-that- $(\forall x \in A)(\phi(x, f(x)))$. You might think, Dear Reader, that the desired f is staring us in the face, being nothing more arcane than the certificate \mathcal{C} that we have in our hand. Sadly, life is not that simple. According to the consequent of S, inputs to this function f are members of A rather than certificates-that-things-are-in- A . Recall now the plot point: our ability (embodied in the certificate \mathcal{C}) to produce y s.t. $\phi(x, y)$ relied on more than just the identity of x , it relied on the certificate that $x \in A$.

This reminds me (for one) of the fact that $A \rightarrow (B \vee C) \rightarrow (A \rightarrow B) \vee (A \rightarrow C)$ is not constructive. [Better men than i have been fooled by this—PTJ for one, albeit only briefly!]

You might think that the problem will go away if we drop the restriction on the range of ‘ x ’, so we are considering ‘ $(\forall x)(\exists y)\phi(x, y)$ ’ and you’d be right. However this is less use than it might seem, since we forfeit generality. The point is that altho’ *classically* $(\forall x \in A)(\exists y)\phi(x, y)$ is equivalent to $(\forall x)(\exists y)(x \in A \rightarrow \phi(x, y))$, nevertheless the equivalence is not good *constructively*. This is not hard to see, for in the second case we have to conjure up a y on given a bare x , without any information about whether it is in A or not.

Myhill

So we don’t believe that S is constructively correct. Observe that if we make Myhill’s move, and require the existential quantifier in the antecedent to be a ‘ $\exists!$ ’ rather than a ‘ \exists ’ then the problem goes away. This is because the uniqueness of the y means that the dependence on the certificate for $x \in A$ no longer affects the outcome. [Really...?] Of course the ‘ $\exists!$ ’ has some nontrivial internal structure, so it would be quite instructive to write out this case with the same thoroughness we showed in our attack on S...it would be a *lot* longer. But that’s for later, if at all. For the moment the question is: *what about DC?*

DC

$$(\forall x \in A)(\exists y \in A)(\phi(x, y)) \rightarrow (\forall x \in A)(\exists f : \mathbb{N} \rightarrow A)(f(0) = x \wedge (\forall n \in \mathbb{N})(\phi(f(n), f(n+1)))) \quad \text{DC}$$

I did the discussion of S in some detail in the hope of being able to deal with DC with slightly more despatch. Certainly the antecedent of DC is very nearly the same as the antecedent of S so we are off to a flying start. The only difference is that the \mathcal{C} that we get from *this* antecedent also provides a certificate that $y \in A$.

What about certificates for the consequent? The key observation here is that the f whose existence is claimed depends on x , so when we are called upon to produce it we are also given a certificate that $x \in A$.

Okay, so suppose we have our certificate \mathcal{C} , modified from that in section 12.2.3. The challenge is to find a certificate for

$$(\forall x \in A)(\exists f : \mathbb{N} \rightarrow A)(f(0) = x \wedge (\forall n \in \mathbb{N})(\phi(f(n), f(n+1)) \wedge f(n) \in A))$$

A certificate for this will be a function that takes x and a certificate that $x \in A$ and returns a certificate that

$$(\exists f : \mathbb{N} \rightarrow A)(f(0) = x \wedge (\forall n \in \mathbb{N})(\phi(f(n), f(n+1)) \wedge f(n) \in A))$$

A certificate for this last is going to be a function $f : \mathbb{N} \rightarrow A$ together with a certificate that $f(0) = x \wedge (\forall n \in \mathbb{N})(\phi(f(n), f(n+1)) \wedge f(n) \in A)$. Can we cook up such a function given \mathcal{C} ? Yes: set $f(0)$ to be x ; for larger n we repeatedly apply \mathcal{C} . If we apply \mathcal{C} to the certificate for x we obtain $y \in A$ s.t. $\phi(x, y)$. We announce to the world that $f(1)$ is to be this y . However we also get a certificate-that- $y \in A$, and this is what we need to keep the recursion going, for we can now apply \mathcal{C} —this time to the certificate for y —to obtain $f(2)$...and beyond.

12.3 Recursive Analysis

Douglas Bridges sez: \mathbb{R} has only two subsets that are detachable, itself and the empty set.

See [9] pp 53ff.

Reals can arise as all sorts of things, from Dedekind cuts, or Cauchy sequences for example. But if we have the added dimension of computability to worry about then even if we have decided to think of computable reals as computable Cauchy sequences (in the rationals of course) we can wonder whether we think of those computable Cauchy sequences as functions-in-intension (programs) or as function graphs (functions in extension). Both make sense. If we do the first, then Rice's theorem will ensure that the equality relation between computable reals is undecidable.

Another thing we can do is say that a real is computable iff there is a Cauchy-sequence-in-intension whose limit it is. That way our computable reals aren't different things from reals, but delineate a subset \mathbb{R}_c of \mathbb{R} ; this is how Bridges does it.

Analysis is full of dependencies: If $f : \mathbb{R} \rightarrow \mathbb{R}$ is continuous then $(\forall x)(\forall \epsilon)(\exists \delta)(\dots)$ But how does the δ depend on x and ϵ ? Riemann's theorem: if f is integrable then $\forall \epsilon \exists \delta \dots$ In the realistic cases we deal with in ordinary² mathematics we can obtain values for δ from the arguments x and ϵ in fairly explicit ways that one would like to be allowed to describe as 'computable'. People in Analysis

²I know one shouldn't use the phrase 'ordinary mathematics' but sometimes temptation gets the better of one.

don't make much of these dependencies but occasionally you will see the ϵ s and δ s equipped with subscripts, as below:

Remember TWK's proof that:

if $a_n \rightarrow a$ and $b_n \rightarrow b$ then $a_n + b_n \rightarrow a + b$. (Q1 on Analysis 1 sheet 1):

$a_n \rightarrow a$ so $(\forall \epsilon > 0)(\exists N_a(\epsilon))(\forall n > N_a(\epsilon))(|a_n - a| < \epsilon) \dots =$

and

$b_n \rightarrow b$ so $(\forall \epsilon > 0)(\exists N_b(\epsilon))(\forall n > N_b(\epsilon))(|b_n - b| < \epsilon)$

Now set $N(\epsilon) := \max(\{N_a(\epsilon/2), N_b(\epsilon/2)\})$, and take it from there

\dots

$(\forall n > N(\epsilon))(|(a_n + b_n) - (a + b)| < \epsilon)$

Connect with skolemisation.

Sometimes people are tempted

Is it, in fact, OK to describe this process as 'computable'? There is an obvious *prima facie* problem in that the quantities x , ϵ and δ are *infinite precision objects*, so we cannot compute with them in the way we have been accustomed to so far. But that's not really a problem because we can always take these quantities to be rationals.

12.4 A constructive treatment of infinitesimals

see [6].

Chapter 13

Notes and Appendices

13.1 Chapter 3

13.1.1 Horn clauses in retype declarations

This illustration comes from Ben Millwood.

... we can declare a datatype C equipped with a constructor $\text{con} : (C \rightarrow \emptyset) \rightarrow C$. Now, by recursion, declare a function f defined on this datatype by:

$$f(\text{con}(g)) = g(\text{con}(g))$$

This is a legitimate (if degenerate) declaration of f by recursion.

Do some type-checking... g must be of type $C \rightarrow \emptyset$ since it is an argument to con ; so $\text{con}(g)$ is of type C , and $f(\text{con}(g))$ is of type \emptyset which is impossible, so f must be the empty function.

So C must be empty if we are allowed to define f .

Ben says:

It's worse than that: f is a legitimate function $C \rightarrow 0$, so then $\text{con}(f)$ is a legitimate element of C . So C can't be empty.

You can look at the declaration $\text{con} : (C \rightarrow 0) \rightarrow C$ as saying:

- if C is uninhabited, then there is a function $h : C \rightarrow 0$, but then $\text{con}(h) : C$, so C is inhabited,
- if C is inhabited, say by $x : C$, then C only has one constructor, so $x = \text{con}(h)$ for some $h : C \rightarrow 0$. But the existence of such an h proves that C is uninhabited.

So C has elements if and only if it doesn't—being empty doesn't resolve the paradox. Here's another example: let D be a datatype with one constructor, $\text{don} : \mathcal{P}(D) \rightarrow D$, where \mathcal{P} is the powerset. Then clearly D is inhabited, since $\text{don}(\emptyset)$ is an element of it. But don itself is (by definition) an injection from the power set of D to D , which Cantor says is impossible. This is a less

striking example than the previous one to my eye, but maybe a more familiar one. (Perhaps the resolution is simply that this signature gives a retype that is a proper class, but that's quite an awkward conclusion.)

