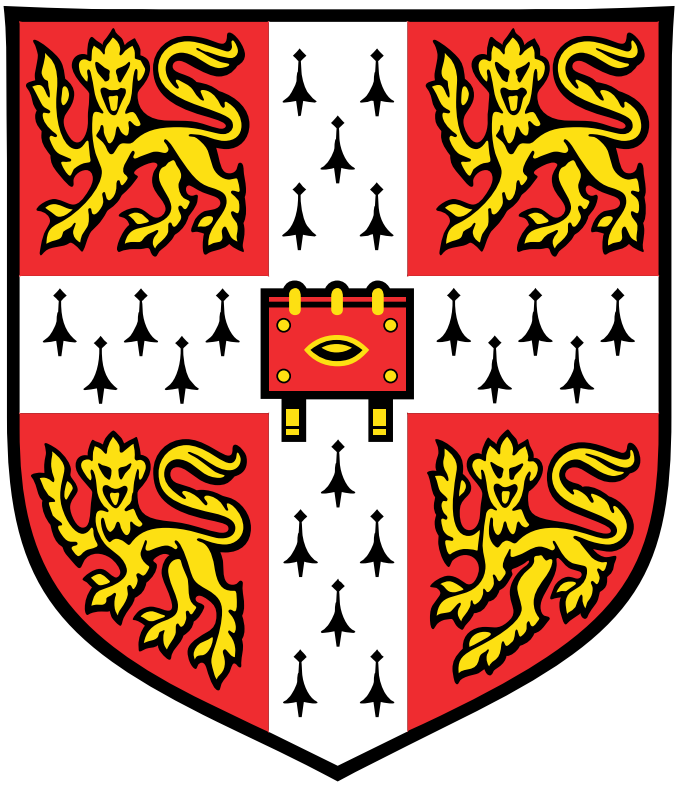
Part III Computability and Logic: 24 Lectures in 2020/21

Thomas Forster

October 10, 2020



Contents

Dear Reader,

Please do **not** download this document. I update it and reload it all the time as errors and infelicities come to light, and students make helpful remarks—or prompt me to make helpful remarks—which need to be incorporated. It's probably a good idea to *bookmark* it.

This file contains more stuff than I am going to be able to lecture, but all of it is good for your soul. At some point in the lent term i will make an official pronouncement on what has been lectured and what i consider examinable. This doesn't mean that everything that has been pronounced examinable will actually be examined (there aren't enough questions available for that!); what it means is that you will not suffer in the exam by skipping material that has not been declared examinable. I think that, nearer the time, the stuff that i consider examinable will be printed in red.

At this stage this text contains lots of *messages to myself*; but they are my *notes-towards-a-lecture-course* and are emphatically not tested-on-animals *materials* for you; while you are of course welcome to eavesdrop, these notes come with no guarantees.

0.1 Revision material

Mostly what follows in this section is Part II Set Theory and Logic. Very little of the material traditionally covered that course genuinely essential for Part III Computability and Logic. Really only the fixed point theorems (and Dr Russell read the small print of the schedules and concluded that he wasn't required to lecture Tarski-Knaster, so he didn't. However tyou need to know it). Altho' the Part II material is not really essential, finding it scary is a predictor of a bumpy time ahead, so it will do you no harm to skim through it, and take steps should you feel scared. Much of it is genuinely useful for the Part III *Set theory* course, and many of these exercises (dating from when i lectured Part II ST&L in 2016) will be helpful. Sheet 5 was designed as an extra sheet for people who were thinking of pursuing that material in Part III, tho' it has to be said it was with the Part III *Set Theory course* in mind. If you are doing Part III Computability and Logic you are quite likely to be doing Set Theory as well, so tuck in.

My lecture notes from 2016 are linked here `www.dpmms.cam.ac.uk/~tf/cam_only/partiilectures2016.pdf`

0.2 Example Sheets from Part II

Questions marked with a '+' are brief reality-checks; questions marked with a '*' are for enthusiasts/masochists only; ☠ means what you think it means, and particularly tasty questions are decorated with a pink marzipan pig: 

Sheet 0: Numbers and Sets Revision

1A Numbers and Sets is the only prerequisite for Part II Set Theory and Logic, and—even if you are a Part III student not a Part II student—it can do you no harm to give a quick going-over to your notes for that course. You might like to have a quick glance at my supervision/lecture notes for Discrete maths for Computer Scientists, linked from my 1a teaching page. Those notes dwell on *Sets* rather than *Numbers* but that's OK beco's there is no number theory in Part II ST&L. And none to speak of in Part III Computability and Logic.

Countability

"uncountably many" wasn't ever a complete answer to the question "How many wombats are there?" It *just may* (sometimes) still be an *adequate* answer but—now that you are doing Part II you should always be prepared to give more

detail. Read `www.dpmms.cam.ac.uk/~tf/countability.pdf` and do the exercises therein; it won't take you long.

(i)

Explain briefly why the diagonal argument that shows that $\mathcal{P}(\mathbb{N})$ is uncountable doesn't show that there are uncountably many finite sets of naturals.

Set Theory and Logic, Michaelmas 2016, Sheet 1: Ordinals and Induction

Questions marked with a '*' may be skipped by the nervous.

(i)

Write down subsets of $\mathbb{R}$ of order types $\omega + \omega$, ω^2 and ω^3 in the inherited order.

(ii)

Which of the following are true?

(a) $\alpha + \beta$ is a limit ordinal iff β is a limit ordinal;
(b) $\alpha \cdot \beta$ is a limit ordinal iff α or β is a limit ordinal;
(c) Every limit ordinal is of the form $\alpha \cdot \omega$;
(d) Every limit ordinal is of the form $\omega \cdot \alpha$.

For these purposes 0 is a limit ordinal.

(iii)

Consider the two functions $On \to On$: $\alpha \mapsto 2^\alpha$ and $\alpha \mapsto \alpha^2$. Are they normal?

(iv)

Prove that $\langle X, <_X \rangle$ is a total order satisfying "every subordering is isomorphic to an initial segment" iff it is a wellordering.

(v)

What is the smallest ordinal you can not embed in the reals in the style of question (i)?

(vi)

Prove that every [nonzero] countable limit ordinal has cofinality ω. What about ω_1?

(vii)*

Recall the recursive definition of ordinal exponentiation:

$$\alpha^0 = 1;\ \alpha^{\beta+1} = \alpha^\beta \cdot \alpha, \text{ and } \alpha^{sup(B)} = \sup(\{\alpha^\beta : \beta \in B\}).$$

Ordinal addition corresponds to disjoint union [of wellorderings], ordinal multiplication correponds to lexicographic product, and ordinal exponentiation corresponds to . . . ? Give a definition of a suitable operation on wellorderings and show that your definition conforms to the spec: $\alpha^{\beta+\gamma} = \alpha^\beta \cdot \alpha^\gamma$.

(viii)

Let $\{X_i : i \in I\}$ be a family of sets, and Y a set. For each $i \in I$ there is an injection $X_i \hookrightarrow Y$. Give an example to show that there need not be an injection $(\bigcup_{i\in I} X_i) \hookrightarrow Y$. But what if the X_i are nested? [That is, $(\forall i, j \in I)(X_i \subseteq X_j \vee X_j \subseteq X_i)$.]

(ix)

Prove that every ordinal of the form ω^α is **indecomposible**: $\gamma + \beta = \omega^\alpha \ \rightarrow\ \gamma = \omega^\alpha \ \vee\ \beta = \omega^\alpha$.

(x)

Show that an arbitrary intersection of transitive relations is transitive. The **transitive closure** R^* (sometimes written '$tr(R)$') is the $\subseteq$-least transitive relation $\supseteq R$.

Let $\langle X, R\rangle$ be a wellfounded binary structure, with rank function ρ. Prove that $(\forall x \in X)(\forall \alpha < \rho(x))(\exists y)(\rho(y) = \alpha)$.

[A later—perhaps preferable—version of this question. . .

Let $\langle X, R\rangle$ be a wellfounded binary structure, with rank function ρ. Prove that $(\forall x \in X)(\forall \alpha < \rho(x))(\exists y \in X)(\rho(y) = \alpha)$.]

(xi)

Let $\{X_i : i \in \mathbb{N}\}$ be a nested family of sets of ordinals.

(a) Give an example to show that the order type of $\bigcup_{i\in\mathbb{N}} X_i$ need not be the sup of the order types of the X_i.
(b) What condition do you need to put on the inclusion relation between the X_i to ensure that the order type of $\bigcup_{i\in\mathbb{N}} X_i$ is the sup of the order types of the X_i?
(c) Show that the ordered set of the rationals can be obtained as the union of a suitably chosen ω-chain of some of its finite subsets.

(xii)

Using the uniqueness of subtraction for ordinals, and the division algorithm for normal functions, show that every ordinal can be expressed uniquely as a sum

$$\omega^{\alpha_1} \cdot a_1 + \omega^{\alpha_2} \cdot a_2 + \cdots \omega^{\alpha_n} \cdot a_n$$

where all the a_i are finite, and where the α_i are strictly decreasing.

(xiii)

Let f be a function from countable [nonzero] limit ordinals to countable ordinals satisfying $f(\alpha) < \alpha$ for all (countable limit) α. (f is "*pressing-down*".) Can f be injective?

Set Theory and Logic, Michaelmas 2016, Sheet 2: Posets

'+' signifies a question you shouldn't have trouble with; '☠' means what you think it means.

(i)

(a) For $n \in \mathbb{N}$, how many antisymmetrical binary relations are there on a set of cardinality n? How many binary relations satisfying *trichotomy*: $(\forall xy)(R(x,y) \vee R(y,x) \vee x = y)$? How are your two answers related?

(b) How many *symmetric* relations and how many *antisymmetric trichotomous* relations are there on a set of cardinality n? How are your two answers related?

(c) Contrast (a) and (b)

(ii)

Consider the set of equivalence relations on a fixed set, partially ordered by $\subseteq$. Show that it is a lattice. Must it be distributive? Is it complete?

(iii)

Cardinals: Recall that $\alpha \cdot \beta$ is $|A \times B|$ where $|A| = \alpha$ and $|B| = \beta$. Show that a union of α disjoint sets each of size β has size $\alpha \cdot \beta$. Explain your use of AC.

(iv)

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.

(v)

(Mathematics Tripos Part II 2001:B2:11b, modified).

Let U be an arbitrary set and $\mathcal{P}(U)$ be the power set of U. For X a subset of $\mathcal{P}(U)$, the **dual** $X^\vee$ of X is the set $\{y \subseteq U : (\forall x \in X)(y \cap x \neq \emptyset)\}$.

1. Is the function $X \mapsto X^\vee$ monotone? Comment.
2. By considering the poset of those subsets of $\mathcal{P}(U)$ that are subsets of their duals, or otherwise, show that there are sets $X \subseteq \mathcal{P}(U)$ with $X = X^\vee$.
3. $X^{\vee\vee}$ is clearly a superset of X, in that it contains every superset of every member of X. What about the reverse inclusion? That is, do we have $Y \in X^{\vee\vee} \to (\exists Z \in X)(Z \subseteq Y)$?
4. Is $A^{\vee\vee\vee}$ always equal to $A^\vee$?

(vi)

Use Zorn's Lemma to prove that

(i) every partial ordering on a set X can be extended to a total ordering of X;

(ii) for any two sets A and B, there exists either an injection $A \hookrightarrow B$ or an injection $B \hookrightarrow A$.

(vii)

(Tripos IIA 1998 p 10 q 7)

Let $\langle P, \leq_P \rangle$ be a chain-complete poset with a least element, and $f : P \to P$ an order-preserving map. Show that the set of fixed points of f has a least element and is chain-complete in the ordering it inherits from P. Deduce that if $f_1, f_2, \ldots, f_n$ are order-preserving maps $P \to P$ which commute with each other (i.e. $f_i \circ f_j = f_j \circ f_i$ for all i, j), then they have a common fixed point. Show by an example that two order-preserving maps $P \to P$ which do not commute with each other need not have a common fixed point.

(viii)

$\mathbb{N} \rightharpoonup \mathbb{N}$ is the set of partial functions from $\mathbb{N}$ to $\mathbb{N}$, thought of as sets of ordered pairs and partially ordered by $\subseteq$.

Is it complete? Directed-complete? Separative? Which fixed point theorems are applicable?

For each of the following functions $\Phi : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$, determine (a) whether Φ is order-preserving, and (b) whether 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.

(ix)

Players `I` and `II` alternately pick elements (`I` plays first) from a set A (repetitions allowed: A does not get used up) thereby jointly constructing an element s of A^ω, the set of ω-sequences from A. Every subset $X \subseteq A^\omega$ defines a game $G(X)$ which is won by player `I` if $s \in X$ and by `II` otherwise. Give A the discrete topology and A^ω the product topology.

By considering the poset of partial functions $A^{<\omega} \to \{\mathtt{I}\}$ ($A^{<\omega}$ is the set of finite sequences from A) or otherwise prove that if X is open then one of the two players must have a winning strategy.

(x) 

$\mathbb{R} = \langle 0, 1, +, \times, \leq \rangle$ is a field. Consider the product $\mathbb{R}^{\mathbb{N}}$ of countably many copies thereof, with operations defined pointwise. Let $\mathcal{U}$ be an ultrafilter $\subseteq \mathcal{P}(\mathbb{N})$ and consider $\mathbb{R}^{\mathbb{N}}/\mathcal{U}$. Prove that it is a field. Is it archimedean?

(xi)

(i)$^+$ How many order-preserving injections $\mathbb{R} \to \mathbb{R}$ are there?

(ii)☠ Let $\langle X, \leq_X \rangle$ be a total order with no nontrivial order-preserving injection $X \to X$. Must X be finite?

Set Theory and Logic, Michaelmas 2016, Sheet 3: Propositional and Predicate Logic

(i)

Show how $\wedge$, $\vee$ and $\neg$ can each be defined in terms of $\rightarrow$ and $\bot$. Why can you not define $\wedge$ in terms of $\vee$? Can you define $\vee$ in terms of $\rightarrow$? Can you define $\wedge$ in terms of $\rightarrow$ and $\vee$?

(ii)

(a) Show that for every countable set A of propositions there is an independent set B of propositions with the same deductive consequences.

(b) If A is finite show that we can find such a B with $B \subseteq A$.

(c) Give an example to show that we should not expect $B \subseteq A$ if A is infinite.

(d) Show that if A is an infinite independent set of propositions then there is no finite set with the same deductive consequences.

(iii)

Explain briefly the relation between truth-tables and Disjunctive Normal Form.

Explain briefly why every propositional formula is equivalent both to a formula in CNF and to a formula in DNF.

Establish that the class of all propositional tautologies is the maximal propositional logic in the sense that any superset of it that is a propositional logic (closed under $\models$ and substitution) is trivial (contains all well-formed formulæ).

(iv)

A formula (of first-order Logic) is in **Prenex Normal Form** if the quantifiers have been "pulled to the front"—every propositional connective and every atomic subformula is within the scope of every quantifier.

> Explain briefly why every first-order formula is equivalent to one in PNF.
>
> Axiomatise the theory of groups in a signature with '=' and a single three-place relation "x times y is z". Put your axioms into PNF. What are the quantifier prefixes?
>
> Find a signature for Group Theory which ensures that every substructure of a group is a semigroup-with-**1**.

(v)

Show that the theory of equality plus one wellfounded relation is not axiomatisable.

(vi)

Write down axioms for a first-order theory T with equality plus a single one-place function symbol f that says that f is bijective and that for no n and no x do we have $f^n(x) = x$.

(a) Is T finitely axiomatisable?

(b) How many countable models does T have (up to isomorphism)?

(c) How many models of cardinality of the continuum does it have (up to isomorphism)? (You may assume that the continuum is not the union of fewer than $2^{\aleph_0}$ countable sets, a fact whose proof—were you to attempt it—would need AC.)

(d) Let κ be an uncountable aleph. How many models does T have of size κ?

(e) Is T complete?

(vii)

Show that monadic predicate logic (one place predicate letters only, without equality and no function symbols) is decidable.

(viii) 

(a)$^+$ Suppose A is a propositional formula and 'p' is a letter appearing in A. Explain how to find formulæ A_1 and A_2 not containing 'p' such that A is logically equivalent to $(A_1 \wedge p) \vee (A_2 \wedge \neg p)$.

(b) Hence or otherwise establish that, for any two propositional formulæ A and B with $A \models B$, there is a formula C, containing only those propositional letters common to both A and B, such that $A \models C$ and $C \models B$. (Hint: for the base case of the induction on the size of the common vocabulary you will need to think about expressions over the empty vocabulary).

(ix)

Why does T not follow from K and S?

Show that Peirce's Law: $((A \to B) \to A) \to A$ cannot be deduced from K and S.

(x$^+$)

Look up *monophyletic*. Using only the auxiliary relation "is descended from" give a definition in first-order logic of what it is for a set of lifeforms to be monophyletic.

(xi)

Is

$$(\forall x)(\exists y)(F(x,y)) \to (\forall x)(\exists y)(\forall x')(\exists y')[F(x,y) \wedge F(x',y') \wedge (x = x' \to y = y')]$$

valid?

(xii)

(a) Show that the theory of fields of characteristic zero is (first-order) axiomatisable but not finitely axiomatisable. Show that the theory of fields of finite characteristic is not first-order axiomatisable.

(b) Recall that a simple group is one with no nontrivial normal subgroup. Is the theory of simple groups first order?

(c) A local ring is a ring with a unique maximal ideal. Is the theory of local rings first-order? [Hint: what might the unique maximal ideal be?]

(d) Is the theory of posets in which every element belongs to a unique maximal antichain first-order?

(e) A theory T is **equational** iff every axiom of T is of the form $(\forall \vec{x})\Phi$ where ϕ is a conjunction of equations between T-terms.

Prove that, if T is equational, then a pointwise product of models of T is another model of T, and substructures and homomorphic images of models of T are models of T.

Which of the theories in (a)–(d) are equational?

(xiii) 

A *type* in a propositional language $\mathcal{L}$ is a countably infinite set of formulæ.