The lesson that I take from this, at least, is that some constructor types are permissible and some are not. In particular, the retype C above isn't merely empty, it cannot exist at all.

He's right, but it's easy to see where the problem lies: it's the "negative occurrence" of the datatype in the declaration. It prevents the datatype declaration being Horn.

13.1.2 Infinitary Languages

While we are on the subject of infinitary languages let's have the following morsel:

THEOREM 32 *Scott's Isomorphism theorem*

Every countable structure can be characterised up to isomorphism by a single sentence of $\mathcal{L}_{\omega_1\omega}$.

Proof:

We can obviously do this by cheating: if we want to characterise \mathfrak{A} up to isomorphism by providing a name a for every element of A , the carrier set of \mathfrak{A} . However we want to do it without cheating!

(lifted from [33] who lifted it from [12]).

Let \mathfrak{A} be a countable structure for a language \mathcal{L} . We will show that there is a sentence ϕ of $\mathcal{L}_{\omega_1\omega}$ such that, for all countable structures \mathfrak{B} for \mathcal{L} , $\mathfrak{B} \models \phi$ iff $\mathfrak{B} \simeq \mathfrak{A}$.

It will be easier to understand the construction of ϕ if we bear in mind that it is intended to power a back-and-forth construction of an isomorphism $\mathfrak{B} \simeq \mathfrak{A}$.

For each tuple $a_1 \dots a_n$ from A , and every $\beta < \omega_1$, we define a formula $\phi_{a_1 \dots a_n}^\beta(x_1 \dots x_n)$ by recursion on β as follows:

$\beta = 0$:

$\phi_{a_1 \dots a_n}^0(x_1 \dots x_n)$ is

$$\bigwedge \{ \theta(x_1 \dots x_n) : \mathfrak{A} \models \theta(a_1 \dots a_n) \}$$

where θ is atomic or negatomic.

$\beta \neq \alpha^+$:

Naturally $\phi_{a_1 \dots a_n}^\beta(x_1 \dots x_n)$ is $\bigwedge_{\gamma < \beta} \phi_{a_1 \dots a_n}^\gamma(x_1 \dots x_n)$

$\beta = \alpha^+$:

This is where the work gets done. $\phi_{a_1 \dots a_n}^{\alpha+1}(x_1 \dots x_n)$ is the conjunction

$$\phi_{a_1 \dots a_n}^\alpha(x_1 \dots x_n) \quad \wedge$$

$$\bigwedge_{a_{n+1} \in A} (\exists x_{n+1}) (\phi_{a_1 \dots a_{n+1}}^\alpha(x_1 \dots x_{n+1})) \quad \wedge$$

$$(\forall x_{n+1}) \bigvee_{a_{n+1} \in A} (\phi_{a_1 \dots a_{n+1}}^\alpha(x_1 \dots x_{n+1})).$$

(Both these clauses look like infinitary $\forall\exists$)

Reality check:

for all tuples $a_1 \dots a_n$ and all $\beta < \omega_1$,

1. the formula $\phi_{a_1 \dots a_n}^\beta$ has at most the free variables ' x_1 ' ... ' x_n '; and
2. $\mathfrak{A} \models \phi_{a_1 \dots a_n}^\beta[a_1 \dots a_n]$;
3. $\mathfrak{A} \models (\forall x_1 \dots x_n)(\phi_{a_1 \dots a_n}^\beta \rightarrow \phi_{a_1 \dots a_n}^\gamma)$ whenever $\gamma < \beta$.

We prove (2) by induction on ordinals. The hard stage is the successor. By the induction hypothesis we know that the first conjunct is satisfied. The second conjunct $\bigwedge_{a_{n+1} \in A} (\exists x_{n+1})(\phi_{a_1 \dots a_{n+1}}^\alpha(x_1 \dots x_{n+1})) \wedge$ is satisfied by instantiating ' x_{n+1} ' to ' a_{n+1} '. The third conjunct is satisfied similarly because we can take a_{n+1} to be x_{n+1} .

We are now in a position to prove theorem 32.

Observe that—by (3)—for each tuple $a_1 \dots a_n$ from A and for every tuple $x_1 \dots x_n$ the truth-value of $\phi_{a_1 \dots a_n}^\beta(x_1 \dots x_n)$ decreases monotonically as β increases. (If it ever becomes false it remains false). So the truth value is eventually constant. So to each $2n$ -tuple $a_1 \dots a_n$ with tuple $x_1 \dots x_n$ we can associate the ordinal at which the truth-value of $\phi_{a_1 \dots a_n}^\beta(x_1 \dots x_n)$ *settles down*. Fix $a_1 \dots a_n$. There are only countably many tuples $x_1 \dots x_n$ so there are only countably many such ordinals. ω_1 is regular, so, for each tuple $a_1 \dots a_n$, there will come a stage by which the truth-values of $\phi_{a_1 \dots a_n}^\beta(x_1 \dots x_n)$ have settled down for all $x_1 \dots x_n$. Again, there are only countably many tuples $a_1 \dots a_n$, so (by regularity of ω_1 again) there is a countable sup of all the settling-down ordinals; call it α .

The ϕ we want is now:

$$\phi_\emptyset^\alpha \wedge \bigwedge_{\substack{n < \omega \\ a_1 \dots a_n \in A}} (\phi_{a_1 \dots a_n}^\alpha(x_1 \dots x_n) \rightarrow \phi_{a_1 \dots a_n}^{\alpha+1}(x_1 \dots x_n))$$

Now suppose $\mathfrak{B} \models \phi$ and that \mathfrak{B} is countable. We use a back-and-forth construction to show that $\mathfrak{B} \simeq \mathfrak{A}$. To do this it will suffice to establish.

$$(\forall a_{n+1} \in A)(\exists b_{n+1} \in B)(\mathfrak{B} \models \phi_{a_1 \dots a_{n+1}}^\alpha(b_1 \dots b_{n+1})) \quad (1)$$

and

$$(\forall b_{n+1} \in B)(\exists a_{n+1} \in A)(\mathfrak{B} \models \phi_{a_1 \dots a_{n+1}}^\alpha(b_1 \dots b_{n+1})) \quad (2)$$

(1) holds because $\mathfrak{B} \models \phi$, so $\mathfrak{B} \models (\exists x_{n+1})\phi_{a_1 \dots a_{n+1}}^\alpha(b_1 \dots b_n, x_{n+1})$.

To show (2) we use again the fact that $\mathfrak{B} \models \phi_{a_1 \dots a_n}^{\alpha+1}(b_1 \dots b_n)$. This gives $\mathfrak{B} \models (\forall x_{n+1}) \bigvee_{a_{n+1} \in A} \phi_{a_1 \dots a_{n+1}}^\alpha(b_1 \dots b_n, x_{n+1})$ whence, for some $a_{n+1} \in A$, $\mathfrak{B} \models \phi_{a_1 \dots a_{n+1}}^\alpha(b_1 \dots b_{n+1})$.

13.2 Chapter 4

13.2.1 A bit of pedantry

If $f(x, y)$ is a primitive recursive function of two arguments, $f(x, x)$ is a primitive recursive function of one argument. Cutland [16] calls this construction *identification* and writes as tho' it is not a special case of substitution. I'm wondering if it is actually a derived rule after all, as a special case of substitution...

Why isn't $f(x, x)$ a straightforward instance of substitution? Substitute ' x ' for ' y ' in ' $f(x, y)$ '. One obvious problem is that the rule of substitution enables us to replace a variable by a function term. Is a variable a function term in this sense? Perhaps it is. But even if it isn't we can perhaps do the following.

$proj_1^2(x, y)$ is a primitive recursive function that takes two arguments and returns the first one as its answer. So, if $f(x, y)$ is a primitive recursive function of two arguments as above, the desired unary function is presumably

$$x \mapsto f(x, proj_1^2(x, 0)).$$

Is this OK? Can we substitute constants for variables under this rubric? Constants are nullary functions after all. But then the nullary function (aka the constant) 0 has to be primitive recursive. (Wikipædia, for one, doesn't give this nullary function as a primitive recursive function.) Or should it be

$$x \mapsto f(x, proj_1^2(x, z(x)))$$

where z is the identically zero function? That seems to work. Is that what is meant?

Any apprentice pedants out there like to sort this out for us? Usual inducements ...

A message from Ben Millwood

The unfortunate thing about composition of primitive recursive functions is that there's more than one obvious thing, and though they're all equivalent or easily made so, it's hard to make sense of the basic proofs unless we're all talking the same language.

One way, and what seems to me to be the easiest, is to define the composite of the m -ary function f with m n -ary functions g_1, \dots, g_m to be the n -ary function which takes arguments (x_1, \dots, x_n) and returns

$$f(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$$

i.e. passes the *same* n arguments to every g_i and then passes the result of g_i as the i^{th} argument of f .

For example, composing a binary f with g_1 and g_2 both identity functions gives $x \mapsto f(x, x)$, an instance of what Cutland called identification. In fact, Cutland refers to identification only as something you might want to do, and then immediately proves that it is achievable by composition with appropriate projections, so *is* in fact a special case of composition (substitution).

An alternative composition method is similar, but you give different arguments to each g_i . This has the advantage that the g_i need not be the same arity, but suffers the disadvantage that the arity of your functions can then only increase, which is problematic.

With a bit of thought, it's clear that the first composition can be used to implement the second: if you have, say, g_1 with arity 3 and g_2 with arity 1 and you want to compose with f of arity 2, just compose g_1 with $proj_1^4, proj_2^4$, and $proj_3^4$ and g_2 with $proj_4^4$. This gives two four-argument functions which can then be composed with f in the usual way, to give a four-argument function that passes its first three to g_1 , its last one to g_2 , and then the result of both to f , i.e. the function you get with the second notion of composition.

Since the second composition can only increase arity, it clearly can't be used to implement the first: we'll need some way of taking one argument and plugging it into two places. But equally clearly, that's all we'll need.

Once you've done the above thinking, you might as well compose and duplicate things however you want, leaving all the projections implicit. But when you're doing your structural-induction proofs, you'll probably want to stick to just one idea, and the first is probably the easiest to formalise.

13.2.2 The Ackermann function

One of you pointed out to me that to perform this wellfounded induction we do not need the relation on which we do the induction to be the *whole* of the lexicographic order on $\mathbb{N} \times \mathbb{N}$. This is true. One can do it on the weaker relation (relation with fewer ordered pairs) given by the transitive closure of

$$\begin{aligned} \langle n, m \rangle &\triangleleft \langle n, m + 1 \rangle; \\ \langle m, A(m + 1, n) \rangle &\triangleleft \langle m + 1, n + 1 \rangle \end{aligned}$$

The point being that to get the induction to work we need

$$\langle m, A(m + 1, n) \rangle \triangleleft \langle m + 1, n + 1 \rangle$$

but we don't need

$$\langle m, A(m + 1, n) + 1 \rangle \triangleleft \langle m + 1, n + 1 \rangle$$

or

$$\langle m, A(m+1, n) \rangle \triangleleft \langle m+1, n \rangle.$$

But one needs to know that this relation \triangleleft is wellfounded and well-defined and presumably one can't do that without first proving that Ackermann is total. There may nevertheless be something enlightening one can say about this situation.

A message from Auke Booij

about the footnote on p 42.

From: Auke Booij <abb40@cam.ac.uk>
 To: tf@maths.cam.ac.uk
 Date: Tue, 3 Jun 2014 23:15:11 +0100
 Subject: Re: the status of \underline{n}

\underline{n} is a metatheoretical symbol: inside the theory, it represents one specific symbol $S(\dots S(0) \dots)$, but it is “meta-generated” by a variable of the theory in which we phrased our theory (aka the metatheory). So in the metatheory, \underline{n} is a function of the meta-variable n , which generates some symbol (e.g. a Church numeral) which can be interpreted inside the theory (e.g. lambda calculus).

Similarly, in the metatheory, we generate [logical formulas for the theory] such as the ones you give—the theory itself has no way to do that (since there is no internal concept of quoting of logical formulas). Hence, in the metatheory, we generate [logical formulas for the theory] as a function of the metavariable n (ie. as a function of n , which is a variable in the language of the metatheory rather than in the theory).

I think that should answer your suspicion in the footnote on page 30 of the Part III computability notes.

So on page 106, you are defining a miaow function for every possible metatheoretical choice of n . I don't think this is what you mean (I'd like to give a counterargument using nonstandard natural number objects, but that doesn't seem to work). What you instead want to say is that “we can test for equality of Church numerals within lambda calculus”. Indeed, if you are writing “ $\text{snd}(\text{hd } x) = \underline{n}$ ”, you are expressing that in the theory of lambda calculus, we (somehow) test for equality of the lambda term “ $\text{snd}(\text{hd } x)$ ” with the constant symbol “ \underline{n} ”. But there is no need to involve any kind of metavariables or quoting for that:

```
and :=  $\lambda bb'.bb'$  false
equal :=  $\lambda nm. \text{and}(\text{iszero}(mpred(nsucc0)))(\text{iszero}(npred(msucc0)))$ 
```

The implementation of `pred` (predecessor) is a bit contrived but possible (see e.g. http://en.wikipedia.org/wiki/Church_encoding#Derivation_of_predecessor_function).

Indeed, the later use in that definition of miaow of non-underlined n therefore becomes ill-typed (what does the metavariable n mean to the theory?).

In my humble opinion, the interpretation of relative computation for models versus theories I told you about a few weeks ago is a good way to understand these things.

13.3 Chapter 5

13.4 Chapter 6

13.5 Chapter 7

13.6 Chapter 10

13.7 Chapter 11

Bibliography

- [1] J-P Allouche and J Shallit. “Automatic Sequences: Theory, Applications Generalisations”. CUP 2003.
- [2] Peter Aczel and Michael Rathjen, draught of book on constructive set theory.
<http://www1.maths.leeds.ac.uk/~rathjen/book.pdf>
- [3] Bachmann: *Transfinite Zahlen* Springer, 1967.
- [4] T. P. Baker, J. Gill, R. Solovay. “Relativizations of the $P = ? NP?$ Question”. SIAM Journal on Computing, 4(4): 431-442 (1975)
- [5] Baumslag. Review of [21] Bull Am Maths Soc **31** 1994 pp 86–91
- [6] J. L. Bell “A Primer of Infinitesimal Analysis”. Cambridge University Press, 1998. Second Edition, 2008.
- [7] George S Boolos and Richard C Jeffrey “Computability and Logic”, various editions. CUP
- [8] W Buchholz and Stan Wainer. “Provably Computable functions and the fast-growing hierarchy”. Logic and Combinatorics, Proceedings of a Summer Research Conference held August 4-10, 1985, Contemporary Mathematics **65** American Mathematical Society 1987. pp. 179–198.
- [9] Douglas. S. Bridges. “Computability, a Mathematical Sketchbook” Springer Graduate texts in Mathematics **146** 1994.
- [10] Bunder, M. “The Logic of Inconsistency”. Journal of Non-Classical Logic **6** 1989 pp 57–62
- [11] Martin Gardner The Annotated Alice. lib.rmvoz.ru/sites/default/files/fail/carroll_lewis_-_the_annotated_alice.pdf
- [12] C.C. Chang “Some remarks on the model theory of infinitary Languages” in LNM **72**
- [13] W. Craig. “On Axiomatizability within a System”, JSL **18** No. 1, pp. 30–32 (1953).