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 satisfies 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 $\phi \to \sigma$ for every $\sigma \in \Sigma$, then $T \vdash \neg\phi$.

(a) Prove the following:

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.

(b) Prove the following

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.

(xiv)

Prove that, for every formula ϕ in CNF, there is a formula ϕ' which
(i) is satisfiable iff ϕ is;
(ii) is in CNF where every conjunct contains at most three disjuncts.
(Hint: there is no assumption that $\mathcal{L}(\phi') = \mathcal{L}(\phi)$.)

Set Theory and Logic, Michaelmas 2016, Sheet 4: More Predicate Logic and Some Set Theory

(i)$^+$

Show that if x is a transitive set, then so are $\bigcup x$ and $\mathcal{P}(x)$. Are the converses true?

(ii)$^+$

Show that the Pair-set axiom is deducible from the axioms of empty set, power set, and replacement.

(iii)$^+$

Show that $\{z : \neg(\exists u_1, \ldots, u_n)((z \in u_1) \wedge (u_1 \in u_2) \wedge \cdots \wedge (u_n \in z))\}$ is not a set for any n. What assumptions have you made?

(iv)

Write down sentences in the language of set theory to express the assertions that, for any two sets x and y, the product $x \times y$ and the set y^x of all functions from x to y exist. You may assume that your pairs are Wiener-Kuratowski.

Which axioms of set theory are you going to have to assume if these assertions are to be provable?

(v)

(a) Prove that every normal function $On \to On$ has a fixed point.
(b) Prove that the function enumerating the fixed points of a normal function $On \to On$ is itself normal.
(c) If α is a regular ordinal and f is a normal function show that f has a fixed point of cofinality α.
(d) Are any of your fixed points regular?

(vi)

Show that the axiom of choice follows from the assumption that cardinals are totally ordered by $\leq_{card}$.

(vii)

Explain briefly the equivalence of the four versions of the axiom of foundation given in lectures: (i) The axiom scheme of $\in$-induction; (ii) The assertion that every set is wellfounded; (iii) Axiom of Regularity; (iv) Every set belongs to the cumulative hierarchy.

(viii)

f is an $\in$-automorphism if f is a permutation of V that preserves $\in$: $x \in y \longleftrightarrow f(x) \in f(y)$. Show that a model of ZF (with foundation of course) can have no nontrivial $\in$-automorphisms.

Give an example to show that the surjectivity condition on f is necessary; that is to say, there are non-trivial injective $\in$-homomorphisms.

(ix)

For the Wiener-Kuratowski ordered pair $\rho(\langle x, y\rangle) = \max(\rho(x), \rho(y)) + 2$. ($\rho$ is set-theoretic rank.)

(a) Can you define a ordered pair such that $\rho(\langle x, y\rangle) = \max(\rho(x), \rho(y)) - 1$?
(b) Can you define a ordered pair such that $\rho(\langle x, y\rangle) = \max(\rho(x), \rho(y)) + 1$?
(c)* Can you define a ordered pair such that $\rho(\langle x, y\rangle) = \max(\rho(x), \rho(y))$ for all but finitely many x and y?

(x)

There are various ways of constructing implementations (as sets) of $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{C}$ from an implementation (as sets) of the naturals. For one of these constructions compute the ranks of the sets that have the rôles of $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{C}$.

Different implementations will almost certainly give you different answers. Are there any lower or upper bounds on the answers you might get?

(xi)

Consider the binary relation E on $\mathbb{N}$ defined by: $n\,E\,m$ iff the nth bit (counting from the right, starting at 0) in the binary expansion of m is 1. What can you say about the structure $\langle \mathbb{N}, E\rangle$?

(xii)

Prove that, for each $n \in \mathbb{N}$, there is a set of size $\aleph_n$. Is there a set of size $\aleph_\omega$?

(xiii)

Assume that the cartesian product $x \times y$ always exists however you implement ordered pairs. Infer the axiom scheme of replacement.

(xiv)

Assume that every normal function $On \to On$ has a regular fixed point. Consider the function that enumerates the initial ordinals and deduce that there is a "weak inaccessible" κ. Which axioms of ZF hold in V_κ?

(xv)

Suppose $\{A_i : i \in I\}$ and $\{B_i : i \in I\}$ are families of sets such that for no $i \in I$ is there is a surjection $A_i \twoheadrightarrow B_i$. Show that there is no surjection $\bigcup_{i\in I} A_i \twoheadrightarrow \prod_{i\in I} B_i$.

You will need the axiom of choice. Is there a converse?

Using these ideas you can show that $\aleph_\omega \neq 2^{\aleph_0}$ *without* using AC.

Sheets from here on are still under construction!!

Set Theory and Logic, Michaelmas 2016, Sheet 5

Sheet 5 is for Part II enthusiasts who want to take this stuff further; it's a mixture of revision, consolidation and looking-ahead. It is also for Part III students who want something to get them used to what they are going to be doing later in the year. (Part III Logic is lectured in Lent.)

(i)

Explain to your supervision partner (or to anyone listening who might be confused) the difference between

(i) Nonstandard naturals
(ii) Countable ordinals
(iii) Infinite Dedekind-finite cardinals

(ii)

For P a poset, let P^* be the poset of chains-in-P partially ordered by end-extension. (Chains are allowed to be empty). Show that there is no injective homomorphism $P^* \hookrightarrow P$.

(iii)

Any two countable dense linear orders without endpoints are isomorphic. Give an illustration to show how your back-and-forth construction might not work for dense linear orders of size $\aleph_1$. How do you have to spice up the denseness condition to prove an analogous result for linear orders of size $\aleph_1$?

(iv)

(For those of you who did Languages and Automata)

A wellordering of $\mathbb{N}$ is *recursive* iff its graph (subset of $\mathbb{N} \times \mathbb{N}$) is decidable ("recursive"); an ordinal is *recursive* iff it is the order type of a decidable ("recursive") wellordering of $\mathbb{N}$. Which of the countable ordinals you have learned to know and love are recursive? Come to think of it, are *all* countable ordinals recursive?

(v)*

(For those of you who did Languages and Automata)

Prove Trakhtenbrot's theorem that if S is a signature with equality and at least one binary relation symbol then the set of S-sentences true in all finite structures is not semidecidable ("r.e.").

(vi)

(A taster for *forcing*)

A poset $\langle P, \leq_P \rangle$ is [upwards] separative if $(\forall x, y \in P)(x \not\leq y \rightarrow (\exists z \geq y)(\forall w)(w \not\geq z \vee w \not\geq x))$

For each of the following posets say whether or not it is (i) separative (ii) directed (iii) chain-complete.

The set of finite sequences of countable ordinals (thought of as sets of ordered pairs) partially ordered by $\subseteq$.

The set $\{f : f$ is an injection from some set of countable ordinals $\hookrightarrow \mathbb{R}\}$ ordered by $\subseteq$. Think of f as a set of ordered pairs.

(vii)

(For those of you who did some graph theory in Lent term)

Using propositional logic only, show that a(n undirected) graph and its complement cannot both be disconnected. (Hint: propositional letters will correspond to edges)

(viii)

A poset $\langle P, \leq \rangle$ is called *downwards separative* if for all $x \not\leq y$ there is $z \leq x$ with z incompatible with y. ("incompatible" means "have no lower bound"). We say that a poset is *downwards splitting* if for every x there are y and z such that $y, z \leq x$, and y and z are incompatible.

(a) Show that not every downwards separative poset is downwards splitting.

(b) Show that if a poset has no minimal elements and is downwards separative, then it is downwards splitting.

A set $D \subseteq P$ is called *downwards dense* if for every p in P there is a d in D such that $d \leq p$.

Suppose XX is a collection of subsets of P. We say that $G \subseteq P$ is XX-*generic* if G has nonempty intersection with every downwards dense element of XX.

We say that G is a *filter* if

1. for any x, y in G there is z in G such that $z \leq x$ and $z \leq y$, and
2. for any x in G and $x \leq y$, we have y in G.

(c) If XX is countable, show that there is an XX-generic filter.

(d) Let $\langle P, \leq \rangle$ be a downwards separative poset with no minimal elements and let XX be a collection of subsets of P closed under complementation (i.e., if $X \in XX$, then also $P \setminus X \in XX$). Show that if G is an XX-generic filter, then $G \notin XX$.

(e) Let $\langle P, \leq \rangle$ be the set of finite sequences of zeros and ones, ordered by reverse inclusion. Show that this is a downwards separative poset without minimal elements.

(f) Let XX be the collection of recursive sets of finite sequences of zeros and ones. Show, using (c), (d), and (e), that there is a non-recursive such set.

(ix)

(Concrete constructions of limits in ZF)

Let $\langle I, \leq_I \rangle$ be a directed poset and, for each $i \in I$, let A_i be a set and, for all $i \leq_I j$, let $\sigma_{i,j} : A_i \hookrightarrow A_j$ be an injection, and let the injections commute.

Show that there is a set A_I with, for each $i \in I$, an injection $\sigma_i : A_i \hookrightarrow A_I$ and the $\sigma_{i,j}$ commute with the σ_i.

Show also that A_I is minimal in the sense that if B is any set such that for each $i \in I$ there is an injection $\tau_i : A_i \hookrightarrow B$ and the τ_i commute with the $\sigma_{i,j}$, then there is a map $A_I \hookrightarrow B$.

Let $\langle I, \leq_I \rangle$ be a directed poset and, for each $i \in I$, let A_i be a set and, for all $i \leq_I j$, let $\sigma_{j,i} : A_j \twoheadrightarrow A_i$ be a surjection, and let the surjections commute.

Show that there is a set A_I with, for each $i \in I$, a surjection $\pi_i : A_I \twoheadrightarrow A_i$.

Show also that A_I is minimal in the sense that, if B is any set such that for each $i \in I$ there is a surjection $\tau_i : B \twoheadrightarrow A_i$ and the τ_i commute with the $\sigma_{i,j}$, then there is a map $B \twoheadrightarrow A_I$.

(x) ☠

Let G be the alternating group of permutations of V_ω. For each $n \in \mathbb{N}$ its members can move x by permuting those elements of $\bigcup^n x$ that are of finite rank and fixing the remainder. A set that is fixed by everything in G under the nth action of G is said to be n-**symmetric**; if it is n-symmetric for all sufficiently large n it is just plain **symmetric**.

Consider the collection of sets that are hereditarily symmetric. Which axioms of ZFC are true in this structure?

(xi)* (A taster for Large Cardinals)

Prove Łoś's theorem :

THEOREM 1 *Let $\mathcal{U}$ be an ultrafilter $\subseteq \mathcal{P}(I)$. For all first-order expressions ϕ,*

$$(\prod_{i\in I} \mathcal{A}_i)/\mathcal{U} \models \phi \text{ iff } \{i : \mathcal{A}_i \models \phi\} \in \mathcal{U}.$$

(You may assume AC)

Suppose there is a set K with a nonprincipal ultrafilter $\mathcal{U} \subseteq \mathcal{P}(K)$ that is closed under countable intersections. By using Scott's trick concretise the elements of the ultrapower $V^K/\mathcal{U}$. Prove that it is wellfounded. What can you say about the Mostowski collapse?

(xii)*

$\mathcal{I} = \langle I, \leq_{\mathcal{I}} \rangle$ is a **set of indiscernibles** for a model $\mathfrak{M}$ for a language $\mathcal{L}$ iff $\leq_{\mathcal{I}}$ is a total order, and for all $\phi \in \mathcal{L}$, if ϕ is a formula with n free variables in it then for all distinct n-tuples $\vec{x}$ and $\vec{y}$ from $\mathcal{I}$ **taken in $\leq_{\mathcal{I}}$-increasing order** we have $\mathfrak{M} \models \phi(\vec{x}) \longleftrightarrow \phi(\vec{y})$.

Now let $\mathcal{I}$ be a total order, T a theory with infinite models and a formula $P()$ with one free variable s.t. T thinks that the extension of P is an infinite total order. Then T has a model $\mathfrak{M}$ in which $\mathcal{I}$ is embedded in (the interpretation of) P as a set of indiscernibles.

(Notice that there is no suggestion that the copy of $\mathcal{I}$ in $\mathfrak{M}$ is a set of $\mathfrak{M}$, or is in any way definable.)

It is comparatively straightforward, given $\mathcal{I}$ and T and $P()$, to find $\mathfrak{M}$ as in the theorem if we do not ask that I should be embedded as a set of indiscernibles: compactness does the trick. To get the set of indiscernibles you need to use Ramsey's theorem from Graph theory.

(xiii)

(GRM revision from a logical point of view). Wikipædia says:

> Commutative Rings $\supseteq$ Integral Domains $\supseteq$ Integrally Closed Domains $\supseteq$ GCD domains $\supseteq$ Unique Factorization Domains $\supseteq$ Principal Ideal Domains $\supseteq$ Euclidean Domains $\supseteq$ Fields

All these families-of-structures can be thought of as belonging to the one signature: 0, $\mathbb{1}$, $+$, $\cdot$ and $-$. Which of them are first-order axiomatisable? In each case provide axiomatisations or explain why there are none. Identify the quantifier complexity of the axiomatisations you find.

(xiv)(☠)

How many countable [linear] order types are there whose automorphism group is transitive on singletons?

(xv)

How many transitive subsets of V_ω are there?

How many transitive sets are there all of whose members are countable?

(xvi)

Recall the difference between a **wellorderable** set and a **wellordered** set.

Prove without using AC or foundation or ordinals that every set of wellorderable sets has a member that injects into all the others.

Is this the same as saying that the collection of alephs is wellordered by the order relation on cardinals?

(xvii)

A directed limit of wellfounded structures under end-extension is wellfounded.

Languages and Automata Extra Sheet

Recall Cantor Normal Forms for ordinals below ϵ_0. Show that if we don't care about writing the terms in decreasing order then the language is context-free, but that if we write them out properly, with terms in decreasing order, then it is not context-free.

Chapter 1

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 $\mathbb{R} \to \mathbb{R}$ 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 gnumbered[1] and we can contemplate which of the manipulations of the finite objects it characterises correspond to the computable manipulations of the gumbers 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:

[1] Gödel-numbering; the 'g' is silent.

movements are controlled *locally* by a nextwork 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 (remember from last year a Part II ST&L question about the theory of simple groups of order 60) a presentation of a group does not straightforwardly give rise to a categorical[2] first-order theory that characterises it. You cannot compute the first-order theory of a group from a presentation of it. Burnside groups?

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 $s$low and $r$apid respectively) One can think of the two sets of s-process and r-process nuclei as inductively defined sets as follows.[3]

• 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

[2] A Categorical theory is a theory with a unique model (up to isomorphism).

[3] One 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.

You probably learnt in Part II how to use continued fractions to find all solutions to Pell's 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 computablity 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 5.

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".[4]

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 *Logic and Set Theory* 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.

This syntactic/semantic distinction is not the same as the function-in-intension/function-in-extension distinction of course

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

1.1 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 pigs are made of pink icing sugar). 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: $f``\mathbb{N} := \{f(n) : n \in \mathbb{N}\}$

$\mathrm{Dom}(f) := \{n \in \mathbb{N} : f(n)\downarrow\}$ (see p. 91)

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

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

[4] Philosophically inclined readers may wish to reflect on the curious fact that Church's 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 2

Induction, Wellfoundedness and Recursion in a General Context

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

Kierkegaard

2.1 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

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

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 $A \subseteq dom(R)$. 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 $dom(R()$ 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 that 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

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

which is

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

which is

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

R'-induction allows you to infer $\phi(x)$ as long as everything that bears R' to x has ϕ. R-induction similarly. If $R' \subset R$ then there are fewer things that bear R' to x than there are that bear R, so you are inferring $\phi(x)$ on the basis of less information; a stronger inference.

This explains why wellfounded induction over longer wellorderings is stronger than wellfounded induction over shorter wellorderings. This matters beco's, typically, (the graph of) a wellordering of $\mathbb{N}$ to length α (where $\alpha > \beta$) is a proper subset of (the graph of) a wellordering of $\mathbb{N}$ to length β.

Exercise 1 *(*)*[1]

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

Provide a sequent calculus or natural deduction proof that

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

and

$$(\forall xy)(R'(x,y) \to R(x,y))$$

together imply

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

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

Exercise 2 *Let $\langle A, R\rangle$ and $\langle B, S\rangle$ be wellfounded binary structures.*

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

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

(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 $A \to B$ that take only finitely many nonzero values is wellfounded.

Theorem 2 *The Recursion Theorem*
If $\langle X, R\rangle$ is a wellfounded structure and $G : X \times \mathcal{P}(V) \to V$ 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 $G : X \times V \to V$, 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 3 *Prove theorem 2*

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. ■

2.2 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 Nfinite; if X is Nfinite and $y \notin X$ then $X \cup \{y\}$ is Nfinite[2].

• The empty set is hereditarily finite[3]; if x and y are hereditarily finite so is $x \cup \{y\}$.

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

- 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.

You might benefit from looking at exercises 6 and 10 on PTJ's Part II Set Theory and Logic sheet 3 2012/3 on `https://www.dpmms.cam.ac.uk/study/II/Logic/2012-2013/LSqns3.pdf` There is a discussion of them in the materials on my Part II Materials page (probably not linked at the moment

All these rectypes are *of finite character*: the operations that construct them are finitary. [Is $\mathbb{N}$ the terminal object in the category of rectypes of finite character...? Only if the maps are parsimonious]

The collection HC (aka $H_{\aleph_1}$) of hereditarily countable sets is a rectype of infinite (but bounded) character. The constructor ("grab countably many sets") is not of finite character. There is an old tripos question on it wot i set years ago `https://www.maths.cam.ac.uk/undergrad/pastpapers/files/2017/list_ii_1.pdf` 15H p 61

[2]This property is also called 'cardinal-finite'. There doesn't seem to be a standard notation for it.