- [14] Craig and Vaught, “Finite axiomatizability using additional predicates”, JSL, **23** (1958), pp. 289–308.
- [15] James Cummings “Notes on Singular Cardinal Combinatorics”, Notre Dame J. Formal Logic **46**, Number 3 (2005), pp 251–282. http://www.math.cmu.edu/users/jcumming/papers/1911_001.pdf
- [16] N.J. Cutland “Computability, an Introduction to Recursive Function Theory”, CUP
- [17] Christian Delhommé, “Automaticité des ordinaux et des graphes homogènes.” C. R. Acad. Sci. Paris Ser I **339** pp 5–10 (2004). available from <http://personnel.univ-reunion.fr/delhomme/filename.dvi>
- [18] Radu Diaconescu, “Axiom of Choice and Complementation”. Proc. AMS **51** (1975) 176–178.
- [19] John Doner and Alfred Tarski. “An Extended Arithmetic of ordinal numbers”. Fundamenta Mathematicæ **LXV**(1969) pp. 95–127. Also on <http://www.math.ucsb.edu/~doner/articles/>.
- [20] Ehrenfeucht, A. “Polynomial functions with exponentiation are wellordered” *Algebra universalis* **3** December 1973, Issue 1, pp 261–262
- [21] David B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson and W. P. Thurston. “Word Processing in Groups”. Jones and Bartlett 1992
- [22] Benson Farb “Automatic Groups a guided Tour”. Enseignement mathématique **38** (1992) pp 291–313. Also at <http://retro.seals.ch/digbib/view?rid=enstat-001:1992:38:528&id=&id2=&id3=>
- [23] Forster, T. E. Talk to the TMS www.dpmms.cam.ac.uk/~tf/TMStalk2012.pdf
- [24] Forster, T. E. Tutorial on countable Ordinals. www.dpmms.cam.ac.uk/~tf/fundamentalsequence.pdf
- [25] Martin Gardner “Logic Machines and Diagrams” University of Chicago Press and Harvester Press second edition 1982 ISBN 0-226-28243-0
- [26] R. Gilman, “Groups with a rational cross-section”, in: Combinatorial Group Theory and Topology, Annals of Math. study 111, ed. by S. Gersten and J. Stallings,
- [27] Girard Lafont and Taylor “Proofs and Types” CUP
- [28] M. Goldberg: On the Recursive Enumerability of Fixed-Point Combinators BRICS RS-05-1. 2005 University of Aarhus

- [29] Andrzej Grzegorzcyk “Some classes of Recursive functions”, *Rozprawy Matematyczne* **IV** 1953
- [30] Hardy, G. H. “A theorem concerning the infinite cardinal numbers”. *Quarterly J. of Pure and Applied Mathematics*. **35** (1903) 87-94.
- [31] Wilfrid Hodges: *Model theory*
- [32] Douglas Hofstadter “Gödel, Escher, Bach”.
- [33] H.J. Keisler “Model Theory for Infinitary Logic” *North Holland Studies in Logic and the foundations of mathematics* **62**, 1971
- [34] Hummel, T. L. “Effective versions of Ramsey’s theorem: avoiding the cone above 0”. *Journal of Symbolic Logic* **59** (1994) pp. 1301–1325.
- [35] R.W. Kaye. “Tennenbaum’s theorem for models of arithmetic”. <http://web.mat.bham.ac.uk/R.W.Kaye/papers/tennenbaum/tennenbaum.pdf>
- [36] Bakhadyr Khoussainov and Sasha Rubin “Some Thoughts On Automatic Structures”, Auckland 2002, linked from Wikipædia page on Automatic Groups.
- [37] S.C. Kleene, *Introduction to Metamathematics*.
- [38] S.C. Kleene, “Finite Axiomatizability of theories in the predicate calculus using additional predicate symbols” *Memoirs of the AMS*, **10**.
- [39] Lerman, Manuel. “Degrees of Unsolvability, local and Global theory”. *Perspectives in Mathematical Logic*, Springer Verlag 1983.
- [40] Hilbert Levitz “An ordinal bound for the set of polynomial functions with exponentiation”. *Algebra universalis* **8** (1978) 233–243
- [41] M. Makkai. Review of [38] *JSL* **36** (1971), pp. 334–335.
- [42] D. Marker “Model Theory”
- [43] A.R.D. Mathias “Weak systems of Gandy, Jensen and Devlin” *Trends in Mathematics* Springer 2006, pp 149-22
- [44] Mendelson, E. “Introduction to Mathematical Logic”. various editions Van Nostrand. We want the *first* edition.
- [45] Piergiorgio Odifreddi, “Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers” (*Studies in Logic and the Foundations of Mathematics* **125**) 1992
- [46] Larry Paulson’s Computer Science 1b functional programming notes: <http://www.cl.cam.ac.uk/~lp15/papers/Notes/Founds-FP.pdf>

- [47] Valery Plisko “A Survey of Propositional Realizability Logic” *Bull. S. Log* **15** (2009) pp 1–42.
- [48] by D. Richardson “Solution of the identity problem for integral exponential functions” *Zeilschr. f. math. Logik und Grundlagen d. Math.* **15**, S. 333–340 (1969)
- [49] Hartley Rogers
- [50] Rozsa Péter. “Recursive functions” Third (English) edition Academic Press 1967.
- [51] Andrew Pitts Lecture Notes for 1a RLFA:
<http://www.cl.cam.ac.uk/teaching/1112/RLFA/materials.html>
- [52] W. v O. Quine “Set theory and its Logic”. Harvard Belknap Press 1969
- [53] H. E. Rose, “Subrecursion: functions and hierarchies”. Oxford Logic Guides **9** OUP 1984.
- [54] Schütte, K. “Beweistheoretische Erfassung der unendlichen Induktion in der Zahlentheorie”. *Math Ann* **122** pp 369–389.
- [55] Diana Schmidt. “Built-up Systems of Fundamental Sequences and Hierarchies of Number-Theoretic Functions”. *Arch. Math. Logik.* **18** pp 47–53 1976.
- [56] Schwichtenberg and Wainer. “Proofs and Computations”. CUP 2012
- [57] Scott, D. “Logic with denumerably long formulæ and finite strings of quantifiers” in Addison Henkin Suppes (eds) “The theory of models” Studies in Logic and the Foundations of Mathematics NH 1965.
- [58] Scott D.S. Semantical Archæology, a parable. In: Harman and Davidson eds, *Semantics of Natural Languages*. Reidel 1972 pp 666–674.
- [59] Dana Scott, “Axiomatizing set theory” in Jech, Thomas, J., ed., *Axiomatic Set Theory II*, Proceedings of Symposia in Pure Mathematics 13. American Mathematical Society Volume XIII Part II, 1974
- [60] Harold Simmons “The Ackermann functions are not optimal, but by how much?” *JSL* **75** 1 (march 2010) pp 289–313.
- [61] Peter Smith “An Introduction to Gödel’s Theorems” 2nd Edition, Cambridge University Press 2013 ISBN 9781107606753.
<http://www.cambridge.org/gb/knowledge/isbn/item7137024/>
- [62] http://www.logicmatters.net/resources/pdfs/tennenbaum_new.pdf

- [63] Stearns, R. E., Hartmanis, J., Lewis, P. M. "Hierarchies of memory limited computations". Sixth Annual Symposium on Switching Circuit Theory and Logical Design, 1965. SWCT 1965. Date of Conference: 6-8 Oct. 1965 pp 179–190
- [64] Frank Stephan. Recursion theory preprint 125pp.
- [65] Patrick Suppes "Introduction to Logic". Dover
- [66] Jaap van Osten "Realizability - An Introduction to its Categorical Side"
- [67] Väänänen, Jouko. Models and Games CUP (ISBN-13: 9780521518123)

Chapter 14

Answers to selected questions

Chapter 3: Recursive Datatypes

Exercise 5

Discussion

This is a beautiful question, co's it touches several important points. It tests your understanding of structural induction; it tests your ability to do the fiddly manipulation necessary to perform the inductive step; it underlines the importance of having a sufficiently strong induction hypothesis, and finally it makes a point about dereferencing.

So: we have a propositional language—a recursive datatype of formulæ—which starts off with three propositional letters (“literals”) ‘ a ’, ‘ \top ’ and ‘ \perp ’. We then build up compound formulæ by means of the constructors ‘ \wedge ’, ‘ \vee ’ and ‘ \neg ’. We have a *length* function defined on objects in the datatype of formulæ, written with two vertical bars as in the question, which is roughly what you think it is—so that the length of a literal is 1, and the length of a conjunction (or a disjunction) of two formulæ is one plus the sum of their lengths, and the length of the negation of a formula is one plus the length of the formula. Evidently the question-designer thought that the length of a ‘(’ or a ‘)’ is zero!

One tends naturally to write the second half of the preceding paragraph with expressions like

$$|A \wedge B| = |A| + |B| + 1.$$

This looks fair enough, and in some sense it is, but we need to be clear about the conventions we are using. The letter ‘ A ’ by itself is a single symbol, so a pedant might insist that $|A| = 1$. This is wrong of course: the letter ‘ A ’ is not a formula, but a variable ranging over formulæ... when looking for the length $|A|$ of A we have to *see through* the variable all the way to the value it takes—and that value is a formula. All this is well and good, but it can cause some confusion when we start thinking about expressions like: $|A \vee B|$. The constructor ‘ \vee ’ is something we put between two formulæ to make a new formula; we don’t put it between two *names* of formulæ or between two *pointers* to formulæ! Until we have a convention to make our practice OK, writing things like ‘ $|A \vee B|$ ’ should generate a **syntax error** warning. If you look back to page 21 where this exercise first appears you will find that i wrote

“...length of a literal is 1, and the length of a conjunction (or a disjunction) of two formulæ is one plus the sum of their lengths...”

... and this is syntactically correct. When we wrote ‘ $|A \wedge B|$ ’ we should really have written ‘| the conjunction of A and B |’.

There are two ways of dealing with this. One is to have explicit names for the constructors, as it might be ‘conjunction of ...’ and ‘disjunction of ...’ and ‘negation of ...’ This makes huge demands on our supply of alphanumerics. The other solution is to have a kind of **environment** command that creates an environment within which [deep breath]

*Constructors applied to **pointers** to objects*

construct

***pointers** to the objects thereby constructed.*

Inside such a context things like ‘ $|A \vee B|$ ’ have the meaning we intend here. There is a culture within which this environment is created by the ‘ \ulcorner ’ symbol (L^AT_EX: **\ulcorner**) and closed by the ‘ \urcorner ’ symbol (L^AT_EX: **\urcorner**). In practice people tend to leave these things out. The fact that this is—apparently—a safe strategy tells us quite a lot about the skills of our language module: it’s very good at dereferencing (among other things)

Thus we should/should-have posed the question as:

“Define the length of a Boolean proposition by structural induction as follows:

$$\begin{aligned} |a| &= 1, \\ |\top| &= 1, \\ |\perp| &= 1, \\ |\ulcorner A \wedge B \urcorner| &= |A| + |B| + 1, \\ |\ulcorner A \vee B \urcorner| &= |A| + |B| + 1, \\ |\ulcorner \neg A \urcorner| &= |A| + 1. \end{aligned}$$

[or *something* like that, with the corners placed correctly!]

Define a translation which eliminates disjunction from Boolean expressions by the following recursion:

$$\begin{aligned} tr(a) &= a, \quad tr(\top) = \top, \quad tr(\perp) = \perp, \\ \ulcorner tr(A \wedge B) \urcorner &= tr(A) \wedge tr(B), \\ tr(A \vee B) &= \neg(\neg tr(A) \wedge \neg tr(B)), \\ tr(\neg A) &= \neg tr(A)^\neg. \end{aligned}$$

Prove by structural induction on Boolean propositions that

$$|\ulcorner tr(A) \urcorner| \leq 3|A| - 1^\neg,$$

for all Boolean propositions A ."

The above use of corner quotes illustrates how there is no restriction that says that the scope of the corner quotes has to live entirely inside a single formula. I use corner quotes in what follows, but (although—I *think*—I have put them in correctly) they can be inserted correctly in more than one way.

The Proof by Structural Induction

We aspire to prove by structural induction on the recursive datatype of formulae that

$$(\forall A)(|tr(A)| \leq 3 \cdot |A| - 1)$$

The base case we verify easily. The induction step has three cases

- \neg If $|tr(A)| \leq 3 \cdot |A|$ what is $|\ulcorner tr(\neg A) \urcorner|$? $\ulcorner tr(\neg A) \urcorner = \neg tr(A)^\neg$ so $|\ulcorner tr(\neg A) \urcorner| = |\neg tr(A)|^\neg$, and $|\ulcorner \neg tr(A) \urcorner|$ is $|tr(A)| + 1$ which is certainly $\leq 3 \cdot |\neg A|^\neg$.
- \wedge If $|tr(A)| \leq 3 \cdot |A|$ and $|tr(B)| \leq 3 \cdot |B|$ what is $|\ulcorner tr(A \wedge B) \urcorner|$? $\ulcorner tr(A \wedge B) \urcorner$ is $\ulcorner tr(A) \urcorner \wedge \ulcorner tr(B) \urcorner$. By induction hypothesis $|tr(A)| \leq 3 \cdot |A| - 1$ and $|tr(B)| \leq 3 \cdot |B| - 1$ so $|\ulcorner tr(A) \urcorner \wedge \ulcorner tr(B) \urcorner| \leq (3 \cdot |A| - 1) + (3 \cdot |B| - 1) + 1$. The final '+1' is for the ' \wedge '. This rearranges to $|\ulcorner tr(A) \urcorner \wedge \ulcorner tr(B) \urcorner| \leq 3 \cdot (|A| + |B|) - 1$ but $|A| + |B| \leq |\ulcorner A \wedge B \urcorner|$ whence $|\ulcorner tr(A) \urcorner \wedge \ulcorner tr(B) \urcorner| \leq 3 \cdot (|A \wedge B|) - 1^\neg$ and finally $|\ulcorner tr(A \wedge B) \urcorner| \leq 3 \cdot (|A \wedge B|) - 1^\neg$.
- \vee If $|tr(A)| \leq 3 \cdot |A|$ and $|tr(B)| \leq 3 \cdot |B|$ what is $|tr(A \vee B)|$? $\ulcorner tr(A \vee B) \urcorner$ is $\ulcorner \neg(\neg tr(A) \wedge \neg tr(B)) \urcorner$. What is the length of this last expression? Clearly it's going to be $|tr(A)| + |tr(B)| + 1$ for the outermost ' \neg ' + one for the ' \neg ' attached to $tr(A)$ + one for the ' \neg ' attached to $tr(B)$ +

one for the ‘ \wedge ’ ... giving $|tr(A)| + |tr(B)| + 4$. By induction hypothesis $|tr(A)| \leq 3 \cdot |A| - 1$ and $|tr(B)| \leq 3 \cdot |B| - 1$ so we have

$$\begin{aligned} \lceil |tr(A \vee B)| \rceil &\leq (3 \cdot |A| - 1) + (3 \cdot |B| - 1) + 4 \rceil. \text{ We can rearrange this to} \\ \lceil |tr(A \vee B)| \rceil &\leq 3 \cdot (|A| + |B|) - 1 - 1 + 4 \rceil \text{ and further to} \\ \lceil |tr(A \vee B)| \rceil &\leq 3 \cdot (|A| + |B|) + 2 \rceil. \end{aligned}$$

Now $|A| + |B| = \lceil |A \vee B| \rceil - 1$ so we can substitute getting

$$\begin{aligned} \lceil |tr(A \vee B)| \rceil &\leq 3 \cdot (\lceil |A \vee B| \rceil - 1) + 2 \rceil \text{ and rearrange again to get} \\ \lceil |tr(A \vee B)| \rceil &\leq 3 \cdot \lceil |A \vee B| \rceil - 1 \rceil \text{ as desired.} \end{aligned}$$

A final thought ... I wouldn't mind betting that quite a lot of thought went into this question. We've proved $|tr(A)| \leq 3 \cdot |A| - 1$ so we've certainly also proved the weaker claim $|tr(A)| \leq 3 \cdot |A|$. However wouldn't stake my life on our ability to prove the weaker claim by induction. You might like to try ... i'm not going to!

Exercise 15

A D-finite set is a set without a countably infinite subset.

(i) Prove that every hereditarily D-finite set is inductively finite.

The set of all hereditarily finite sets is countably infinite, so every set of hereditarily finite sets is countable finite or countably infinite. Consider a D-finite set x of hereditarily D-finite sets. By induction hypothesis all its members are hereditarily finite, so x is either inductively finite or countably infinite. It is not countably infinite (being D-finite) so it must be inductively finite.

There is a slightly easier proof. The collection V_ω aka H_{\aleph_0} of hereditarily inductively finite sets contains all its D-finite subsets. This is because it is countable and any D-finite subset of a countable set is inductively finite.

(ii) Provide a constructive proof that every hereditarily K-finite set is N-finite.

A proof from Andreas Blass:

“First, I claim that equality between hereditarily K-finite sets is decidable, i.e., either $x = y$ or not $x = y$. This is proved by induction on hereditarily K-finite sets x (for all y simultaneously) as follows. Given (hereditarily K-finite) x and y , we have, for all members x' of x and y' of y , that $x' = y'$ is decidable, by induction hypothesis. But decidability is preserved by quantification over K-finite sets and by conjunction, so we also have decidability of

$$(\forall x' \in x)(\exists y' \in y)x' = y'$$

and

$$(\forall y' \in y)(\exists x' \in x)x' = y'.$$

That is, we have decidability of $x = y$.”