[3]*vide* old Set Theory and Logic example sheets. See q 8 on `https://www.dpmms.cam.ac.uk/study/II/Logic/2019-2020/20sheet4.pdf`

2.3 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 thing[4], 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 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.[5]

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

$$\mathbb{N}(0); \quad (\forall x)(\mathbb{N}(x) \to \mathbb{N}(S(x)))$$

We declare the datatype of α-lists by

$$\alpha\texttt{-list}(\texttt{null}); \ (\forall l)((\alpha\texttt{-list}(l) \wedge \alpha(x)) \to \alpha\texttt{-list}(\texttt{cons}(x, l)))$$

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 course[6].

[4] That is the order extension principle, a consequence of Zorn's lemma.

[5] You 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.

[6] There 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. You can see results like this in a model theory course.

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 8.1.1.

2.4 Structural Induction

Recursive datatypes support **Structural Induction** ("ancestral induction" in Russell-and-Whitehead. The terminology 'ancestral' for what is nowadays often called the *transitive closure* (of a relation (not a set—that's something different!) 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 rectype 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: $F(n) \to F(n+1)$) 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 rectypes. 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 rectype of the language that is being reasoned about. There now follows a rather nice illustration from Part Ia of the Computer Science Tripos.

Exercise 4

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

$|a| = 1$,
$|\top| = 1$,
$|\bot| = 1$,
$|A \wedge B| = |A| + |B| + 1$,
$|A \vee B| = |A| + |B| + 1$,
$|\neg A| = |A| + 1$.

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

$tr(a) = a$, $tr(\top) = \top$, $tr(\bot) = \bot$,
$tr(A \wedge B) = tr(A) \wedge tr(B)$,
$tr(A \vee B) = \neg(\neg tr(A) \wedge \neg tr(B))$,
$tr(\neg A) = \neg tr(A)$.

Prove by structural induction on propositional formulæ that

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

for all Boolean propositions A."

Message for students on the 20/21 reading course: this next question is a sleeper for WQO theory and can be omitted.

Exercise 5

• *Declare the rectype of α-lists. (Observe that it is free.) Suppose the type α has been equipped with a quasiorder $\leq_\alpha$. 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.*

• *Declare the rectype of α-trees, and observe that it is free. Define stretching for α-trees, and give a recursive definition.*

Should really provide a model answer 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 2.10.2) we can take this kind of thing for granted.

2.5 Engendering Relations

All rectypes–since they are generated by constructors–will have a sort of **engendering relation**[7] 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 subformulæ, 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.[8]

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., $\lambda p.\langle S(\texttt{fst}\ p), S(\texttt{snd}\ p)\rangle$).

The following triviality is important.

THEOREM 3 *The engendering relation of a rectype is well-founded.*

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 $(\forall y)(y < x \rightarrow y \notin X)$. ■

> *HOLE Actually we have to be very careful how we state this (Thank you, Julian Ziegler Hunts!) beco's it's not correct as stated. It's certainly true if the rectype is free, but we can make it fail for silly reasons. Suppose we add to the constructors for the rectype* $\mathbb{N}$ *the identically zero function* $\lambda n.0$. *We must find a way of excluding perverse cases like that!*

EXERCISE 6 *(*)*

(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.*

[7] This is not standard terminology.
[8] A joke from Allen Hazen: "is the transitive closure of" is the transitive closure of "is the transitive closure of".

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 $(\forall n)(\forall m < n)(\forall X \subseteq \mathbb{N})(m \in X \to X \text{ has a } <\text{-least member})$.

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.*

The rest of this section is an amusing aside which can be skipped

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 nonempty set of power sets has an* $\in$*-minimal member.*

Proof:

Let $\mathcal{X}$ be a nonempty 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 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 endebted 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 not a fact about the cumulative hierarchy.*

2.6 Rectypes and Least Fixed Points

2.6.1 Fixed Point Theorems

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

I assume you know the Tarski-Knaster theorem from course materials of earlier incarnations of Part II Logic-and-Set-Theory[9], so i shall not recapitulate it here. You were told the Bourbaki-Witt theorem, but i'll recapitulate anyway.

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

THEOREM 4 *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 $X \to X$ 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 $(\forall z \in C(x))(z < y \to f(z) \leq y)$. 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.

[9]It was not lectured in 19/20, so you might have to scout around: wikipædia is a good place to start.

1. (Contains x) Vacuously!

2. (Closed under f) Suppose $y \in \{w \in C(x) : (\forall z \in C(x))(z < w \to f(z) \leq w\}$. We will show $(\forall z \in C(x))(z < f(y) \to f(z) \leq f(y))$. 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 $S \subseteq \{w \in C(x) : (\forall z \in C(x))(z < w \to f(z) \leq w)\}$ 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. ■

I do not propose to regard this proof as examinable. It is worth noting that the definition of $C(x)$ is—more-or-less bang-on-the-nose—the same as a declaration of the recursive datatype On of all ordinals.

2.6.1.1 Exercises on fixed points

Exercise 7 *Show that the fixed point of the Tarski-Knaster theorem is $\leq_X$-minimal.*

Exercise 8 *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 9 *Let R be a binary relation on a set X. Let F be a fuzzy on X. Define a new fuzzy on X by $x\,F'\,y$ iff $(\forall x')(x'\,R\,x \to (\exists y')(y'\,R\,y \wedge x'\,F\,y')) \wedge (\forall y')(y'\,R\,y \to (\exists x')(x'\,R\,x \wedge y'\,F\,x'))$. 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 10 *(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 11 *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 12 *An old Part II 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 $f : X \to Y$ and $g : Y \to X$ be injections. By considering $F : \mathcal{P}(X) \to \mathcal{P}(X)$ defined by

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

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

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

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

Exercise 13 *(Probably only for Group Theorists)*

A group G is **complete** *iff it is isomorphic to $Aut(G)$, 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 $Aut(G)$, and thence that every group with trivial centre embeds in a complete group with trivial centre.

2.6.2 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 sets[10] into itself. Here are some examples.

[10] And 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.

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

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

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

($\mathcal{P}_{\aleph_1}(x)$ is the set of countable subsets of x.

Explain this in more detail!!

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.

Exercise 14 *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 Nfinite.

In (i) you are of course not allowed to use countable choice—that would make it trivial[11]*.*

Do not attempt part (ii) unless/until you are fluent in constructive logic.

In both these case we mean 'hereditarily' in the sense of the least fixed point.[12]

[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.]

2.7 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 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 rectypes of finite character ($\mathbb{N}$, language of first-order Logic, etc etc). Here are some of infinite character

[11] And you actually have to prove that every hereditarily D-finite set is *hereditarily* finite.

[12] That is: the set of hereditarily D-finite sets is the $\subseteq$-least set identical to the set of its D-finite subsets.

(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.

Rectypes (i) and (ii) are of bounded (countable) character; (iii)—(v) are of *unbounded* ("absolutely infinite") character. Explain these expressions

Miniexercise: provide recursive declarations of (ii)–(v). (We declared (i) in section 2.6.2.)

You have probably not worried at all about whether or not rectypes of finite character (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 rectypes 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.

2.7.1 Rectypes of Unbounded Character are Paradoxical

Let's get out of the way the fact that rectypes of unbounded character are practically guaranteed to *not* be sets: they are paradoxical. (Interestingly the corresponding *co*-rectypes 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*:

$x \setminus \{y\}$ exists for all x and y. **Subscission**[13]

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 15.

Exercise 15 *(*)*
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 rectype of unbounded (absolutely infinite) character: the obvious inductive definition of wellfounded sets defines $WF(x)$ as $(\forall y)(\mathcal{P}(y) \subseteq y \to x \in y)$.

If we want to do this is 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 $(\forall y)((\forall z)(z \subseteq y \to z \in y) \to x \in y)$.
Substitute $V \setminus y$ for y getting

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

Contrapose getting

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

This is

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

[13] I don't think this is standard terminology; i learnt it from Allen Hazen and I think the expression is his coinage.

. . . 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 $A \to B$ is logically equivalent to its contrapositive $\neg B \to \neg A$, and that $\neg\forall$ is equivalent to $\exists\neg$. These two principles are not constructively valid. There is a constructive theory of wellfounded sets, and it supports induction over $\in$, but it does not support proofs using "$\in$-minimal elements". (Similarly constructive arithmetic supports mathematical induction but does not support the least-number principle).

2.7.2 Natural Numbers and Quine's trick

$\mathbb{N}$ is a rectype 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 16 *(*)(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

$$q(n) \longleftrightarrow (\forall Y)((n \in Y \wedge (P\text{“}Y \subseteq Y)) \to 0 \in Y)$$

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

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

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.

2.7.3 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 rectype, 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 rectype. This works for rectypes of finite character because there are only ω layers, and we use replacement for a function defined on $\mathbb{N}$ that enumerates the layers. Rectypes 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 rectype 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 rectypes of infinite character than for rectypes of finite character (tho' not for Borel sets in a fixed topological space). *HC* (see https://www.dpmms.cam.ac.uk/study/II/Logic/2012-2013/LSqns3.pdf Q9) 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.

2.8 Ordinals

One particularly important rectype of infinite character is the ordinals. The ordinals are a rectype 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 4. 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.

2.8.1 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.

Definition 3 *When $\langle X, R\rangle$ and $\langle Y, S\rangle$ are wellfounded structures a morphism $f : \langle X, R\rangle \to \langle Y, S\rangle$ 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 $\rho : X \to On$ is the unique parsimonious morphism $f : \langle X, R\rangle \to On$.

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 rectypes of finite character and parsimonious maps.

This section is very short beco's the idea of rank functions is very simple. But they are ubiquitous and very useful. A rank function for a wellfounded structure is a measure of the complexity of the structure. Look at `https://en.wikipedia.org/wiki/Sylver_coinage`. What is the rank of the tree of possible positions in this game?

2.9 Restricted Quantifiers

Quantifiers in the style '$(\forall x\, R\, y)(\ldots)$' and '$(\exists x\, R\, y)(\ldots)$' are said to be **restricted**. The intended semantics treats them as '$(\forall x)(x\, R\, y \to \ldots)$' and

'$(\exists x)(x\,R\,y \wedge \ldots)$'. 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'[14]) but the two *loci classici* of restricted quantifiers are

(i) [wellfounded] set theory, with the quantifiers '$(\forall x \in y)(\ldots)$' and '$(\exists x \in y)(\ldots)$'

and

(ii) arithmetic of $\mathbb{N}$, where the quantifiers are '$(\forall x < y)(\ldots)$' and '$(\exists x < y)(\ldots)$'.

In both these classical settings the binary relation doing the guarding is the engendering relation of the rectype[15]

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

Definition 4
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* $(\forall m \in M)(\forall m' \in M')(m'\,R\,m \to m' \in M)$.[16]

Exercise 17 *(*)*
Review exercise 11 on the wellfounded part of a binary structure, and establish that every [binary] structure is an end-extension of its wellfounded part.

Explain axiom scheme of collection here

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

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:

[14] I 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.

[15] Well, $\in$ 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

[16] However, 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.

If $\mathfrak{M}$ is a substructure of $\mathfrak{N}$ and ϕ is quantifier-free then $\mathfrak{M} \models \phi$ iff $\mathfrak{N} \models \phi$[17];

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 $\mathfrak{N} \models \phi$.

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?)

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

Explain axiom scheme of collection at this point

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 (repectively 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 classes[18]: 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 '$(\exists x)(R(x, y) \wedge \ldots)$' and '$(\forall x)(R(x, y) \rightarrow \ldots)$', 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).

[17] You proved this in Part II.

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

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.

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 5 *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

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

xis 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 existental 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.

2.10 Infinitary Languages

Definition 5 *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.)

2.10.1 "Wellfounded" is Infinitary Horn

Exercise 18 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 18 *Show that there is no first-order theory of a wellfounded relation.*

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

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

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 42). 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 is invited to prove it if they have not already done so.

Exercise 19 *(*)*
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}.

2.10.2 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:

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

Some are straight-out illfounded[19]:

$$\sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2 + \ldots}}}}.$$

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

[19] I made the foolish mistake of evaluating this expression and—yes!—it's the bloody golden section. Would be. That is *so* sad.... Serves me right.

first-year Analysis that semantics for these infinitary expressions is not a trivial exercise, and (*pace* the remarks on p. 30) one spends a lot of the early stages of Analysis learning that a lot of things that shouldn't matter do, indeed, not matter. See the footnote on page 89.

It is presumably in some sense undecidable whether or not an expression of this kind *has* semantics at all.

For example:

Solve

$$x^{x^{x^{x^{x^{\cdots}}}}} = 2$$

. . . but first of all make sense of it!

The exponent on the LHS is $x^{x^{x^{x^{x^{\cdots}}}}}$ which we are told is 2, so $x^2 = 2$ and $x = \sqrt{2}$. That was easy. The problem with this is that the second equation gives $x^4 = 4$ and thence $x = \sqrt{2}$ again. They can't both be right!

Of course the answer is that the reasoning that led us to conclude that $x = \sqrt{2}$ in the first place doesn't prove that that is the answer. All we have done is show that **if** there is a solution it must be $\sqrt{2}$. We haven't shown that there **is** a solution. In fact it is a simple matter to show by induction that the approximants to the LHS, which we generate as follows

$$a_0 =: \sqrt{2}; \quad a_{n+1} =: (\sqrt{2})^{a_n}$$

. . . are all less that 2. We do this as follows

$$a_{n+1} = (\sqrt{2})^{a_n} < (\sqrt{2})^2 = 2$$

where the middle inequality follows by induction hypothesis. So the sequence has a limit which is ≤ 2. (tho' of course we have to prove that the approximants converge to the answer).

Let's see what we can do that is more general.

Let $F(x) =_{df} x^{x^{x^{x^{x^{\cdots}}}}}$

We have $x^{F(x)} = F(x)$. The inverse to this function is the function $x \mapsto x^{1/x}$, obtainable as follows.

$$x^y = y$$

$$y \cdot log(x) = log(y)$$

$$log(x) = (1/y)log(y)$$

$$x = y^{1/y}$$

This is much easier to understand. For example we can differentiate it. It is the same as $e^{(log\ x)/x}$ whose differential is of course $e^{(log\ x)/x} \cdot (1/x^2 - (log\ x)/x^2)$. This is zero when $x = e$, and this is clearly a maximum. The fact that the differential is zero there of course means that F reaches a maximum at $e^{1/e}$ and

that $F'(e^{1/e})$ is infinite. This gives us the amusing but (as far as i know) useless fact that

$$(e^{1/e})^{(e^{1/e})(e^{1/e})(e^{1/e})(e^{1/e})(e^{1/e})\cdots} = e$$

(Check this: if the LHS is to evaluate to x we must have $(e^{(1/e)})^x = x$ and e is certainly a solution to this equation.)

We can get a power series expansion of F for values of x not much bigger than 1. Let Σ be the power series for $F(1+x)$. Then we have

$$(1+x)^\Sigma = \Sigma$$

and we can use the binomial theorem to expand the left hand side. This gives us a sequence of equations expressing later coefficients of Σ in terms of earlier coefficients in a wellfounded way. I haven't worked out the general formula for a_n the coefficient of x^n in $F(1+x)$ tho' in principle it could be done. ($a_0 = 1$ for a start!)

A slightly better example is continued fractions. These are wffs in a language with an ill-founded subformula relation. Might be an opportunity to say something about lazy evaluation. One can decide the truth-values of things like $F > n$ (where F is a continued fraction) sometimes by lazy evaluation.

If we try to write out the first epsilon number without using epsilon notation we end up with an infinite formula with an illfounded subformula relation.

But they don't yet know about ϵ numbers!

2.11 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 least fixed point $\bigcap\{Y : \mathcal{P}_{\aleph_0}(Y) \subseteq Y\}$ for the function $\lambda y.\mathcal{P}_{\aleph_0}(y)$ where $\mathcal{P}_{\aleph_0}(y) = \{x \subseteq y.|x| < \aleph_0\}$. $\bigcap\{Y : \mathcal{P}_{\aleph_0}(Y) \subseteq Y\}$ 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 20

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 21 *Co-rectypes support co-induction. Explain and justify co-induction.*

2.12 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 *fiat*; 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 rectype is constructed from objects $y_1 \ldots y_n$ by means of a constructor f, then a certificate that x belongs to the rectype 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 rectype themselves form a rectype.

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

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

2.12.1 Free *vs* non-free rectypes (ambiguous parses)

A rectype if *free* is every element of it has precisely one certificate. Any free rectype is a surjective image of its rectype of certificates (and rectypes of certificates are always free). If a rectype 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 rectype 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 rectype has a certificate implies that a union of countably many countably sets is countable.

The rectype 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 $f : V_\omega \to V$ by $f(\emptyset) = \emptyset$; $f(x \cup \{y\}) = f(x) \times \{y\}$, because we could have $x \cup \{y\} = z \cup \{w\}$ with $x \neq z \wedge y \neq w$.

2.12.2 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 rectype (as countable sets typically are). It is the job of composers to create such melodies. 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 3

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 at `http://www.gutenberg.org/ebooks/12`

3.1 Primitive Recursion

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

Definition 6 *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 nth of its arguments;*
- *the* **constantly zero** *function: the function that always returns 0.*

. . . and closed under

- **composition** *(see below) and*
- **primitive recursion***:*

$$f(\vec{x}, 0) := g(\vec{x}); \qquad f(\vec{x}, S(y)) := h(\vec{x}, y, f(\vec{x}, y)). \tag{3.1}$$

In 3.1 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.*

Fit in somewhere: can by rule induction that are all total, dominate ackermann, and compu by register machine

Observe the extremely restricted nature of the construction of primitive recursion. You "recurse" on only one variable, and you are allowed only one call to earlier values of that variable, and that one call must be to its immediate predecessor. This means that—to take a familiar example—the usual declaration of Fibonacci numbers is not primitive recursive. (That's not to say that the function enumerating them is not primitive recursive . . . it is, but that's because we can declare it in other ways).

Primitive recursion is not the only kind of recursion, but it is special in that the clauses of a primitive recursive declaration echo precisely the clauses in the recursive definition of $\mathbb{N}$ as a recursive datatype. You are a natural number iff you are 0 or the successor of a natural number; a function defined by primitive recursion knows what to do to 0 and, if it knows what to do to n, knows what to do to successor of n.

What might primitive recursion on other rectypes be?

This is a sleeper for Kruskal's theorem and can be skipped

Exercise 22 *(*)*

1. *Consider for example the rectype of α-lists from p. 28. What is primitive recursion on α-lists?*

   If α is equipped with an order $\leq_\alpha$ we say that an α-list l_1 (thought of as a function $l_1 : [1,n] \to \alpha$, for some n) stretches into an α-list l_2 (thought of as a function $l_2 : [1,m] \to \alpha$, 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 23

$\mathbb{N}\times\mathbb{N}$ *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 $\overbrace{S(S(S\ldots 0\ldots))}^{n \text{ times}}$.) '$\underline{n}$' 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).

- 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 second argument place in the *definiens*[2], the thimg on the right.. 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 twice[3] it would be forced to be eventually periodic.

- Observe that in our projection functions ("pick the ith 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 rectype 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

[1] Syntax 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 $(\exists x_1 \ldots x_n) \bigwedge_{i \neq j \leq n} (x_i \neq x_j)$. See the discussion on page 156

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

[3] Well, 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...

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 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 7

(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 Red Queen's claim that Alice can't do addition.

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

Definition 8
$f_0(m, k) := m + k$;
$f_{S(n)}(m, 0) := m$;
$f_{S(n)}(0, m) = f_n(m, 1)$;
$f_{S(n)}(S(m), S(k)) = f_n(m, f_{S(n)}(S(m), k))$.

Exercise 24 *(*) 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 9

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

[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 $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$, whereas any D-T function with an infinite subscript sends at least some pairs of naturals to infinite ordinals.]

Exercise 25 *What is the next function in the Doner-Tarski hierarchy after exponentiation? And the one after that?*

Exercise 26 *(*)*

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.*

3.1.1 Some quite nasty functions are primitive recursive

Exercise 27 *(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?

3.1.2 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 3.1 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:

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

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

Rewrite this section define $\restriction$

One thing we can do immediately is define each restriction $\texttt{fact}\restriction[0,k]$. That's easy. The hard part is to glue them together. For each k we can define a function which we can call fact-k which will of course be $\texttt{fact}\restriction[0,k]$. We show that any two functions in $\{\text{fact-}k : k \in \mathbb{N}\}$ 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 5.

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:

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

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 $\forall^2_1$. 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 2.12.

Suppose we have defined f by primitive recursion:

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

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 $0 \le i \le n$, 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 89.

The prime powers trick enables us to prove the following:

REMARK 4 *If f is a function $\mathbb{N}^k \to \mathbb{N}$ declared by primitive recursion then there is a formula $\phi(y, x_1 \dots x_k, \vec{z})$ in the language with 0, 1, +, ×, 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 6 *If f is a function $\mathbb{N}^k \to \mathbb{N}$ declared by primitive recursion then there is a formula $\phi(y, x_1 \dots x_k, \vec{z})$ in the language with 0, 1, +, ×, < and = ("the language of ordered rings") containing no unrestricted quantifiers such that*

$$y = f(x_1 \dots x_k) \text{ iff } (\exists \vec{z})\phi(y, \vec{x}, \vec{z})$$

(In fact—and this is a famous theorem of Davis, Putnam, Robinson and Matiyasievich[4]—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.

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

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: '$(\exists I)(\exists O)(\exists p)(\ldots)$'. The prime p will be chosen big enough so that the following picture makes sense.

$\ldots$	p^2	p^1	p^0	
$\ldots$	2	1	0	I
$\ldots$	$f(2)$	$f(1)$	$f(0)$	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.

Some Local Definitions

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

check this dfn

x `DIV` y is the largest integer z s.t. $y \cdot z \leq x \leq y \cdot (z+1)$ and
x `rem` y is the remainder when x is divided by y.

Strictly speaking[5] 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

"$z = x$ `DIV` y iff $x \cdot z \leq y < x \cdot (z+1)$" and
"$w = x$ `rem` y iff $(\forall z < x)(z = (x$ `DIV` $y) \rightarrow z \cdot y + w = x)$".

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

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

or

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

[5]Thank you, David Edey!

. . . depending on whether we want the quantifier to be '$\exists$' or '$\forall$' (which will in turn depend on whether the occurrence of $\phi(x \texttt{ DIV } y)$ 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 $(\forall w < z)(w|z \to p|w)$—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 remainder[6] then divide that remainder by (z/p), we find that the quotient is o_z."

In symbols:

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

> 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, $(O \texttt{ rem } z) \texttt{ div } (z/p) = o_z$. Suppose $O = a+bp+cp^2+dp^3$, and we take $z = p^2$. Then $(O \texttt{ rem } p^2) \texttt{ div } p = (a + bp)/p = b$, but we want c.
>
> I think $(O \texttt{ div } z) \texttt{ rem } p$ works though. $(O \texttt{ div } p^2) \texttt{ rem } \mathrm{p} = (c+dp) \texttt{ rem } p = c$. Alternatively $(O \texttt{ rem } pz) \texttt{ div } z$ does it too I think."

Exercise 28 *Is Jack Webster correct? Bottle of college port for LaTeX source code of a corrected version.*

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 $I \texttt{ rem } z) \texttt{ DIV } (z/p)$;

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 $(O \texttt{ rem } z) \texttt{ DIV } (z/p)$).

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 $o_z = g(o_{(z/p)}, i_{(z/p)}, \vec{s})$;

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

(2) Initialising: we have to say $i_1 = 0$ and $o_1 = f(0, \vec{s})$;

(3) The nth 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 $I = 1 \cdot p + 2 \cdot p^2 + \ldots$. This explains the bound "$< I$" whevever 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.

■

3.2 Exercises

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

Exercise 29 *(*)*

For each of the following functions $\Phi : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$, 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 $f, g : \mathbb{N} \to \mathbb{N}$, 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 $[\mathbb{N} \rightharpoonup \mathbb{N}]$ 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?

3.2.1 Primitive Recursive Relations

A relation is primitive recursive if the characteristic function of its graph is primitive recursive[7] [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 $r : \mathbb{N}^\rho \to \{0,1\}$ (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.

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.

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

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 64) 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 xs 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

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

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:

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

`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

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

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*

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

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

This is **bounded search**. Hofstader [32] memorably calls this "BLOOP".

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

$$\texttt{pair}(x,y) = \tfrac{(x+y)\cdot(x+y+1)}{2} + x$$

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

$\texttt{fst}(\texttt{pair}(m,n)) = m$,
$\texttt{snd}(\texttt{pair}(m,n)) = n$ and
$\texttt{pair}(\texttt{fst}(r),\texttt{snd}(r)) = r$.

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.*

In future when we write '$\texttt{pair}(x,y)$' 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 rectype; so too is the graph of a primitive recursive relation. For example, the 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 64 p 114.

Exercise 35 *(*)*

1. *The declaration:*

   $\texttt{Fib}(0) := 1$;
   $\texttt{Fib}(1) := 1$;
   $\texttt{Fib}(n+2) := \texttt{Fib}(n+1) + \texttt{Fib}(n)$;

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

2. *The* **iterate** $\mathrm{It}(f)$ *of f is defined by:* $\mathrm{It}(f)(m, n) = f^m(n)$. *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* $\texttt{pair}(H(y), y)$*.]*

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 27 on page 55). 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

$$n \mapsto (\texttt{if } n = 2k \texttt{ then } k \texttt{ else fail})$$

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*

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

where k is the number of distinct prime factors of n.[8]

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

3.2.2 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. (Miniexercise: supply this definition.) Here is another example—from a 1a Computer Science exam of some years ago.

$$f(n) := \texttt{if } n = 0 \texttt{ then } 0 \texttt{ else } g(f(n-1)+1, 1) - 1;$$
$$g(n, m) = f(f(n-1)) + m + 1.$$

(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.

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*

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

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

Show that f is primitive recursive too.

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

We will later need the fact that that the function enumerating the primes in increasing order is primitive recursive. For this we need that a search $(\exists x < f(n))(\ldots)$ 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 nth prime is primitive recursive*

3.3 μ-recursion

Does the rectype 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.

I may have my ms and ns muddled up

3.3.1 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 it does, indeed, not have a primitive recursive declaration. 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. We will do both of these.

Actually the project (i) has some interest independent of (ii). The proof obviously is going to procede by an induction of some kind. The allegation (i) is of the form "for all m and for all n, something happens". An allegation of the form "for all n, something happens" might, on the face of it, be provable in either of two ways. It might be universal generalisation: "Let n be an arbitrary natural number ..."; one then reasons about this natural number, establishes whatever it is, and then reflects that the proof crucially didn't depend on n

being any particular number, so we have in fact proved it for *all* n. Or we might do an induction on n. This means that when trying to prove an assertion that starts "for all $n_1 \dots n_k \dots$" we have (in principle) 2^k different strategies. In this case—the totality of Ackermann—there are apparently four ways of attacking it. As it happens, only one way works. You have to do *two* inductions, one on each variable.

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

Proof:

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

Base case: $m = 0$.
This is straightforward; we have $A(0,n) := n+1$ by stipulation.

Induction step:

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

Base case:

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

Induction step:

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

■

(You will note (i hope) how i have artistically indented the inner loop!)

Check that the ms and ns have not got jumbled

Here is another proof of theorem 8, this time by induction on the lexicographic order of $\mathbb{N} \times \mathbb{N}$. (You proved in exercise 19 that a lexicographic product of finitely many wellfounded strict partial orderings is a wellfounded strict partial ordering).

Proof:

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

1. $m = 0$. In this case $A(m,n) = n+1$;
2. $n = 0$. Then $A(m,n) = A(m-1,1)$ which is defined by induction hypothesis, since $\langle m-1,1\rangle <_{lex} \langle m,n\rangle$;

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

■

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. If we need to do an induction over a wellorder of length ω^2 we need (look at the exponent) *two* nested inductions.

It's probably not unduly fanciful to think of double recursion (*à la Ackermann*) as primitive recursion over a different rectype—$\mathbb{N} \times \mathbb{N}$ tho'rt of in the right way.

3.3.2 The Ackermann function dominates all primitive recursive functions

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

[The idea behind this proof is very simple but the details are horrible and I do not propose to lecture it. It will not be examinable. Nevertheless you might like to try the exercises in this section.]

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). This exercise has something of the nature of writing machine code for register machines. You don't want to *spend time doing* it, but it's a good thing to *have done once*.

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) $A(m,2n) < A(m+2,n)$*.*

Exercise 42 *(For those who did Number Theory in Part II)*
Is the class number function h primitive recursive?
Justify your answer in general terms.

Now that you are familiar with the concept of *primitive recursive function* and with at least one function that isn't primitive recursive it can do you no harm to get into the habit of asking questions along the lines of exercise 42. May as well put the concept to good use!

Definition 10
$f : \mathbb{N} \to \mathbb{N}$ **dominates** $g : \mathbb{N} \to \mathbb{N}$ *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 $f = \mathrm{S}$ (the successor function), f the 0 function, and f a projection.

Composition.

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

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

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:

$$\begin{array}{lll}
g(f_1(\vec{x}), \ldots, f_n(\vec{x})) & < A(c_g, f_1(\vec{x})) & \text{definition of } c_g \\
& < A(c_g, A(c_{f_1}, \max \vec{x})) & \text{definition of } c_{f_1} \\
& \leq A(m, A(m, \max \vec{x})) & \text{definition of } m \\
& < A(m, A(m+1, \max \vec{x})) & \text{mono in both args} \\
& \leq A(m, A(m+2, \max \vec{x} - 1)) & \text{Ex 41 part (iii)} \\
& < A(m+1, A(m+2, \max \vec{x} - 1)) & \text{mono in first arg} \\
& = A(m+2, \max \vec{x}). &
\end{array}$$

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

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:

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

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 $y = 0$:

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

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. ■

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 $\mathbb{N} \to \mathbb{N}$ ordered by dominance and that this sequence is cofinal in the primitive recursive functions (has no primitive recursive upper bound).

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 52) 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 43 *(*) (1991:5:10 (CS))*[9]

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

Ackermann's function is defined as follows:

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

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 44 *(*)*

1. *Write out a definition of a constructor of* **double recursion** *so that you now have a rectype 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 44 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 44 invites you to prove analogues—for

[9] This was question 10 on paper 5 of the Cambridge Computer Science tripos 1991.

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.][10] 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.[11] This is because we want unbounded search[12] 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 rectype 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.**

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

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

[11] On page 65 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.

[12] Fans of [32] might be helped by a reminder that Hofstader calls unbounded search FLOOP (as opposed to BLOOP, *bounded* search, which we saw on page 63).

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 just *cannot* simply 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.[13]

Fortunately it will turn out that any function that we can define by more than one inversion can always be defined using only one.[14] 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.

[13]It 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.

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

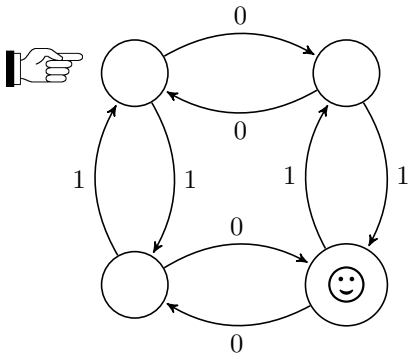
Chapter 4

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[1]) 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.

4.1 Finite State Machines

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



[1]Some of you encountered pushdown automata and contxt-free languages in Part II: I have nothing to say about them in this course

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

Th 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 **??**.

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

Exercise 45 *Show that, for any base b, the set of base-b notations for natural numbers is a regular language.*

4.1.1 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.

4.1.2 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 and Nerode.

The definition we have given of *regular language* can be rephrased by saying that if we start with a machine $\mathfrak{M}$, and think of it as a digraph with edges decorated with characters from Σ, and some vertices decorated with smilies (and one with a pointy finger thingy) then we can think of the corresponding regular language as a kind of *unfolding* of $\mathfrak{M}$, the set of paths thru' $\mathfrak{M}$ that start at the pointy finger and end on a smilie—or rather the sequence of decorations of the edges along such a path. That's how a machine gives rise to a regular language, by an unfolding. Can we sensibly describe an inverse to this process? In the course of the unfolding we make lots of copies of the various states of the machine; the challenge is, on being given something that might have arisen by an unfolding of this kind, to recover what the machine was that was unfolded. We have to discover which things in the unfolding are copies of a single thing (state) in the machine. In other words we are looking for an equivalence relation on Σ^* whose quotient will turn out to be $\mathfrak{M}$.

Accordingly let $L \subseteq \Sigma^*$, be a language *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 $\Sigma^*/\sim$ 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.

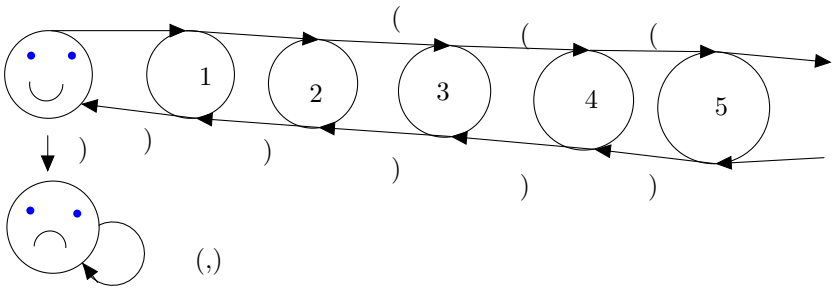
And *vice versa*: equally clearly, if L is regular, then there will be a fixed point of finite index: every machine that recognises L embodies a fixed point. Observe that we can prove by induction on n that $\sim_n$ is a superset of any fixed point, so the fixed point supplied by this construction must be 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[2].

[2]Look at `https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2009p2q9.pdf` It de-

[The existence of a fixed point follows from general results such as Knaster-Tarski (and probably Bourbaki-Witt) which you saw in Part II. Knaster-Tarski not only tells you there are fixed points but that there is a complete lattice of them, so there is probably quite a lot to be said.]

What if L is not regular? 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. The machine we get for the matching brackets language will have, as it were, $1 + \omega$ states. The start state (which is also the sole accepting state) is 0; -1 is the fail state, and n is the state where there are n outstanding left brackets. (I must put in the missing '(' pushing us to the right ...)



This machine has infinitely many states, but is a finite object in the sense that it admits (as we have just shown!) a finite description. Our determination to think of mathematical objects as *finite* objects (wherever possible) leads us to invent other machine architectures which will enable us to think of these sensible (infinite) quotients as explicitly finite objects[3] by somehow turning the finite description of the infinite machine into a specification of a finite machine of the new architecture. One such architecture is *pushdown automaton*. However we probably won't get round to them, and we will occupy ourselves with finite state machines only, while allowing them to be *nondeterministic*.

4.1.3 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.

---

fines an equivalence relation on Σ^* by $w \sim w'$ iff $(\forall u \in \Sigma^{)})(wu \in L \longleftrightarrow w'u \in L)$. It invites the reader to notice that, if $L = a^n b^n$, for $n \neq m$, a^n and b^n are inequivalent, There is a raven-protected discussion answer on `https://www.cl.cam.ac.uk/teaching/exams/solutions/2009/2009-p02-q09-solutions.pdf`.

[3]We will see later something of the same flavour: every theory with an (infinite but) decidable axiomatisation has a conservative extension in a new language that is finitely axiomatisable. See remark 9.

Nondeterministic machines (hereafter **NFA**s—"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 $\{c_1, c_2, c_3 \ldots\}$ 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 ... 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 ...". 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 labelled 'c' to the empty set of states.

(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 decomposible 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.

4.2 Stuff to fit in

If w sends $\mathfrak{M}$ thru' a loop and $\mathfrak{M}$ accepts uw^2v then one tends to assume that $\mathrm{hd}(w) \neq \mathrm{hd}(v)$. Should make a fuss about this. It can confuse people.

We ideally need the concept of a stream, so we can explain how we start the machine in the designated start state, and then give it a *stream* of characters. Ever thereafter, at each stage, when it has has been given an initial segment of a stream, it must be in an accepting state iff that initial segment belongs to the language in question. And it must be able to do this for all streams.