“To finish the proof, I claim that K-finiteness of a set z plus decidability of equality between its members implies N-finiteness of z . (This is undoubtedly well-known, but I’ll give the proof anyway for completeness.) Proceed by induction on K-finite sets, the case of the empty set being trivial. So suppose $a \cup \{x\}$ has decidable equality between all its members (where a is a K-finite set for which the result is known to hold). In particular, each member of a is either equal to x or not. Using again that quantification over K-finite sets preserves decidability, we find that x is either in a or not. So $a \cup \{x\}$ is either just a (which is N-finite by induction hypothesis, because equality between its members is decidable) or the disjoint union of a and $\{x\}$, which is N-finite by definition of N-finiteness. That completes the proof.”

Exercise 16

REMARK 25

Suppose f is monotone and injective: $(\forall xy)(x \subseteq y \longleftrightarrow f(x) \subseteq f(y))$.

Let $A := \bigcap \{x : \mathcal{P}(f(x)) \subseteq x\}$.

Then A is not a set.

Proof:

Suppose there is such a set A ; we will show that $f(A)$ both is and is not a member of A .

First we prove that $f(A) \notin A$.

The idea is that if $f(A) \in A$ then $A \setminus \{f(A)\}$ is also a fixpoint, contradicting minimality of A .

We want

$$\mathcal{P}(f(A \setminus \{f(A)\})) \subseteq A \setminus \{f(A)\}$$

which is to say

$$X \subseteq f(A \setminus \{f(A)\}) \rightarrow X \in A \setminus \{f(A)\}$$

which is

$$X \subseteq f(A \setminus \{f(A)\}) \rightarrow X \in A \wedge X \neq f(A)$$

Now f is monotone and injective, so $f(A \setminus \{f(A)\})$ is a proper subset of $f(A)$ and no subset of $f(A \setminus \{f(A)\})$ can possibly be a subset of $f(A)$, let alone equal to it, so we have only the first conjunct to worry about:

$$X \subseteq f(A \setminus \{f(A)\}) \rightarrow X \in A$$

$f(A \setminus \{f(A)\}) \subseteq f(A)$ by monotonicity of f ; $f(A) \in A$ by assumption, so $f(A \setminus \{f(A)\})$ is in A and so too is any subset of $f(A \setminus \{f(A)\})$, since A is a power set, so is closed under \subseteq .

For the other horn, we use the fact that A is a fixpoint for $\mathcal{P} \circ f$: $(\forall x)(x \subseteq f(A) \rightarrow x \in A)$. Now specialise ‘ x ’ to ‘ $f(A)$ ’ to obtain $f(A) \subseteq f(A) \rightarrow f(A) \in A$, which tells us that $f(A) \in A$ after all. ■

Note that the only set-theoretic axiom we have used is subscission.

Exercise 17

Let \mathbb{N}^* be $\{n : q(n)\}$; we claim $\mathbb{N}^* = \mathbb{N}$.

Clearly \mathbb{N}^* contains 0 and is closed under S and so $\mathbb{N} \subseteq \mathbb{N}^*$. (i.e., we can prove $m \in \mathbb{N}^*$ for all $m \in \mathbb{N}$ by induction).

For the other direction we will justify induction over \mathbb{N}^* : this will enable us to prove that everything in \mathbb{N}^* is in \mathbb{N} . Suppose (i) that $F(0)$ and (ii) that $(\forall n)(F(n) \rightarrow F(n+1))$, and take $a \in \mathbb{N}^*$. Suppose, *per impossibile*, that $\neg F(a)$. Then $\{m : m \leq a \wedge \neg F(m)\}$ contains a and is closed under P (by (ii)), and so must contain 0, contradicting (i). ■

Here is another proof that $\mathbb{N}^* \subseteq \mathbb{N}$.

Let $m \in \mathbb{N}^* \setminus \mathbb{N}$. Set $M = \bigcap \{Y : m \in Y \wedge P[Y \subseteq Y]\}$. (Notice that the intersection remains the same if we take it not over all Y with that property, but only over those Y satisfying additionally $Y \subseteq \{k : k \leq m\}$, all of which are finite. This is because if $m \in Y \wedge P[Y \subseteq Y]$ then the same goes for $Y \cap \{k : k \leq m\}$.) M is finite. Notice that $M \setminus \mathbb{N}$ contains m and is closed under P and so is a superset of M , whence M and \mathbb{N} are disjoint. But $0 \in M$ by hypothesis.

Exercise 2

An answer from Maria Gorinova.

Provide a sequent calculus or natural deduction proof that

$$\forall x(\forall y(R(y, x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)) \text{ and } \forall xy(R'(x, y) \rightarrow R(x, y))$$

together imply $\forall x(\forall y(R'(y, x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow \forall z(\phi(z))$.

SEQUENT PROOF:

$$\begin{aligned} \text{Let} \quad & XY = \forall xy(R'(x, y) \rightarrow R(x, y)), \\ & Y = \forall y(R(y, x) \rightarrow \phi(y)), \\ & Y' = \forall y(R'(y, x) \rightarrow \phi(y)) \text{ and} \\ & Y'_z = \forall y(R'(y, z) \rightarrow \phi(y)). \end{aligned}$$

We want to prove

$$(\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z))) \wedge XY \rightarrow (\forall x(Y' \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)))$$

$$\frac{\frac{\frac{R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z), R(y, x), R'(y, x)}{(b)} \quad \frac{R(y, x), R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z), R(y, x)}{(b)} \quad \frac{R'(y, x) \rightarrow R(y, x), R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z), R(y, x)}{(2 \times \forall I)} \quad \frac{\forall xy(R'(x, y) \rightarrow R(x, y)), R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z), R(y, x)}{(expand)} \quad \frac{XY, R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z), R(y, x)}{(***)}}{(\rightarrow I)}$$

$$\begin{array}{c}
\frac{(*)}{XY, R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z), R(y, x)} \quad \frac{XY, \phi(y), R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z)}{(*)} \quad (b) \\
\hline
\frac{XY, R(y, x) \rightarrow \phi(y), R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z)}{XY, \forall y(R(y, x) \rightarrow \phi(y)), R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z)} \quad (\forall I) \\
\hline
\frac{XY, Y, R'(y, x), R'(y, z) \vdash \phi(x), \phi(y), \phi(z)}{XY, Y, \vdash \phi(x), \phi(z), R'(y, x) \rightarrow \phi(y), R'(y, z) \rightarrow \phi(y)} \quad (2 \times \rightarrow r) \\
\hline
\frac{XY, Y, \vdash \phi(x), \phi(z), R'(y, x) \rightarrow \phi(y), R'(y, z) \rightarrow \phi(y)}{XY, Y, \vdash \phi(x), \phi(z), \forall y(R'(y, x) \rightarrow \phi(y)), \forall y(R'(y, z) \rightarrow \phi(y))} \quad (2 \times \forall r) \\
\hline
\frac{XY, Y, \vdash \phi(x), \phi(z), Y', Y'_z}{(*)} \quad (expand)
\end{array}$$

$$\begin{array}{c}
\frac{(*)}{XY, Y, \vdash \phi(x), \phi(z), Y', Y'_z} \quad \frac{XY, Y, \phi(x), \phi(z) \vdash \phi(x), \phi(z)}{(*)} \quad (b) \\
\hline
\frac{XY, Y, Y'_z \rightarrow \phi(z) \vdash \phi(x), \phi(z), Y'}{(**)} \quad (\rightarrow I)
\end{array}$$

$$\begin{array}{c}
\frac{(**)}{XY, Y, Y'_z \rightarrow \phi(z) \vdash \phi(x), \phi(z), Y'} \quad \frac{XY, Y, \phi(x), Y'_z \rightarrow \phi(z) \vdash \phi(x), \phi(z)}{(*)} \quad (b) \\
\hline
\frac{XY, Y, Y'_z \rightarrow \phi(x), Y'_z \rightarrow \phi(z) \vdash \phi(x), \phi(z)}{XY, Y, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(x), \phi(z)} \quad (2 \times \forall I) \\
\hline
\frac{XY, Y, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(x), \phi(z)}{XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z), Y \rightarrow \phi(x)} \quad (\rightarrow r) \\
\hline
\frac{XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z), Y \rightarrow \phi(x)}{XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z), \forall x(Y \rightarrow \phi(x))} \quad (\forall r) \\
\hline
(*)
\end{array}$$

$$\begin{array}{c}
\frac{(*)}{XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z), \forall x(Y \rightarrow \phi(x))} \quad \frac{\phi(z), XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z)}{\forall z(\phi(z)), XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z)} \quad (b) \\
\hline
\frac{XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z), \forall x(Y \rightarrow \phi(x))}{\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)), XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z)} \quad (\forall I) \\
\hline
\frac{\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)), XY, \forall x(Y' \rightarrow \phi(x)) \vdash \phi(z)}{\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)), XY, \forall x(Y' \rightarrow \phi(x)) \vdash \forall z(\phi(z))} \quad (\rightarrow r) \\
\hline
\frac{\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)), XY \vdash \forall x(Y' \rightarrow \phi(x)) \rightarrow \forall z(\phi(z))}{\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)) \wedge XY \vdash \forall x(Y' \rightarrow \phi(x)) \rightarrow \forall z(\phi(z))} \quad (\wedge I) \\
\hline
\vdash ((\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z))) \wedge XY) \rightarrow (\forall x(Y' \rightarrow \phi(x)) \rightarrow \forall z(\phi(z))) \quad (\rightarrow r)
\end{array}$$