Of course we should say something here about how, in the natural realistic motivation for this stuff—parsers—the machines do in fact sit and field streams of stuff in precisely this way. Of course in that realistic setting there is a sort of `RESET` command which we might want to say something illuminating about.

Say something about string search engines. They use regular expressions

If you are a Part III student doing the reading course then you probably did all this stuff in Part II, so there should be no need to do the exercises which follow. Cast a very quick eye over to check you genuinely are on top of this stuff. I'm not going to set any questions on material that examined last year!

4.2.1 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. Describe[4] 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 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) $X_1 = (BX_1 \cup \{\epsilon\})$; $X_2 = BC(X_1 \cup \{\epsilon\})$.

[4] This word is very carefully chosen!

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?

   you have had a sleepless night over this you may consult the footnote for a hint.[5]

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 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.

[5] If this is to be a regular language, there must be a FDA that recognises it. What might this FDA be?

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) $\{w \in \{a, b\}^* : w$ contains exactly twice as many as as bs$\}$

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 46 *(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.*[6]

1. Does the set of strings in $\{a, b, c\}^*$ which have as many as as bs and cs put together make a regular language?

2. Let K, L and M be regular languages. Is $\{u \in L : (\exists v \in K)(uv \in M)\}$ a regular language?

3. Is the language of Roman numerals regular?

Stuff to fit in

[*this stuff is messages to myself: do not read!!!*]

To show that the reverse L^{-1} of a regular language L is regular, use regexps. Define an operation recursively:

$$(RS)^{-1} = S^{-1}R^{-1};\ (R^*)^{-1} = (R^{-1})^*;\ (R|S)^{-1} = R^{-1}S^{-1}$$

[6] 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!

The interleaving $R\,I\,S$ of two regular languages is defined recursively as follows:

The interleaving of RS with $R'S'$ is
$R'|S'$
R'^*
The interleaving of R^* with
$R'S'$ is
$R'|S'$
R'^*
The interleaving of $R|S$ with
$R'S'$ is
$R'|S'$ is $(R\,I\,R')|(R\,I\,S')|(S\,I\,R')|(S\,I\,S')$
R'^* finish this off

How are we to think of a nondeterministic machine? If it's in state s and we give it character c ('c' here is a variable!) then it goes into another state, but we just don't know what that state is?

Must make a big fuss about the recovery of a DFA from an NFA. This is interesting because this is a DFA arising abstractly.

Have to explain this business of maintaining more info than you actually need to answer questions on.

Prima facie every string has its own state, in the sense that a state is simply a state-of-knowledge about the string you have seen so far. You can think of the string as a history. Some states are alike in the sense that the differences between them have no bearing on the questions you have to answer.

It's a PROLOG program

The presence on the scene of nondeterministic machines rubs in the importance of distinguishing between *accepting* and *recognising*. A deterministic machine **recognises** the set of those strings which it accepts. A *nondeterministic* machine recognises the set of those strings that it *might* accept.

LAC students wonder: what happens if you give the machine a character that doesn't belong to its input alphabet? The answer is that this is absurd: you can't! It's a bit like saying what happens if the batsman's response to a googly is to play the ace of spades. The answer is; there is no action the batsman can make which constitutes playing the ace of spades...

Sometimes we need to think abstractly. Sometimes we need to think concretely. (The Rubik cube question)

Remember to distinguish between states and bits. If the machine needs to remember the truth-value of n propositions, then it will need 2^n states.

How you need the three characterisations at different times. You need regular expressions to explain why the reverse of a regular language is regular; you need machines to explain why the complement of a regular language is regular. (Still need examples to illustrate why you need NFAs and grammars)

We ideally need the concept of a stream, so we can explain how we start the machine in the designated start state, and then give it a *stream* of characters. Ever thereafter, at each stage, when it has has been given an initial segment of a stream, it must be in an accepting state iff that initial segment belongs to the language in question. And it must be able to do this for all streams.

Of course we should say something here about how, in the natural realistic motivation for this stuff—parsers—the machines do in fact sit and field streams of stuff in precisely this way.

There is a sort of reset command which we might want to say something illuminating about.

We can glue the streams together to get a tree if we like.

Make a point about the pigeonhole principle and the pumping lemma.

How to get a machine from a language. Ask yourself: "I am a machine: when i power myself up and look at the first character, what do i want to know? And depending on what answer i get, what might i want to know next?"

Fit in somewhere: overloading of juxtaposition: $abc\ uvw$ and ABC. I think this is something to do with quasiquotation ...

4.3 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

`https://www.youtube.com/watch?v=E3keLeMwfHY`

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. $R^+ \to L$: add 1 to contents of register R and jump to instruction with label L.

2. $R^- \to L', L''$: 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 nth member of the sequence is the instruction to be executed when in state n.[7]

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 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.[8]

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

> The sequence of length k whose nth entry is e_n is sent to $\prod_{0<n\leq k} {p_n}^{1+e_n}$.
>
> (p_n is of course the nth prime.[9]) 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 recur-

[7] I 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.

[8] Indeed 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: "*Un*important, your Majesty means, of course," he said in a very respectful tone, but frowning and making faces at him as he spoke.
>
> "*Un*important, 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.

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

sive. 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 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 11 *Hartley Rogers [49] says a system of indices ψ is* **acceptable** *if, for every n, there are total computable f and g such that*

$$\psi^n_e \simeq \phi^n_{f(e)} \ \wedge \ \phi^n_e \simeq \psi^n_{g(e)}$$

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 incomputable permutation—you can see what happens.

I think this is the first place where we use the $\{n\}$ notation for computable functions

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 mth machine is the machine with gnumber

m. There is a convention of writing '$\{e\}$' for the function computed by the eth machine/eth program (we do not distinguish between machines and programs), and also writing

Definition 12

- *"$\{e\}(n)\downarrow = k$" to mean that the eth machine halts with input n and outputs k;*
- *"$\{e\}(n)\uparrow$" means that the eth machine does not halt with input n. In these circumstances we say $\{e\}(n)$* **diverges**.
- *"$\{e\}_z(n)\downarrow$" to mean that the eth machine halts with input n in $\leq z$ steps.*
- *"$\{e\}_z(n)\downarrow = x$" to mean that the eth 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 model[10] 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.

Make this announcement earlier

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*.) Left to my own devices i would write '$\{e\}$"$\mathbb{N}$!!

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-*in*tension, 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 '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 13

[10] In 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*.

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

Scrape together all earlier uses of '$\{n\}$'

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

An essential gadget is

Definition 14 *(Kleene's T function)*
Input m and i and t, then output a list of t states of the mth 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 6 on p. 57 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. 63.)

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 mth 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 rectype 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.

4.4.1 A Universal Register Machine

Kleene's T-function is primitive recursive, so there is a machine that computes it. Any such machine can be tweaked[11] into a **universal** or all-purpose machine: one that can simulate all others.

We need three auxilliary 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.

[11] By a process the computer scientists call *wrapping*.

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:

$\texttt{register\_0}(\texttt{last}(T(m,i,(\mu t)(\texttt{current\_instruction}(\texttt{last}(T(m,i,t))) = \texttt{HALT}))))$

which is what the mth machine does on being given i.

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

4.5 Decidable and Semidecidable Sets

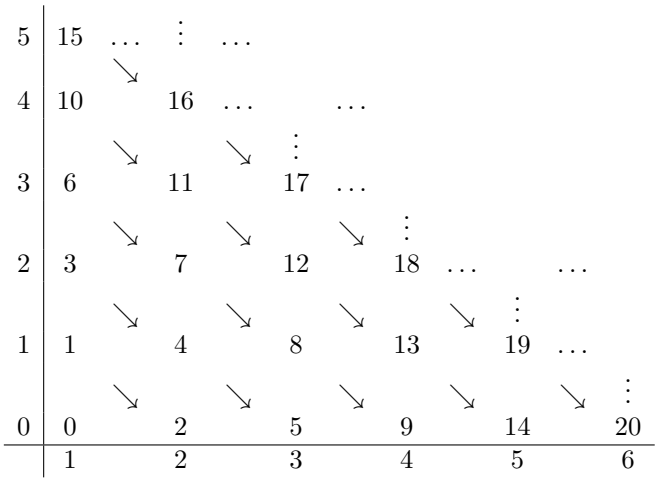
Mohammad is the last prophet; there are to be no further revelations. In contrast, both Christianity and Judaism hold out the possibility that there will be more revelation. Thus the set of truths revealed to/by Islam is decidable. The set of truths that are (to be eventually) revealed by Christianity or Judaism is semidecidable. It has a decidable axiomatisation of course but we have no idea what that axiomatisation is.

Well no, actually. That's a striking thought, but it's wrong.

One wants to say that the hallmark of the semidecidable set is that each and every one of its members get revealed to us at some point before the end of time. That is the intuition one tries to get across to students, but it's not the whole story. It's a necessary condition all right, but it's not sufficient. It's necessary also that the revelation be done by a humble *finite* engine. For consider the set of gnumbers of total computable functions. The oracle for this set could divulge all its members to us over time—in increasing order indeed—and we would know all its members and all its nonmembers by the end of time, but that doesn't make it semidecidable. The oracle is not a finite engine!

4.5.1 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 $\texttt{fst}(n)$ for $\texttt{snd}(n)$ 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 old inputs–and continually starting computations on new, later inputs–so that every computation is given infinitely many chances to halt. Of course once a computation with input k has halted, we do not revisit it: we emit the answer and carry on with the zigzag. (Therefore, at stage n, if $\texttt{fst}\ n$ is an input that has already halted, we procede at once to stage $n+1$.)



The y axis is inputs, the x axis is the number of steps you run. Thus, at stage 18, you run $\mathfrak{M}$ on input 2 for 4 steps[12].

This autoparallelism is really a breadth-first search through all the computations that $\mathfrak{M}$ is capable of.

We need a nice snappy name for engines that do this. I like to call them **volcanoes**. The idea is that volcanoes emit things unprompted—they don't need input; all you have to do is power them up. Any machine can give rise to a volcano, since all we need is a computable implementation of pairing-and-unpairing. It is true that the volcanoes we can get from a machine $\mathfrak{M}$ will differ in the order in which they emit their emissions, this order depending on our choice of implementation of pairing-and-unpairing, but this won't matter. We can plump for one such implementation and have done with it. Given time, $\mathfrak{M}$'s volcano will emit every number that is a value of the function $\{m\}$ computed by $\mathfrak{M}$. It is not guaranteed to emit the value for input 4 before it emits the value for input 3 (even if they both halt). Exercise 61 addresses this question and might merit a pit stop.

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 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.

[12]The 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. Perhaps it leaves little *caches* of information by each input. 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.

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 47 *Explain to people in your bubble why, if we think of a volcano as a function $\mathbb{N} \to \mathbb{N}$, 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 $\mathbb{N} \to \mathbb{N}$, 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 $f(0)$, $f(1)$, $f(2)$ ... (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.]

4.5.2 Decidable and Semidecidable Sets

One of the intentions behind the invention of computable functions was to capture the idea of a decidable set. One tries 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"$\mathbb{N}$, 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"$\mathbb{N}$ in these circumstances even though we are not promised an exclusion procedure. Thus the natural idea seems to be that of a *semi*decidable 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 nonempty[13] *set $X \subseteq \mathbb{N}$ are equivalent:*

[13] The empty set is obviously decidable!

(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 nth 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 48 *Show that every computable partial function has a computable (partial) right inverse.*

Provide a discussion answer

Exercise 49 *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 73 in section 3.3. 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 50 *Cook up an example to show that sometimes it won't.*

Haven't i got a nice example from Beeson about this?

Yes!

Define the partial function f by

$f(2x) =$ `if` $\{x\}(x)\downarrow$ `then` x `else fail`;
$f(2x+1) = x$.

Then f is partial recursive, and surjective. $f^{-1}\text{“}\{x\}$ is either $\{2x+1\}$ or $\{2x, 2x+1\}$. The hard inverse of f, on being given x, returns either $2x+1$ (which it does if $\{x\}(x)\uparrow$) or $2x$ (if $\{x\}(x)\downarrow$). Since this solves the diagonal `HALT`ing problem for us we conclude that the hard inverse of f is not computable.

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’.

Definition 15
(1) A set satisfying the conditions in remark 5 is **semi-decidable**[14]
(2) A set X is **decidable** *if X and $\mathbb{N}\setminus X$ are both semidecidable.*

There is an obvious generalisation of this definition of semidecidable to subsets of $\mathbb{N}^k$.

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 $f : \mathbb{N} \to \{0,1\}$ such that $X = f^{-1}\text{“}\{1\}$. 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 $\mathbb{N}^k \rightharpoonup \mathbb{N}$ is a semidecidable subset of $\mathbb{N}^{k+1}$ and the graph of a total computable function $\mathbb{N}^k \to \mathbb{N}$ 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 51 *(*)*

(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 $f\text{“}Y$ for some computable f and semidecidable $Y \subseteq \mathbb{N}$”.

[14] The 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’.

Exercise 52 *(*)*
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 1 [15]
Suppose $f : \mathbb{N} \to \mathbb{N}$ is total with no odd cycles.
Then there is $d : \mathbb{N} \to \{0,1\}$ 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 53 *(*)*
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 16 *A infinite subset of $\mathbb{N}$ is* **immune** *iff it has no infinite semidecidable subset.*

'Immune' is a computable analogue of 'infinite Dedekind-finite'.

Exercise 54 *(Too easy perhaps)*
Prove that the cartesian product of two immune sets is immune.

Exercise 55 *(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 79 in section 4.9.

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"$\mathbb{N}$ 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 55, so a semidecidable set of naturals can be thought of as a subset of $\mathbb{N}$ that happens to be a rectype in its own right. Indeed, we can take this further: by means of gnumbering, every finitely presented rectype 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.

[15] This question was put to me years ago by my Ph.D. supervisor, Maurice Boffa. It was open then, but i don't know if it is still open.

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 $X = \mathrm{dom}(\{m\})$ for some computable function $\{m\}$. Then

$\vec{x} \in X$	iff
$\{m\}(\vec{x})\downarrow$	iff
$(\exists y)(\{m\}_y(\vec{x})\downarrow)$	iff
$(\exists y)(\langle \vec{x}, y\rangle \in \{\langle \vec{z}, y\rangle : \{m\}_y(\vec{z})\downarrow\})$	

Observe that $\{\langle \vec{x}, y\rangle : \{m\}_y(\vec{x})\downarrow\}$ is clearly decidable, and that X is a projection of it.[16]

■

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 *semi*decidable 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

[16] Thanks to Philipp Kleppmann for this improvement on my original.

computable'). A set is **decidable** if its **characteristic function**[17] is total computable.

Definition 17 *The characteristic function χ_A of $A \subseteq \mathbb{N}$ is*

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

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

4.5.3 A Nice Illustration and a Digression

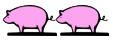
There is a natural example of an immune set, and it arises in a context of some independent interest.

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 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 $U(\rho_f :: \tau)$. (I think the idea is that we compute ρ_f from f before we fire up U). 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 $|\mu|$ where μ is the shortest input[18] 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:

[First we have to show that it's infinite! I assumed this was so obvious that i forgot to establish it!!]

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[19]. $C(n) = log_2(n)$ 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. ■

[17] In other traditions they are sometimes called **indicator functions**.

[18] part of μ is of course the string ρ_f for the function f that U is calculating.

[19] As Ben Millwood says, the overhead is not *literally* c_f, but it's at least a constant.

4.5.4 "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.

Not this year you don't!

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 order type ω. They are isomorphic, of course. Is the isomorphism a semidecidable set of ordered pairs?

Put $\langle \mathbb{N}, R\rangle$ on the left of the board and $\langle \mathbb{N}, S\rangle$ on the right. Clearly we can discover the isomorphism by the following deterministic process, of progressive refinements of finite approximations.

We start off by pairing 0 on the left with 0 on the right. Thereafter, at stage n we have paired off $\{0, 1 \ldots n-1\}$ on the left somehow with $\{0, 1 \ldots n-1\}$ on the right. After all, the ordering $\{0, 1 \ldots n-1\}$ equipped with R is isomorphic to $\{0, 1 \ldots n-1\}$ equipped with S! Then we add the extra element n to both sides. We know where n stands in relation to the numbers $0 \ldots n-1$ in the R ordering because by assumption R was a decidable set of pairs. S similarly. Sticking n into both sides might involve some rearranging! We aren't going to *reorder* the things on the left (or on the right) but we might insert n in the middle, and therefore be compelled to redirect some arrows. How long can this faffing about go on, for heavan's sake? Well, if 0 is the 23rd element according to R and the 17th element according to S then we will have finally paired off 0 on the left with its destined partner on the right and 0 on the right with its destined partner on the left by... by when? By the time we have seen all the numbers that R-precede 23 and all the numbers that S-precede 17. There are only finitely many such numbers, so there does come a point after which 0 lives happily ever after. The trouble is, we have no idea when that is, so this analysis does not give a proof that the bijection is decidable.

[It might be worth giving an example of such R and S to show that the bijection need not be decidable. I'm guessing there is such a pair, i don't know! For the moment, the point is that this example presses the 'in finite time' button but not in a way that guarantees computability of the result.]

Another realistic case in point is the task of mentally reconstructing the proof of Friedberg-Muchnik, (this is theorem 24, still to come) once you have forgotten the details ... and, believe me, you will forget the details.

The following exercise might help/amuse you.

Exercise 56 *(*)*

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

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

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

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

4.6 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. The way to understand this is to grasp that the concept of semidecidable set is much more like the concept of *counted set* than it is like the concept of *countable set*.

Exercise 57

(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"x is always a set whenever x is we say is 1-*setlike*. Only "1"-setlike? If, additionally,

$\{f\text{“}y : y \in x\}$ is a set whenever x is a set[20] we say f is 2-setlike. Similarly 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 $\mathbb{N}^k \to \mathbb{N}$ 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 $X \times Y$, $\texttt{fst}\text{“}X$ ...