□

Exercise 7

(i) Every $X \subseteq \mathbb{N}$ such that $0 \in X$ has a \leq -least member, namely 0 itself. Suppose every X that n belongs to has a least element. Consider $S(n)$. Let X be an arbitrary set containing $S(n)$. If it contains n then by induction hypothesis it has a minimal element (tho' not necessarily n itself!) If not, take the set $\{k - 1 : k \in X\}$. This contains n , since X contained $S(n)$, and so it has a minimal element a . But then $S(a)$ is the minimal element of X . ■

(ii) True for $n = 0$; suppose true for n , and suppose $S(n) \in X$. Consider $X \setminus \{S(n)\}$. Does it contain anything $\leq n$? If not, then $S(n)$ is the desired minimal element. If it does, then it has a minimal element by induction hypothesis. ■

Those of you with refined palates will probably notice that these proofs are not constructive. The only constructive account of induction is structural induction.

Exercise 18

The wellfounded part of a binary structure $\langle X, R \rangle$ is the \subseteq -least set Y s.t. $x \in Y$ whenever $\{y : R(y, x)\} \in Y$.

Chapter 4 Functions

Exercise 20

The lexicographic product of two wellfounded strict partial orders is wellfounded.

The pointwise product of two wellfounded strict posets is wellfounded by horn-ness, and every subset of a wellfounded relation is wellfounded because ‘wellfounded’ is \forall in L_{ω_1, ω_1} .

Exercise 23

Primitive recursion on lists. We need composition and projection. We also need gadgets corresponding to the basic functions. The function that always returns the null list is an obvious candidate. We will need `cons` here just as we need successor in \mathbb{N} . The recursion gadget is presumably

$$\begin{aligned} f(\text{null}, \vec{x}) &= h(\vec{x}); \\ f(\alpha :: l, \vec{x}) &= g(\alpha, f(l, \vec{x}), l, \vec{x}). \end{aligned}$$

where the \vec{x} s can be anything, so certainly either α s or α -lists.
Show how to define `tail` like predecessor.

More work to do here

$$\begin{aligned} \text{tail}(\text{null}) &= \text{null}; \\ \text{tail}(\alpha :: l) &= l. \end{aligned}$$

Comment on the retype of hereditarily finite sets. (Set Theory and Logic 2012/3 sheet 3 q 6); if x and y are hereditarily finite so is $x \cup \{y\}$. This is not free, so the recursions are not safe.

One wants to say

$$\begin{aligned} f(\emptyset, \vec{x}) &= g(\vec{x}); \\ f(a \cup \{b\}, \vec{x}) &= h(f(a), b, \vec{x}) \end{aligned}$$

but we cannot be confident of a unique answer, since $a \cup \{b\}$ might be the same as $c \cup \{d\}$.

Exercise 25

Elementary, just a quick reality check. You do it by induction on ‘ i ’.

$$\begin{aligned} f_0(m, k) &:= m + k; \\ f_{n+1}(m, 0) &:= m;^1 \\ f_{n+1}(m + 1, k + 1) &= f_n(m, f_{n+1}(m + 1, k)). \end{aligned}$$

¹This is surely correct. $f_{n+1}(m, 0)$ must be the result of doing f_n zero times to m and this must be m . The consideration that causes me slight unease is that according to this line of thought $m \cdot 0$ should be m not 0. So the function we call multiplication— $m \cdot j$ —is actually not f_1 but rather $f_1(m, j - 1)$. Not that it matters. But one would have expected to see something about it in the literature. Ben Millwood says not to worry. Perhaps he’s right.

$$\begin{aligned} f_2(m+1, k+1) &= f_1(m, f_2(m+1, k)) \\ f_2(m+1, k+1) &= m \cdot (f_2(m+1, k) + 1). \end{aligned}$$

Define an operation $\mathcal{DT} : (\mathbb{N}^2 \rightarrow \mathbb{N}) \rightarrow (\mathbb{N}^2 \rightarrow \mathbb{N})$ by

$$\begin{aligned} (\mathcal{DT}f)(m, 0) &:= m; \\ (\mathcal{DT}f)(0, m) &:= (\mathcal{DT}f)(m+1, 0); \\ (\mathcal{DT}f)(m+1, k+1) &:= f(m, (\mathcal{DT}f)(m+1, k)). \end{aligned}$$

Observe that if f is primitive recursive so is $\mathcal{DT}(f)$. That powers the induction.

Exercise 26

It isn't what you think!

The next Doner-Tarski operation beyond exponentiation is declared by

$$\begin{aligned} \beta \uparrow\uparrow 0 &= \beta; \\ \beta \uparrow\uparrow (\alpha + 1) &= (\beta \uparrow\uparrow \alpha)^\beta; \\ &\text{taking sups at limits.} \end{aligned}$$

Thus $x \uparrow\uparrow 1 = x^x$; $x \uparrow\uparrow 2 = (x^x)^x = x^{x^2}$; $x \uparrow\uparrow 3 = (x^{x^2})^x = x^{x^3}$; and presumably $x \uparrow\uparrow n = x^{x^n}$ for $n \in \mathbb{N}$.

And, when $\beta = \omega$.

$$\begin{aligned} \omega \uparrow\uparrow 0 &= \omega; \\ \omega \uparrow\uparrow (\alpha + 1) &= (\omega \uparrow\uparrow \alpha)^\omega; \\ &\text{taking sups at limits.} \end{aligned}$$

It's worth noting that if you get it the other way round, so that the successor step is

$$\omega \uparrow\uparrow (\alpha + 1) = \omega^{(\omega \uparrow\uparrow \alpha)}$$

—which looks more natural—you find that $\omega \uparrow\uparrow \omega = \epsilon_0$ and $\omega \uparrow\uparrow (\omega + 1) = \omega^{\omega \uparrow\uparrow \omega} = \omega^{\epsilon_0} = \epsilon_0$ so $\beta \mapsto \omega \uparrow\uparrow \beta$ grinds to a shuddering halt, and is not strictly increasing, let alone normal.

Exercise 27

Show that, if f is primitive recursive, so are

1. the function $\sum_f(n) = \sum_{0 \leq x < n} f(x)$ that returns the sum of the first n values of f ;
 $\sum_f(0) := 0$; $\sum_f(S(n)) = \sum_f(n) + f(n)$.
 $\sum_f(0)$ must be zero because it is the sum of the empty set of numbers!
and

2. the function $\prod_f(n) = \prod_{0 \leq x \leq n} f(x)$ that returns the product of the first n values of f .

$$\prod_f(0) := 1; \quad \prod_f(S(n)) = \prod_f(n) \cdot f(n).$$

Exercise 28

I don't know what proof you were given, but here is a sketch of how to show that there are primitive recursive upper bounds.

(Notation: $\alpha \rightarrow (\beta)_\delta^\gamma$)

We will start off by proving $\omega \rightarrow \omega_2^2$. (There are many proofs: I think this one is due to Rado.)