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

Because of the natural bijection between $(A \times B) \to C$ and $A \to (B \to C)$ one can think of a natural-number–valued function of two (natural number) variables either as a function $\mathbb{N}^2 \to \mathbb{N}$ or as a function $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$. We have a notion of primitive recursive function $\mathbb{N}^2 \to \mathbb{N}$ or as a function and a notion of a primitive recursive function $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$. 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.

4.6.1 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

[20] PTJ would have me write this as '$f\text{“}x$'!

regular language. We will return to this in section 6.1.

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... errm ...) Parameters are allowed. get the definition straight

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 58 *(*)*
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:
Let $\mathfrak{M}$ be a volcano that emits axioms of T, and notate the nth axiom emitted by $\mathfrak{M}$ as ϕ_n. Then we obtain a decidable axiomatisation for T as

$$\{(\bigwedge_{0 \le i < n} \phi_i) \to \phi_n : n \in \mathbb{N}\}$$

It's probably worth spelling out why it is decidable. Let $\langle \phi_n : n \in \mathbb{N} \rangle$ be the stream of axioms emitted by the volcano that is zigzagging over the computable function f whose range is the set of axioms. The axioms in the decidable set are

$$(\bigwedge_{i<n} \phi_i) \to \phi_n$$

My decision procedure for this set of axioms is as follows

> If the formula is not a conditional, reject.
>
> If the antecedent A is not a list-conjunction ask whether or not A is the first thing emitted by the volcano. If it isn't, reject;

> If the antecedent is a list conjunction of length n check that, for each $i < n$, the ith thing in the list is the ith thing emitted by the volcano and that the consequent is the nth thing emitted by the volcano. Accept iff this condition is satisfied.

■

[Suppose I am given a formula and i wish to know if it is an axiom of the decidable axiomatisation. I might have to wait for the volcano to emit n axioms from the semidecidable set, where n depends on the length of the candidate. How long might that take? One's first thought is that it might take a ridiculously long time. (Indeed it might). *So* long, in fact, that there is no computable bound on the time taken. But that doesn't follow. The question is not:

> (If f"$\mathbb{N}$ is not recursive can we bound time taken to learn that $x \in f$"$\mathbb{N}$ by a computable function applied to x? (1)

but

> How long does a volcano for f take to emit x values? Can we bound this time by a computable function applied to x? (2)

The answer to (1) is—obviously—"no", and for the usual reasons; the answer to (2) might be 'yes'!]

Miniexercise: Is this axiomatisation *independent*? [See exercise 9 on `https://www.dpmms.cam.ac.uk/study/II/Logic/2011-2012/LSqns2.pdf`]

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.

As alluded to at the start of section 4.5 the axioms for your system of revealed truth (*prima facie*) form a semidecidable set rather than a decidable set. There is of course a decidable axiomatisation but you have no way of knowing what it is. On the assumption that the semidecidable axiomatisation with which we start is not actually decidable (and the situation is not interesting otherwise) the decidable axiomatisation we obtain from it isn't very nice.

I suppose the point that is disquieting me is the thought that the *finite object* that is the Turing machine or register machine that guards the decidable axiomatisation is one that we can't locate in finite time. Or can we? It's finite, so we must have found it at some finite stage. It's just that we don't know when we've found it. We have lots of candidates of course but we never know when we have reached a stage when no revision is necessary. See exercise 56. A detailed discussion may be in order.

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.

n't know what a conservative ex-
is, so you'll have to tell them.

Proof: Omitted. ■[21]

(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...?

Exercise 59 *Since propositional logic is decidable, the set of falsifiable propositional formulæ over an alphabet is semidecidable, so it is a rectype. Give a presentation.*[22]

Exercise 60
Suppose $f : \mathbb{N} \to \mathbb{N}$ *is computable.*

Show that there a computable partial function g *s.t* $(\forall n \in \mathbb{N})(f(n)\downarrow \to f_{g(n)}(n)\downarrow)$.

Suppose further that $\{n : f(n)\downarrow\}$ *is semidecidable but not decidable.*

Show that there is no computable total function g *such that* $(\forall n \in \mathbb{N})(f(n)\downarrow \to f_{g(n)}(n)\downarrow)$.

4.7 The Undecidablity 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 $\mathfrak{M}$ that, on being given a natural number n, decoded it (using the primitive recursive unpairing functions alluded to on page

[21] I 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.

[22] I 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 pointless!

63) into `fst` n and `snd` n (n_1 and n_2 for short), and then$\downarrow= 0$ if the n_1th 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 $\mathfrak{M}^{\skull}$ 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_1th machine diverges on input n_2 (just as before) but diverges if the n_1th machine halts when given input n_2.

wrapping again

Front-end onto *this* machine a machine that accepts an input x and outputs $\texttt{pair}(x, x)$. We now have a machine $\mathfrak{M}^{\skull\skull}$ with the following behaviour.

> On being given n, it tests to see whether or not the nth 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_0th, say. What happens if we give it n_0 as input? Does it halt? Well, it halts iff the n_0th machine loops when given input n_0. But it *is* the n_0th machine itself!

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* $\{\texttt{pair}(p, d) : \{p\}(d)\downarrow\}$ *is*[23] *not decidable.*

Though it is obviously *semi*decidable! ■

Observe how absolutely critical it is in this proof that we can equivocate over the nature of natural numbers. A natural number can be an input to a program and it can be a code for a program... *and it can be both at the same time*. If you want to type your language so that every number variable ranges only over inputs to programs or only over codes for programs then you can't run this proof.

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

[23] I trust the overloading of the curly brackets does not wrong-foot the reader ... (!)

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.

It might be an idea to say a bit more about this fact ...

4.7.1 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,

$$\{e\}(b, a) = \{S(e, b)\}(a),$$

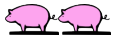
and so on for higher degrees (more parameters).

That is to say: currying, thought of as a function from gnumbers of functions to gnumbers 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 $\lambda b.\{e\}(a, b)$.

It's called the "S-m-n theorem" because in the general case the b could be an m-tuple and the a could be an m-tuple; the 'S' comes from the function in the statement. It probably has an official name but I have never known it[24].

In turn we get a corollary:



COROLLARY 2 *(The fixed point theorem).*

Let $h : \mathbb{N} \to \mathbb{N}$ be a total computable function. Then there is n such that $\{n\} = \{h(n)\}$[25]

[24] Wikipædia sez it's called the *parametrisation theorem* and was proved by Kleene.

[25] In case you are wondering, we of course mean the *graphs* of these two functions are equal ....

Proof: Consider the map

$$\texttt{pair}(e, x) \mapsto \{h(S(e, e))\}(x).$$

This is computable and is therefore computed by the ath 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

At this point I am going to put my hand up and admit that i learnt this next bit from the wikipædia article `https://en.wikipedia.org/wiki/Kleene%27s_recursion_theorem`. I should've learnt it years ago.

In section 3.1.2 we considered the justification of recursive definitions, and the removal of circularities. The second recursion theorem enables us to do that *internally*. Consider declarations of the form:

$f(0, \vec{x}) = g(\vec{x})$;
$f(\texttt{succ}n, \vec{x}) = h((f(n), n, \vec{x})$

There is a function that, on being given code for g and h (in the form of indices for two machines that compute g and h) will output an index for a function that obeys that recursion. Admittedly, this is sort-of obvious from Church's thesis. The extra information we get here is that if g and h are computable then so is f, and that code for f can be computed uniformly from code for g and h.

To be explicit, the way we use the second recursion theorem is as follows. We are trying to recursively define a function f—as it might be the example immediately above. The desired f is a fixed point for an operation H, which enjoys the following noncircular definition.

$H(f)(0, \vec{x}) = g(\vec{x})$;
$H(f)(\texttt{succ}n, \vec{x}) = h((f(n), n, \vec{x})$

H is total computable, so the fixed point theorem applies. So there is $a \in \mathbb{N}$ s.t. $\{a\}$ and $\{H(a)\}$ have the same graph. Then that graph is the graph that we want for our f.

There is a powerful corollary of the second recursion theorem 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 $\{n : Graph(\{n\}) \in A\}$ is not decidable.

Proof: Suppose χ, the characteristic function of $\{n : Graph(\{n\}) \in A\}$ is computable; we will deduce a contradiction.

Find naturals a and b so that $Graph(\{a\}) \in A$ and $Graph(\{b\}) \notin A$. If χ is computable the following function will also be computable:

$$g(n) := \texttt{ if } Graph(\{n\}) \in A \texttt{ then } b \texttt{ else } a$$

("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 $Graph(\{n\})$ in A? Let's assume it is, and derive a contradiction.

$Graph(\{n\}) \in A$	1)
$Graph(\{g(n)\}) \in A$	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. So we infer $Graph(\{n\}) \notin A$

But $Graph(\{n\}) \notin A$ also leads to contradiction:

$Graph(\{n\}) \notin A$	6)
$Graph(\{g(n)\}) \notin A$	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. So we infer $\neg\neg Graph(\{n\}) \in A$. But we proved $Graph(\{n\}) \notin A$ above. This gives a contradiction. So χ was not computable after all.

■

(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 61 *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 $f(m,n) = 0$.

Exercise 62 *Input a natural number n;*

If there are any machines that recognise the range of $\{n\}$ (thought of a set of strings over the alphabet $\{0,1\}$) then output a description of the minimal one; o/w loop forever.

Is this function computable?

4.8 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*

$A = \{e : \{e\}(e)\downarrow > 0\}$ *and*
$B = \{e : \{e\}(e)\downarrow = 0\}$

are recursively inseparable. That is to say, if f is a total function with f"$A = \{0\}$ and f"$B = \{1\}$, 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, $\{n\} \neq f$.

■

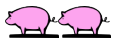
And here is another result with wider ramifications.

DEFINITION 18 *If $\sim$ is an equivalence relation on a set A we say f is a* **classifier** *for $\sim$ iff $(\forall x, y \in A)(f(x) = f(y) \longleftrightarrow x \sim y)$.*

EXERCISE 63 *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 $R \restriction A$ *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 $\{n\}\uparrow$ 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 Terwijn, [66], section 3.2.32. (pages 153–154).

Contrast this with the 2013 tripos question exercise 65 which follows.

4.9 Exercises

Exercise 64
The relational product (see p. 62) of two primitive recursive relations might not be a primitive recursive relation.

Exercise 65 *(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 66 *(*)*
Suppose $f : \mathbb{N}^k \to \mathbb{N}$ is total computable and increasing: $f(\vec{x}) > max(\vec{x})$. Show that there is a decidable $A \subseteq \mathbb{N}$ satisfying $f``A^k = \mathbb{N} \setminus A$.

Exercise 67 *(*)*
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 68 *(Mathematics Tripos IIA 1997 Paper 3 Question 8)(*)*
Does there exist a computable function f such that, for all m and n, if the mth 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 69

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 $(q_1, q_2, \ldots, q_k)$ 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*

$$(\frac{3}{5}, \frac{1250}{567}, \frac{1}{81}, \frac{875}{297}, \frac{5}{27}, \frac{3025}{63}, \frac{125}{9}, \frac{25}{6}, \frac{625}{3}, 35).$$

[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 70 *(*) 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 71
Show that the graph of a total computable function $f : \mathbb{N}^n \to \mathbb{N}$ *is a decidable subset of* $\mathbb{N}^{n+1}$*. Is the graph of a partial computable function decidable?*

Exercise 72

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) $\{e : |\{e\}\text{“}\mathbb{N}| = \aleph_0\}$*;*
   (ii) $\{e : \{e\}\text{“}\mathbb{N} \quad \text{strictly contains} \quad \mathrm{Dom}(\{e\})\}$.

Exercise 73 *(*)*

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 gnumber boxes and explain why the set of gnumbers of boxes that cannot tile the plane is a semidecidable set.

Hint: you will need König's Infinity Lemma.

Exercise 74

Let $f : \mathbb{N} \to \mathbb{N}$ *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* $\mathrm{Dom}(\{e\})$ *of the* e*th partial computable function is infinite, then* $\mathrm{Dom}(\{e\}) \cap A \neq \emptyset$ *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 75

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 76 *Show that the following sets are not decidable.*

(i) $\{e : \{e\}$ *everywhere undefined* $\}$. *(ii)* $\{e : \{e\}$ is total $\}$.
(iii) $\{e : \forall i < e.\, (\{e\}(i)\!\downarrow)\}$. *(iv)* $\{e : \forall i.\, (\{e\}(i)\!\downarrow \to i < e)\}$.

Which of the above sets R *and their complements* $\mathbb{N} \setminus R$ *are semidecidable?*

Exercise 77

Show that there is an $e \in \mathbb{N}$ *such that* $dom(\{e\}) = \{x : x > e\}$.
Show that there is an $e \in \mathbb{N}$ *such that* $dom(\{e\}) = \{x : \{x\}(e)\!\downarrow\}$.

In each case can one decide whether or not an index e *is of the given kind?*

Exercise 78
Suppose that $f, g : \mathbb{N}^2 \to \mathbb{N}$ 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)

Exercise 79 *(*)*

1. *Show that the range of an increasing total function $f : \mathbb{N} \to \mathbb{N}$ is a decidable set, and*
2. *conversely, that every decidable subset of $\mathbb{N}$ is the range of an increasing total computable function $\mathbb{N} \to \mathbb{N}$.*
3. *What if f is merely nondecreasing (but still total)?*
4. *What if f is increasing but perhaps not everywhere defined? (i.e., $(\forall n)(\forall m)(((n < m) \wedge f(n)\downarrow \wedge f(m)\downarrow) \to f(n) < f(m))$?)*
5. *What is the notion of "increasing function $\mathbb{N} \to \mathbb{N}^n$" that one would need were one to prove that every decidable subset of $\mathbb{N}^n$ is the range of an increasing computable function $\mathbb{N} \to \mathbb{N}^n$?*

Exercise 80 *(*)*
Show that every infinite semidecidable set has an infinite decidable subset.

Exercise 81 *(*) (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 82 *(*)*
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?[26]

By considering enumerations of the partial computable functions, find a partial computable function that cannot be extended to a total computable function.

Exercise 83 *(*)*
For a finite family $(A_0, A_1, \ldots, A_n)$ 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 $i \neq j$.

[26] See p 90 for definition of "acceptable".

(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).*

Exercise 84 *[marked by PTJ as HARD]*

A set $A \subseteq \mathbb{N}$ is called Diophantine *if there exists a polynomial $p(x, y_1, \ldots, y_n)$ with integer coefficients such that $x \in A$ if and only if there exist $y_1, \ldots, y_n$ such that $p(x, y_1, \ldots, y_n) = 0$. 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 85 *(*)*

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 5

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.

5.1 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 $A \to A$ is the identity $\mathbb{1}_A$. Any definable function $A \to A$ must commute with all permutations of A, and we all know that the centre of a symmetric group S is $\{\mathbb{1}_S\}$!

What about uniformly definable functions $(A \to A) \to (A \to A)$?

Well, there's obviously $K\mathbb{1}_{A\to A}$ and $\mathbb{1}_{(A\to A)\to(A\to A)}$. Then, for each n, there is the function that takes $f : A \to A$ and returns f^n. We need to explain why that's the lot.[1]

Exercise 86
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: $((A \to B) \to A) \to A$ is not a constructive thesis.

[1] As Julian Ziegler Hunt points out, there must be more to being a λ-term than having a denotation that is invariant under permutation of the base sets. After all, the function that *asks whether $f : A \to A$ is invertible and returns its inverse if it has one, and returns f otherwise* has no λ-term pointing to it, but it commutes with all permutations of A. The idea is to use invariance-under-permutations as a kind of *first check*.

5.2 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
$\lambda n.\lambda m.mn$

(Brief reality check: `succ` n is β-equivalent to `plus` n 1. `mult` similarly).

Don't ask

Wikipædia supplies: `pred`$\equiv \lambda n.\lambda f.\lambda x.n\ (\lambda g.\lambda h.h\ (g\ f))\ (\lambda u.x)\ (\lambda u.u)$

In lambda-talk the Church numerals are often written with underlinings: $\underline{n}$. This overloads the notation on p. 53 but doesn't actually violate its spirit.

In this development we will trade heavily on the *aperçu* that a function $\mathbb{N}^k \to \mathbb{N}$ defined by recursion can always be thought of as a fixed point for a function $(\mathbb{N}^k \to \mathbb{N}) \to (\mathbb{N}^k \to \mathbb{N})$.

Fixed point combinators.

$$Y : \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)).$$

$$(\lambda x.f(xx))(\lambda x.f(xx)) \twoheadrightarrow f((\lambda x.f(xx))(\lambda x.f(xx))).$$

Somewhat to my surprise I learnt recently (from [28]) that the set of [gnumbers of] λ-terms that are fixed-point operators is semidecidable. I find this fact 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 $(A \to B) \to A. \to (A \to B) \to B$.

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 87 *(*)*

`true` *is of type* $A \to (B \to A)$ *and* `false` *is of type* $A \to (B \to B)$*, so they are both of type* $A \to (A \to A)$.

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:

On this subject Wikipædia sez:

IsZero = $\lambda n.n$ ($\lambda x.$ false) true

The following predicate tests whether the first argument is less-than-or-equal-to the second:

LEQ = $\lambda m.\lambda n.$ IsZero (minus m n),

Because of the identity $x = y \equiv (x \leq y \wedge y \leq x)$, the test for equality may be implemented as

EQ = $\lambda m.\lambda n.$ AND (LEQ m n) (LEQ n m)

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`:= $\lambda p.p$ `true`
`snd`:= $\lambda p.p$ `false`
`nil`:= $\lambda x.$`true`

Check that `fst` (`pair` x y) = x and that `snd` (`pair` x y) = y:

`fst` (`pair` x y) =

($\lambda p.p$ `true`)($\lambda f.fxy$)

($\lambda f.fxy$)`true` =

`true` x y =

x

We obtain `snd` (`pair` x y) $= y$ similarly.

Exercise 88 *What are the types of these expressions? Discuss.*

Lists can be tho'rt 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.

$\texttt{NULL} := \lambda p.p(\lambda xy.\texttt{false})$

Write out the calculation

`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. Compscis 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 $f : \alpha \times \beta \to \beta$

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 a (ii) and a (iii) and gives back a β can be thought of as an α-list. So α-list is of type

$$(\forall\beta)(\beta \to (\alpha \times \beta \to \beta) \to \beta)$$

The empty list is thus $\lambda x_\beta.\lambda f_{(\alpha\times\beta\to\beta)}.x$ which is to say: K.

Exercise 89 *On this way of thinking about lists, what are* `cons` *and* `hd` *and* `tl`*?*

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 $\alpha * (\alpha\, list)$. 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:

$$\begin{aligned}
\alpha * \beta &=_{\text{def}} \forall\gamma\,((\alpha \to \beta \to \gamma) \to \gamma) \\
\mathit{makePair} &=_{\text{def}} \Lambda\alpha,\beta\,(\lambda x : \alpha, x' : \beta\,(\Lambda\gamma\,(\lambda f : \alpha \to \beta \to \gamma\ (f\,x\,x')))) \\
\mathit{first} &=_{\text{def}} \Lambda\alpha,\beta\,(\lambda p : \alpha * \beta\ (p\,(\lambda x : \alpha, x' : \beta\,(x)))) \\
\mathit{second} &=_{\text{def}} \Lambda\alpha,\beta\,(\lambda p : \alpha * \beta\ (p\,(\lambda x : \alpha, x' : \beta\,(x'))))
\end{aligned}$$

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 $\alpha\, list$ 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 $\lambda a : \texttt{int}, b : \texttt{int}\ (a + b)$ 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.

$$
\begin{aligned}
\alpha\, list &=_{\text{def}} \forall\beta\, (\beta \to (\alpha \to \beta \to \beta) \to \beta) \\
Nil &=_{\text{def}} \Lambda\alpha, \beta\, (\lambda x : \beta, f : \alpha \to \beta \to \beta\ (x)) \\
Cons &=_{\text{def}} \Lambda\alpha\, (\lambda x : \alpha, l : \alpha\, list\ (\Lambda\beta\, (\lambda x' : \beta, f : \alpha \to \beta \to \beta\ (f\, x\, (l\, \beta\, x'\, f))))) \\
listIter &=_{\text{def}} \Lambda\alpha, \beta\, (\lambda x : \beta, f : \alpha \to \beta \to \beta\ (\lambda l : \alpha\, list\ (l\, \beta\, x\, f)))
\end{aligned}
$$

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.

$$sum =_{\text{def}} \lambda l : int\, list\ (listIter\, int\, int\, 0\, (\lambda x : int, x' : int\ (x + x'))\, l)$$

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.

$$head =_{\text{def}} \Lambda\alpha\,(\lambda x : \alpha, l : \alpha\ list\ (listIter\,\alpha\,\alpha\,x\,(\lambda x' : \alpha, x'' : \alpha\,(x'')))\,l))$$
$$tailIter =_{\text{def}} \lambda x' : \alpha, l' : (\alpha\ list * \alpha\ list)\ (makePair\,(Cons\,x'\,(first\,l'))\,(first\,l'))$$
$$tail =_{\text{def}} \Lambda\alpha\,(\lambda x : \alpha\ list, l : \alpha\ list\ (second\,(listIter\,\alpha\,(\alpha\ list * \alpha\ list)\,(makeList\,Nil\,x)\,tailIter\,l)))$$

end of message from Toby

5.3 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 nth entry is the f of the nth entry of the list l. It has the recursive definition

$$\texttt{map}\,f\,x = \texttt{if null}(x)\,\texttt{then nil else}\,(f(\texttt{fst}(x)))::(\texttt{map}\ \ f(\texttt{snd}(\text{x})))$$

so it will be a fixed point for

$$\lambda mfl.(\texttt{if null}(l)\,\texttt{then nil else}\,(f(\texttt{fst}(l)))::(m\ f\ (\texttt{snd}(\text{l}))))),$$

namely

$$Y(\lambda mfl.(\texttt{if null}(l)\,\texttt{then nil else}\,(f(\texttt{fst}(l)))::(m\ f\ \texttt{snd}(\text{l}))))$$

Next we need the stream of naturals—which we may as well call '$\underline{\mathbb{N}}$'—and which is a fixed point for $\lambda l.(0::(\texttt{map suc } l))$. Then the stream of values of f is just $\texttt{map } f\ \underline{\mathbb{N}}$.

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 $n \mapsto \texttt{pair}(n, f(n))$ instead. (That is to say, the graph of f tho'rt of as a list of pairs). We then do `miaow` to it.[2]

$$\texttt{miaow}\,\underline{n}\,x = \texttt{if } (\texttt{snd}(\texttt{hd } x) = \underline{n})\,\texttt{then } (\texttt{fst } (\texttt{hd } x))\,\texttt{else miaow } n\,(\texttt{tl } x).$$

If we apply `miaow` $\underline{n}$ to the stream of pairs $\texttt{pair}(\underline{m}, f(\underline{m}))$ it returns the least $\underline{m}$ such that $f(\underline{m}) = \underline{n}$. 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 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.

5.4 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:

$$\lambda m.m(\lambda f n.nf(f\underline{1}))\texttt{suc}$$

Does any enthusiast feel like providing me with λ-terms for any of the Péter functions from exercise 44?

[2] Joke! Joke!!

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]:

```
datatype num = O
             | S of num

datatype bit = T | F

fun R u v O = 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 O (fn x => fn x' => x') n
fun SUB x y = R x (fn z => fn z' => PRED z) y
fun MUL x y = R O (fn z => fn z' => ADD x z) y
fun EXP x y = R (S O) (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 O)) (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 O))) (S(S O))

fun num 0 = O
  | num n = S(num (n-1))

fun mun O = 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 O
```

5.4.1 Combinators??

Exercise 90 *Prove that every λ term is equivalent to a combinator word in K and S.*

$T[x] \Rightarrow x$;
$T[E_1E_2] \Rightarrow T[E_1]T[E_2]$;
$T[\lambda x.E] \Rightarrow KT[E]$ (if 'x' does not occur free in E);
$T[\lambda x.x] \Rightarrow I$;
$T[\lambda x.\lambda y.E] \Rightarrow T[\lambda x.T[\lambda y.E]]$ (if 'x' occurs free in E);
$T[\lambda x.(E_1E_2)] \Rightarrow (ST[\lambda x.E_1]T[\lambda x.E_2])$ (if 'x' occurs free in E_1 or E_2)
Observe that every λ-term matches the LHS of precisely one of these rewrite rule.

Chapter 6

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.

6.1 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.

[1] It 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 19

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 $a \in A \cup \{1\!\!1_G\}$, 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 20

*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.*

[2]This 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 $f : X \to Y$ 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 20 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 91 *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 92 *Show that if G under some presentation is automatic, then it is automatic under any presentation.*

Free groups of finite rank are hyperbolic.

6.1.1 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.

6.1.2 Automatic theories

The language of propositional logic is not regular.

A theory is automatic if it has an axiomatisation that is automatic. The question always is: "Is this axiom scheme a regular language?"

Is any theory synonymous with an automatic theory synonymous? Or do we need a notion of automatic synonymy?

An automatic version of Kleene's theorem on finitely axiomatisable conservative extensions?

How on earth do we prove that the theory of algebraically closed fields is not automatic?

Which theories have automatic models?

Every decidable theory has a countable model for which the satisfaction predicate is decidable: given a formula with n free variables and an n-tuple from the structure can say whether or not that tuple bears that predicate.

$Th(\langle \mathbb{N}, \cdot \rangle)$ is decidable but not automatic and has no automatic model.

'You deal with the existential quantifier by the same trick that gives you a FDA from an FNA'

[what did I have in mind there?]

The canonical random graph is not automatic but it does have a decidable (and countably categorical!) theory.

For example the theory of fields of characteristic 0 is clearly automatic. Careful! Khoussainov 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 45 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 93 *(*) Show that the intersection of two automatic theories is automatic.*

6.2 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 51 on p. 98.)

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 21
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 $<_R \restriction \{m \in \mathbb{N} : m <_R n\}$ 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 *non*recursive 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.* $otp(R \restriction X) = \omega$*.*

Proof: Recycle the usual "picking winners" proof that countable limit ordinals have cofinality ω. It works in this context. We enumerate the members of X in increasing order $x_0, x_1 \ldots$. 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 79). ■

Observe that this proof is effective: there is a computable function which, on being given the gnumber of a characteristic function of a wellordering of $\mathbb{N}$, returns the gnumber of the characteristic function of an unbounded subset of length ω.

Exercise 94

The class of recursive ordinals is closed under the Doner-Tarski function f_α (see definition 9 p. 55) for every recursive ordinal α.[3]

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 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 95 *(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 96 *(*)*

Construct a recursive counterexample to König's Infinity lemma: a recursive finitely-branching tree with no infinite recursive path.

6.3 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

[3] Come to think of it i'm not really entirely happy about this ... but Stan says it's obvious so it must be OK

natural numbers, which holds when the nth 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 $\{m :$ the mth 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.

$$\begin{array}{ll} f(0) & = 0; \\ f(n+1) & = \texttt{if}\ \ P(n)\ \ \texttt{then}\ f(n) \oplus 2^n\ \texttt{else}\ f(n). \end{array}$$

Of course, if we prefer coding with primes then we obtain a different definition.

Now we find, in $\mathfrak{M}$, that $\{n : n$ 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 *un*decidable 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 with carrier set $\mathbb{N}$ *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 "$n\,E\,m$" 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:

$$\begin{array}{ll} f(0) & = 0; \\ f(n+1) & = \texttt{let } n+1 = \langle p,i,t\rangle \texttt{ in} \\ & \quad \texttt{if } \{p\}_t(i)\!\downarrow \texttt{ then } f(n) \oplus 2^{\langle p,i\rangle} \texttt{ else } f(n). \end{array}$$

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. For example, let T be the theory PA + $\neg$Con(PA). T is consistent but unsound. Since PA $\subseteq T$ and $T \vdash \neg$Con(PA) then certainly $T \vdash \neg$Con(T). T is recursively axiomatisable so the predicate "...is a T-proof" is decidable. Now let P be the program which, on being given an input i, examines [in order] all numbers bigger than i and stops when it finds one that encodes a T-proof of $\neg$Con(T). P clearly halts on all standard inputs, and never halts in standard time, since all proofs of $\neg$Con(T) are of nonstandard length.

It simply isn't true that a standard p*, applied to a standard* i*, halts in standard time if it halts at all.*[4]

So this ruse doesn't prove the promised theorem of Tennenbaum. Nevertheless it does give us a flavour of the proof that we will eventually end up with, and it does prove a weaker version:

REMARK 14 *If* $\mathfrak{M}$ *is a nonstandard model of true arithmetic with carrier set* $\mathbb{N}$ *then the graphs of* $+$ *and* $\times$ *cannot both be decidable.*

Proof: The only thing that prevented the preceding discussion from being a proof of Tennenbaum's theorem was the possibility of a Turing machine with standard gnumber taking a nonstandard number of steps to halt on some standard input. But if $\mathfrak{M}$ models true arithmetic then this cannot happen. If true arithmetic proves that there is an n s.t. $p(i)$ halts in n steps then there is some

[4] Tho' it is of course true that every *concrete* p applied to a *concrete* i halts in finite (indeed *concrete*) time if it halts at all!!

n such that true arithmetic proves that $p(i)$ halts in n steps, and this n is of course standard. ■

In fact we can indeed encode some undecidable sets even without this extra condition on $\mathfrak{M}$—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 97 *(Scott)(*)*
If $\mathfrak{M}$ is a nonstandard model of PA and $\mathfrak{M}^*$ the corresponding structure for second-order arithmetic, then $\mathfrak{M}^* \models WKL$.

We procede 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

$$(\forall n < \underline{m})(\forall y < \underline{m})(\forall x < \underline{m})(\neg A(n,y) \vee \neg B(n,x)) \tag{1}$$

for any numeral $\underline{m}$. (Recall from p. 53 that $\underline{m}$ is the string $\overbrace{S(S(S\ldots 0\ldots))}^{m \text{ times}}$.)

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

$$(\forall n < m)(\forall y < m)(\forall x < m)(\neg A(n,y) \vee \neg B(n,x)). \tag{C}$$

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:

$$\mathfrak{M} \models (\forall n < e)(\forall y < e)(\forall x < e)(\neg A(n,y) \vee \neg B(n,x)). \tag{D}$$

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 PA $\models_R$ Con(PA) or PA $\models_R \neg$Con(PA).

The second is false, because PA has a model, so we infer PA $\models_R$ Con(PA).

It's a pretty safe bet that $\mathcal{L}$ is not axiomatisable!

6.4 Recursive Saturation

This section is a **stub**

The models of Presburger arithmetic that can be expanded to models of PA are precisely the recursively saturated ones.

6.5 Leftovers

Any connection between Quantifier elimination and automaticity? Presburger is a theory with signature $\langle \mathbb{N}, +, \leq, 0, 1\rangle$. It does not have QE. However, if we add, for each concrete k, a unary predicate is-divisible-by-k we have a theory that *does* have QE. There is also, for each concrete k, a unary function `div` k. This is in [42].

Observe that there is a 2-state Mealy machine that adds two binary strings. Its alphabet is $(\{0,1\}\times\{0,1\})\cup\{\texttt{EOF}\}$. It has two states: `carry` and `don't-carry`. The initial state is `don't-carry`, and its transition table is

If in state	and reading	go to state	and emit
`carry`	$\langle 0,0\rangle$	`don't-carry`	1
`carry`	$\langle 0,1\rangle$	`carry`	0
`carry`	$\langle 1,0\rangle$	`carry`	0
`carry`	$\langle 1,1\rangle$	`carry`	1
`carry`	`EOF`	`don't-carry`	1

`don't-carry`	$\langle 0, 0\rangle$	`don't-carry`	0
`don't-carry`	$\langle 0, 1\rangle$	`don't-carry`	1
`don't-carry`	$\langle 1, 0\rangle$	`don't-carry`	1
`don't-carry`	$\langle 1, 1\rangle$	`carry`	0
`don't-carry`	`EOF`	`don't-carry`	`null`

Take a moment or two to think about the challenge of designing a Mealy machine to *multiply* two bit strings.

Mind you, we didn't define *automatic structure* in terms of Mealy machines but rather in terms of FSAs. So what one should really be doing is defining a finite state machine whose alphabet is $\{0, 1, \texttt{EOF}\}^3$. It will have three ports not two, and it will have an accepting state which it reaches if the string of entries in the third port is the sum of the string of entries in the first two ports. (It will also need a `fail`' state. The reader might like to supply details of this machine.)

Khoussainov sez: Product of automatic structures is automatic.
The first-order theory of an automatic structure is decidable.
Presburger arithmetic is automatic, which is why it's decidable.
$\langle \mathbb{N}, \cdot\rangle$ not automatic.

Khoussainov sez: if I have an automatic structure with some operations, then if I apply the operations n times to get terms of depth n, then there are only exponentially many (instead of doubly exponential—be alert to the difference between the *length* of a word and the *depth* of a word).

The problem of finding whether or not two automatic structures are isomorphic is as hard as the same problem for recursive structures.

Khoussainov says there is a good notion of automatic isomorphism.

Khoussainov sez $\langle \mathbb{Z}, +\rangle$ is automatic, and that there are uncountably many countable total orders elementarily equivalent to it—and they can't all be automatic. $\langle \mathbb{Q}, +\rangle$ is automatic.

If you represent natural numbers in binary then "x is a power of 2" is regular. If you write them in ternary, it ain't! So automaticity is not a property of the predicate but of its representation.

Automatic Groups

A good place to start reading about automatic groups would be Baumslag's review [5] of [21]. There is also a Wikipædia article.

An automatic group is a (countable) group serviced by certain FDAs. What might these machines be? An obvious place to start looking for a machine is the **Cayley Graph**[5] for a group-with-a-presentation, since it is quite literally a machine even as it stands (albeit an infinite machine).

[5]The Cayley Graph for a group-with-a-presentation is a directed graph whose vertices are the elements of the group, and where the directed edges are labelled by generators. Multiply the group element that is the vertex at the fletch of the arrow by the generator that labels the arrow to obtain the group element that is the vertex at the point of the arrow.

We aren't going to take the machine to be the Cayley graph—that way the only automatic groups would be finite groups, and that is not what we want. What we want is a way of understanding how an infinite machine might secretly be merely a finite machine written out many times.

Let GEN be an alphabet of letters for generators (and their inverses!) for a group G. We will say what is is for G to be automatic.

Richard Parker says:

Little note. The correct starting point seems to be a set of generators that are closed under inversion. Therefore if a generator has order two, you can cope with this in the generators and do not need the square of the generator to be an extra relator. I therefore call a group an "inv-tab" group (Inverse-Table group) if it is the free product of free groups and (cyclic) groups of order 2.

Since GEN generates G there is an evaluation map $\sigma : GEN^* \twoheadrightarrow G$. Of course there might be languages $L \subset GEN^*$ such that the restriction $\sigma \restriction L$ is onto the whole of G. If even one of these L is regular then we have satisfied one part of the definition of an automatic group.

One condition is obviously going to be that the word problem for the group is solvable by an FSA. How do we know that this constraint is nontrivial? Are there any group presentations for which the word problem is not solvable by an FDA? Yes. There are even group presentations for which the word problem is not solvable at all. This is not obvious. It's pretty obvious that there are semigroup presentations for which the word problem is unsolvable, and one establishes this by coding up the halting problem for a Turing Machine as a word problem for a semigroup presentation[6] but the introduction of the inversion operator greatly complicates matters, and it is a tricky task to produce a group presentation for which the word problem is not solvable.

How might an FSA solve the word problem for a group? If we feed it two words w_1 and w_2 simultaneously then it ends up in an accepting state? If w_1 and w_2 are the same length then one can imagine a machine that eats characters from the alphabet $GEN \times GEN$. When it has finished reading the string of pairs-of-characters we want it to be in an accepting state. That's OK if w_1 and w_2 are the same length. To accommodate the need to answer questions about equality between words of different lengths we have to include the unit $\mathbb{1}$ as one of the generators.[7]

Richard Parker says:

There is also a generalization of FSA where the state includes information as to which word it wants a generator from next. I prefer that approach myself. Holt's main programs actually only work with cases where the input is taken one-letter-from-each-word, and as far as I know this is a genuine restriction. You cannot try all strings of identity elements in all places in a finite time and I do not think the two approaches are equivalent.