We are given a two-colouring (red and blue) of all the edges in the complete undirected graph on \aleph_0 vertices. We are going to form an infinite finite-branching tree whose nodes are labelled with natural numbers. Below 0, to the left and to the right, respectively, we place the first natural number z such that there are infinitely many numbers greater than z to which z is connected by a blue edge (red edge respectively) and—strictly temporarily—we associate to it that same set of greater numbers. We now build the tree recursively. Below each growing bud (which is a number with a set of greater numbers temporarily associated with it) we place—to the left (and to the right)—the smallest member x of the set-temporarily-associated-to-the-bud such that there are infinitely many larger members of that set to which x is connected by a blue (resp. red) edge.

As we deal with each node we throw away the set that has been temporarily associated with it. When we have finished we have a tree in which every node has either one or two children. It cannot have no children at all since whenever you split an infinite set into two bits, one of the two is infinite. This is a finite-branching infinite tree and must have an infinite branch.² This infinite branch either has infinitely many left turns in it, or infinitely many right turns.

Clearly the choice of a two-colouring was unnecessary: the same construction would have worked for any finite number of colours n . So we have proved $\omega \rightarrow (\omega)_n^2$.

Observe that had we started with a set that was merely finite, and we wanted a monochromatic set of size n we would find (working backwards) that the path through the tree would have to be of length $2n$, so the binary tree would have to be of height $2n$ so we would have had to have started with 2^{2n} elements. And $n \mapsto 2^{2n}$ is clearly primitive recursive.

This proves $2^{2n} \rightarrow (n)_2^2$, or, if c is the number of colours, $c^{cn} \rightarrow (n)_c^2$. (As it happens this is not best possible: e.g., we know $6 \rightarrow (3)_2^2$.)

For higher exponents we reason by induction: assume $(\forall n)(\omega \rightarrow (\omega)_n^m)$ and try to prove $\omega \rightarrow (\omega)_n^{m+1}$. The idea (I'm leaving the execution to you!) is to create a tree somewhat in the style we saw above such that, on any branch through

²König's lemma not needed because the graph is countable and therefore wellordered

it, whenever s is subsequence of that path of length m , then all extensions-of- s -by-one-element receive the same colour. You then use Ramsey for exponent m to obtain a monochromatic subset of that branch.

The challenge for the student is to recover a finite version and the computable upper bound, and show that the upper bound is described by a p.r. function.

Indeed this proof will give us a primitive recursive function $f(x, y, z)$ such that $f(x, y, z) \rightarrow (x)_z^y$.

I think it's probably best to write $f_y(x, z) \rightarrow (x)_z^y$ and prove by induction on ' y ' that $\lambda xz.f_y(x, z)$ is primitive recursive. I haven't done it myself.

Exercise 29

(i) is order-preserving, but its only fixed point is the empty function. (iii) is not order-preserving, but has a unique (and obvious) fixed point. Values of the last two operations are always total so the operations can't be continuous!

Exercise 30

(i) For partial functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$, define $d(f, g) = 2^{-n}$ if n is the least number such that $f(n) \neq g(n)$, and $d(f, g) = 0$ if $f = g$. [The inequality $f(n) \neq g(n)$ is understood to include the case where one side is defined and the other is not.] Show that d is a metric, and that it makes $[\mathbb{N} \rightarrow \mathbb{N}]$ into a complete metric space.

(ii) Show that the function $\Phi (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ which corresponds to the recursive definition of the factorial function is a contraction mapping for the metric d , and hence obtain another proof that it has a unique fixed point.

(iii) Which (if any) of the functions defined in Exercise 29 are contraction mappings?

PTJ says: "30(iii) is a contraction mapping, the other two are not. It's worth emphasizing to students who do this question that the contraction-mapping approach to fixed points is special to the case of partial maps defined on \mathbb{N} (note how the well-orderedness of \mathbb{N} is used in the definition of d), but the order-theoretic approach is applicable to an arbitrary $[A \rightarrow B]$. Perhaps also worth pointing out—to brighter students, at least—that contraction mappings don't form a category (identity maps are missing, for the obvious reason that they don't have unique fixed points)."

Exercise 31

Clearly bounded subtraction will be useful here. Does $n \dot{-} m$ represent $n < m$? Almost but not quite, because it isn't two-valued. We need an auxiliary function to zap all non-true (nonzero) truth values to 1.

$$\text{iszero}(0) := 0; \quad \text{iszero}(S(n)) := S(0).$$

`iszero` returns the truth-value of the assertion that its input is 0.

Then $\text{iszero}(n \dot{-} m)$ represents $m < n$ so of course $\text{iszero}(S(n) \dot{-} m)$ represents $m \leq n$ so of course $\text{iszero}(S(n) \dot{-} m) + \text{iszero}(n \dot{-} m)$ represents $m = n$.

Exercise 32

It has to be admitted that division by 2 looks a bit dodgy, but if one treats that rational expression as something that denoted the $(x + y)$ th triangular number it becomes much more sensible. $T(0) = 0$; $T(S(n)) = T(n) + S(n)$ defines triangular numbers. Then $\text{pair}(n, m) = T(n + m) + m$ does the trick.

As for computing the unpairing functions . . .

According to the definition, $\text{pair}(x, y)$ is the sum of a triangular number (in fact the $x + y$ th triangular number) and a remainder— x —that is less than the difference— $x + y + 1$ —to the next triangular number. Thus the $x + y$ th triangular number is the largest triangular number $\leq \text{pair}(x, y)$. Thus pair is injective. To decode a number z as $\text{pair}(x, y)$, first ascertain the largest triangular number $\leq z$; this gives you the $x + y$; the remainder after subtracting the triangular number is x ; subtract the first component from the remainder to recover y . Thus pair is surjective.

$$\begin{aligned} \text{fst}(x) & \text{ is } \sum_{i < x} (\text{if } (\exists z < x)(\text{pair}(i, z) = x) \text{ then } i \text{ else } 0); \\ \text{snd}(x) & \text{ is } \sum_{i < x} (\text{if } (\exists z < x)(\text{pair}(z, i) = x) \text{ then } i \text{ else } 0). \end{aligned}$$

I imagine that this kind of counting construction will help us show that Euler's totient function is primitive recursive.

Exercise 33

“Prove that ϕ is primitive recursive.”

We will need some auxilliary functions/relations. The relation “ n divides m ” is primitive recursive, since it is $(\exists k < n)(k \cdot m = n)$. So “ n is prime” is a primitive recursive predicate. “ m and n are coprime” is a primitive recursive predicate, being the negation of “ $(\exists k < m)(k|m \wedge k|n \wedge S(0) < k)$ ”. We can now obtain $\phi(n)$ by bounded summation thus

$$\sum_{m < n} \text{if } m \text{ and } n \text{ are coprime then } 1 \text{ else } 0.$$

Exercise 35

35 part 1

“Find a primitive recursive declaration for the function commonly declared by

$$f(0) := f(1) := 1; \quad f(n + 1) := f(n) + f(n - 1).”$$

Declare $F(0) := \langle 1, 1 \rangle$; $F(n+1) = \langle \text{snd}(F(n)), \text{fst}(F(n)) + \text{snd}(F(n)) \rangle$.
Then $\text{Fib}(n) = \text{fst}(F(n))$.

If you program **Fibonacci** in the obvious way you get exponential blowup: by making two calls at each stage you end up with 2^n calls to **Fib**(0) when computing **Fib**(n). If you program it in the primitive recursive way you end up with only one call to **Fib**(0).

This technique is commonly called **pipelining**.

35 part 2

We want to represent H as something obtained by iteration. The function we are going to define by iteration will be $\lambda n. \langle H(n), n \rangle$ (though of course that is not how it is *explicitly* defined!), and then we get H from it by composition with **fst**. Abbreviate $\lambda n. \langle H(n), n \rangle$ to F . Then we have

$$\begin{aligned} F(S(y)) &= \langle H(S(y)), S(y) \rangle \\ &= \langle G(H(y), y), S(y) \rangle \end{aligned}$$

(we know this by the recursion). Now $H(y) = \text{fst}(F(y))$ and $y = \text{snd}(F(y))$, so this is

$$\langle G(\text{fst}(F(y)), \text{snd}(F(y))), S(\text{snd}(F(y))) \rangle,$$

and we notice that all occurrences of ‘ y ’ are wrapped up in F ’s, so this is

$$f(F(y)),$$

where f is

$$\lambda z. \langle G(\text{fst}(z), \text{snd}(z)), S(\text{snd}(z)) \rangle,$$

so $H(y) = \text{fst}(Fy) = \text{fst}(f^y(F0)) = \text{fst}(f^y(b))$, where $b = \langle a, 0 \rangle$.