[6] You might like to try this, Dear Reader!

[7] That's not how it's done in the literature but Benson Farb thinks it's OK.

Take a cartesian product $\mathcal{G}$ of two copies of the Cayley graph. The result is a graph whose vertices are ordered pairs of group elements, and the edges are labelled with ordered pairs from GEN. This is the basis for the FDA that will solve the word problem. The ordered pairs are to be tho'rt of as states and every "diagonal" vertex $\langle x, x\rangle$ as an accepting state. (Any path through this product that leads to a state $\langle x, x\rangle$ represents a pair of words in the generators that both evaluate to the group element x.)

The key thought now is that all the diagonal vertices can be thought of as *the same* accepting state. (This is because groups have an inverse operation ...) Identifying all the diagonal vertices induces an equivalence relation on elements of $\mathcal{G}$ as follows.

We seek a binary "resemblance" relation on vertices satisfying the condition that for all x_1, x_2, x_3 and x_4, vertex $\langle x_1, x_2\rangle$ "resembles" vertex $\langle x_3, x_4\rangle$ iff, for every character χ from $GEN \times GEN$, the vertex $\langle y_1, y_2\rangle$ that we reach from $\langle x_1, x_2\rangle$ by travelling along an edge labelled 'χ' "resembles" the vertex $\langle y_3, y_4\rangle$ that we reach from $\langle x_3, x_4\rangle$ by travelling along the edge labelled 'χ. That is to say, if we think of elements of $GEN \times GEN$ as one-place operations, our resemblance relation must be a conguence relation for all of them.

Richard Parker says:

Not quite got my head around this. As far as I know, your group has to have the "fellow traveller" property, which means that the set of all nice words for group elements $A \cdot B^{-1}$ for pairs of nice words that are the same group element is a finite set. I think that's right.

The relation we desire is clearly a fixed point for the operation + defined as follows:

$$(\forall p_1 p_2 \in \mathcal{G})(p_1 \sim^+ p_2 \iff (\forall \chi \in GEN \times GEN)(\chi(p_1) \sim \chi(p_2))).$$

This + operation is clearly monotone and there are various fixed-point theorems we can use to obtain a fixed point for it.

Miniexercise: which fixed-point theorem are you going to use? Tarski-Knaster[8] or Bourbaki-Witt?

The equivalence relation we start with makes all diagonal pairs $\langle x, x\rangle$ equivalent and otherwise makes all other ordered pairs equivalent only to themselves. Then we apply + repeatedly until we reach a fixed point.

If there is a fixed point that is (an equivalence relation) of finite index then we are happy. The quotient will be a machine with alphabet $GEN \times GEN$.

Check that if a group hosts a machine of this kind that can detect when two words are equivalent then it also hosts a machine that recognises the set of words that evaluate to the unit. The converse is not obvious. (I don't know if it's even true)

Exercise 98 *Show that if G under some presentation is automatic, then it is automatic under any presentation.*

[8]Tarski-Knaster was not lectured at Part II in 2019/20, but there are proofs in the course materials from previous years.

[RP sez: Replace the generators of one presentation for words in the other group that they are equal to.]

Farb sez that [26] uses the pumping lemma to show that finitely generated infinite torsion groups (no elements of infinite order) are not automatic. Khoussainov sez: shurely shome mishtake? Product of lots of copies of S_2 is automatic (in his sense). Is an infinite torsion group an infinite locally finite group?

What is the conection between frieze groups and automatic groups? Is the Cayley graph of a frieze group a frieze? We presumably need in this context (showing that certainly infinite groups are automatic by looking at their Cayley graphs) whatever device we used to extract a finite description of a frieze.

Braid groups are hyperbolic and Thurston-automatic.

Presumably the free product of two automatic groups is automatic?

finitely generated group is K-automatic iff it's abelian-by-finite.

Grigorchuk group is automatic but not finitely presented. It's interesting because it has intermediate growth. Not automatic in Khoussainov's sense, not known if it's Thurston-automatic.

$\mathbb{Z} \times \mathbb{Z}$ is finitely generated but not hyperbolic.

Free groups of finite rank are hyperbolic.

We really should prove this

Hyperbolic groups are automatic.

A group is virtually free iff$_{df}$ it has a subgroup of finite index that is free of finite rank. (Exercise: can require the free subgroup to be normal, using Nilsen-Schreier. Richard sez: consider the action of the group on the set of cosets, then the kernel of this action). A group is virtually free iff its word problem can be solved by a PDA.

Chapter 7

Incompleteness

7.1 Proofs of Totality

I emphasised on p 72 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 111) 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? (And we would have to be able to do that if $\{n : (\forall m)(\{f(n)\}(m)\downarrow)\}$ were semidecidable). 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 to be told that there is in fact *no way at all* of doing it—in finite time.

Consider the total computable function f that we hypothesised in the proof of theorem 18. What happens if we assume that f"$\mathbb{N}$ is not the whole of $\{n : (\forall m)(\{f(n)\}(m)\downarrow)\}$ but merely a subset of it? Then the construction in the proof exhibits a total computable function not in the range of f.

This property of $\{n : (\forall m)(\{f(n)\}(m)\downarrow)\}$ is important enough to deserve a name...

Definition 22
Suppose $X \subseteq \mathbb{N}$ is not only not semidecidable but also comes equipped with a computable function g which "diagonalises out of" any semidecidable $\{n\}$"$\mathbb{N} \subseteq X$ in the sense that $g(n)$ is a member of $X \setminus \{n\}$"$\mathbb{N}$.

Then we say X is **productive**.

It can be hard to see whether or not a function specification specifies a function with the same graph as a recursive specification (recall Waring numbers). However it is mechanical to check whether or not a piece of syntax is *literally* a definition of a computable function.

That's the easy part; 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.

7.2 A Theorem of Gödel's

Definition 23 *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. 143 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.

7.2.1 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 $\{\underline{n}\}$ is total. For each $k \in \mathbb{N}$, we take $V(k)$, which will be a pair $\langle p, n\rangle$. We then compute $\{\underline{n}\}(k) + 1$ 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

$$\lambda k \in \mathbb{N}.(\texttt{let } \langle p, n\rangle = V(k) \texttt{ in } \{\underline{n}\}(k) + 1)$$

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 the T-`bad-function` is total?

We have to prove the following:

> *For every n, if we take the nth 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 nth 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 nth program is total*, but *whether or not T proves that it is total.*

Observe 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 than 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 **??**, 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 $\{n \in \mathbb{N} : T \vdash$ "$\{n\}$ 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. There is nothing to stop us adding ϕ as an axiom whenever T proves that $T \vdash \phi$.

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 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 $\mathcal{A} \setminus T$.

Clearer if we appeal to Gödel's argument

So what we proved above can be stated as:

Remark 15 *The set of (gnumbers of) arithmetic truths is productive.*

■

Indeed $\{n : \{n\} \text{ is total}\}$, the set that kicked off this chapter, 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 complement of the halting set, the set $\{\langle p, i\rangle : \{p\}\!\downarrow (i)\}$.

Exercise 99 *(*)*
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 100 *(*)*
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 arithmétic 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 **??**.

7.3 Undecidablity 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 falsifiablity 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 undecidablity 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 `HALT`ed. 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.

7.4 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:

$$\phi \text{ is true in all finite structures} \quad \text{iff} \quad \mathfrak{M}(n)\!\uparrow. \tag{A}$$

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 `HALT`s. 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 $\to$ 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 $\to$ 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 $\mathfrak{N}$. 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 101 *(*)*

Will the same argument show that the set A of formulæ 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 101, 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.

7.5 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 8

Notes and Appendices

8.1 Chapter 2

8.1.1 Horn clauses in rectype declarations

This illustration comes from Ben Millwood.

. . . we can declare a datatype C equipped with a constructor `con`: $(C \to \emptyset) \to C$. Now, by recursion, declare a function f defined on this datatype by:

$$f(\texttt{con}(g)) = g(\texttt{con}(g))$$

This is a legitimate (if degenerate) declaration of f by recursion.

Do some type-checking. . . g must be of type $C \to \emptyset$ since it is an argument to `con`; so $\texttt{con}(g)$ is of type C, and $f(\texttt{con}(g))$ 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 $C \to 0$, so then $\texttt{con}(f)$ is a legitimate element of C. So C can't be empty.

You can look at the declaration $\texttt{con} : (C \to 0) \to C$ as saying:

- if C is uninhabited, then there is a function $h : C \to 0$, but then $\texttt{con}(h) : C$, so C is inhabited,
- if C is inhabited, say by $x : C$, then C only has one constructor, so $x = \texttt{con}(h)$ for some $h : C \to 0$. 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, $\texttt{don} : \mathcal{P}(D) \to D$, where $\mathcal{P}$ is the powerset. Then clearly D is inhabited, since $\texttt{don}(\emptyset)$ 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 rectype 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 rectype 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.

8.1.2 Infinitary Languages

While we are on the subject of infinitary languages let's have the following morsel:

THEOREM 22 *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 $L_{\omega_1\omega}$ 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 ismorphism $\mathfrak{B} \simeq \mathfrak{A}$.

For each tuple $a_1 \dots a_n$ from A, and every $\beta < \omega_1$, we define a formula $\phi^{\beta}_{a_1 \dots a_n}(x_1 \dots x_n)$ by recursion on β as follows:

$\beta = 0$:

$\phi^{0}_{a_1 \dots a_n}(x_1 \dots x_n)$ 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 $\phi^{\beta}_{a_1 \dots a_n}(x_1 \dots x_n)$ is $\bigwedge_{\gamma < \beta} \phi^{\gamma}_{a_1 \dots a_n}(x_1 \dots x_n)$

$\beta = \alpha^+$:

This is where the work gets done. $\phi^{\alpha+1}_{a_1\ldots a_n}(x_1 \ldots x_n)$ is the conjunction

$$\phi^{\alpha}_{a_1\ldots a_n}(x_1 \ldots x_n) \qquad \wedge$$

$$\bigwedge_{a_{n+1}\in A} (\exists x_{n+1})(\phi^{\alpha}_{a_1\ldots a_{n+1}}(x_1 \ldots x_{n+1})) \qquad \wedge$$

$$(\forall x_{n+1}) \bigvee_{a_{n+1}\in A} (\phi^{\alpha}_{a_1\ldots a_{n+1}}(x_1 \ldots x_{n+1})).$$

(Both these clauses look like infinitary $\forall\exists$)

Reality check:

for all tuples $a_1 \ldots a_n$ and all $\beta < \omega_1$,

1. the formula $\phi^{\beta}_{a_1\ldots a_n}$ has at most the free variables 'x_1' ...'x_n'; and
2. $\mathfrak{A} \models \phi^{\beta}_{a_1\ldots a_n}[a_1 \ldots a_n]$;
3. $\mathfrak{A} \models (\forall x_1 \ldots x_n)(\phi^{\beta}_{a_1\ldots a_n} \rightarrow \phi^{\gamma}_{a_1\ldots a_n})$ 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 $\bigwedge_{a_{n+1}\in A} (\exists x_{n+1})(\phi^{\alpha}_{a_1\ldots a_{n+1}}(x_1 \ldots x_{n+1}))\wedge$ is satisfied by instantiating 'x_{n+1}' to 'a_{n+1}'. The third conjunct is satified similarly because we can take a_{n+1} to be x_{n+1}

We are now in a position to prove theorem 22.

Observe that—by (3)—for each tuple $a_1 \ldots a_n$ from A and for every tuple $x_1 \ldots x_n$ the truth-value of $\phi^{\beta}_{a_1\ldots a_n}(x_1 \ldots x_n)$ 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 $a_1 \ldots a_n$ with tuple $x_1 \ldots x_n$ we can associate the ordinal at which the truth-value of $\phi^{\beta}_{a_1\ldots a_n}(x_1 \ldots x_n)$ *settles down.* Fix $a_1 \ldots a_n$. There are only countably many tuples $x_1 \ldots x_n$ so there are only countably many such ordinals. ω_1 is regular, so, for each tuple $a_1 \ldots a_n$, there will come a stage by which the truth-values of $\phi^{\beta}_{a_1\ldots a_n}(x_1 \ldots x_n)$ have settled down for all $x_1 \ldots x_n$. Again, there are only countably many tuples $a_1 \ldots a_n$, 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:

$$\phi^{\alpha}_{\emptyset} \wedge \bigwedge_{\substack{n<\omega \\ a_1\ldots a_n\in A}} (\phi^{\alpha}_{a_1\ldots a_n}(x_1 \ldots x_n) \rightarrow \phi^{\alpha+1}_{a_1\ldots a_n}(x_1 \ldots x_n))$$

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.

$$(\forall a_{n+1} \in A)(\exists b_{n+1} \in B)(\mathfrak{B} \models \phi^{\alpha}_{a_1 \dots a_{n+1}}(b_1 \dots b_{n+1})) \quad (1)$$

and

$$(\forall b_{n+1} \in B)(\exists a_{n+1} \in A)(\mathfrak{B} \models \phi^{\alpha}_{a_1 \dots a_{n+1}}(b_1 \dots b_{n+1})) \quad (2)$$

(1) holds because $\mathfrak{B} \models \phi$, so $\mathfrak{B} \models (\exists x_{n+1})\phi^{\alpha}_{a_1 \dots a_{n+1}}(b_1 \dots b_n, x_{n+1})$.

To show (2) we use again the fact that $\mathfrak{B} \models \phi^{\alpha+1}_{a_1 \dots a_n}(b_1 \dots b_n)$. This gives $\mathfrak{B} \models (\forall x_{n+1}) \bigvee_{a_{n+1} \in A} \phi^{\alpha}_{a_1 \dots a_{n+1}}(b_1 \dots b_n, x_{n+1})$ whence, for some $a_{n+1} \in A$, $\mathfrak{B} \models \phi^{\alpha}_{a_1 \dots a_{n+1}}(b_1 \dots b_{n+1})$.

8.2 Chapter 3

8.2.1 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.

$proj^2_1(x, y)$ 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

$$x \mapsto f(x, proj^2_1(x, 0)).$$

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

$$x \mapsto f(x, proj^2_1(x, z(x)))$$

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 $g_1, \ldots g_m$ to be the n-ary function which takes arguments $(x_1, \ldots x_n)$ and returns

$$f(g_1(x_1, \ldots x_n), \ldots g_m(x_1, \ldots x_n))$$

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.

8.2.2 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

$\langle n, m\rangle \lhd \langle n, m+1\rangle$;
$\langle m, A(m+1, n)\rangle \lhd \langle m+1, n+1\rangle$

The point being that to get the induction to work we need

$$\langle m, A(m+1, n)\rangle \lhd \langle m+1, n+1\rangle$$

but we don't need

$$\langle m, A(m+1, n)+1\rangle \lhd \langle m+1, n+1\rangle$$

or

$$\langle m, A(m+1,n)\rangle \lhd \langle m+1, n\rangle.$$

But one needs to know that this relation $\lhd$ 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 53.

> 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 $S(\ldots S(0)\ldots)$, but it is "meta-generated" by a variable of the theory in which we phrased our theory (aka the metatheory). So in the metatheory, $\underline{\text{n}}$ 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 125, 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 "`snd`(hd x) = $\underline{n}$", you are expressing that in the theory of lambda calculus, we (somehow) test for equality of the lambda term "snd (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 := $\lambda nm.$ and (iszero($m$pred($n$succ0)))(iszero($n$pred($m$succ0)))

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.

8.3 Chapter 4

8.4 Chapter 5

8.5 Chapter 6

8.6 Chapter ??

8.7 Chapter ??

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 mathematique **38** (1992) pp 291–313. Also at `http://retro.seals.ch/digbib/view?rid=ensmat-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 Grzegorczyk "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 Hofstader "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 9

Answers to selected questions

Chapter 2: Recursive Datatypes

Exercise 4

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 '$\bot$'. 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*[1] 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 29 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]

> *Contructors 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 (LaTeX: `\ulcorner`) and closed by the '$\urcorner$' symbol (LaTeX: `\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:

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

"

[or *something* like that, with the corners placed correctly!]

"Define a translation which eliminates disjunction from Boolean expressions by the following recursion:

[1] I have italicised this word because the metaphor is a good one: google *referential transparency*.

$tr(a) = a$, $tr(\top) = \top$, $tr(\bot) = \bot$,
$\ulcorner tr(A \wedge B) = tr(A) \wedge tr(B)$,
$tr(A \vee B) = \neg(\neg tr(A) \wedge \neg tr(B))$,
$tr(\neg A) = \neg tr(A) \urcorner$.

Prove by structural induction on Boolean propositions that

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

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 formulæ 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 $|\ulcorner tr(\neg A) \urcorner|$? $\ulcorner tr(\neg A) = \neg tr(A) \urcorner$ so $\ulcorner |tr(\neg A)| = |\neg tr(A)| \urcorner$, and $|\ulcorner \neg tr(A) \urcorner|$ is $|tr(A)| + 1$ which is certainly $\ \leq \ 3 \cdot |\ulcorner \neg A \urcorner|$.

$\wedge$ 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 $\ulcorner tr(A) \wedge tr(B) \urcorner$. By induction hypothesis $|tr(A)| \ \leq \ 3 \cdot |A| - 1$ and $|tr(B)| \ \leq \ 3 \cdot |B| - 1$ so $\ulcorner |tr(A) \wedge tr(B)| \urcorner \ \leq \ (3 \cdot |A| - 1) + (3 \cdot |B| - 1) + 1$. The final '+1' is for the '$\wedge$'. This rearranges to
$|\ulcorner tr(A) \wedge tr(B) \urcorner| \ \leq \ 3 \cdot (|A| + |B|) - 1$
but $|A| + |B| \ \leq \ |\ulcorner A \wedge B \urcorner|$ whence
$\ulcorner |tr(A) \wedge tr(B)| \ \leq \ 3 \cdot (|A \wedge B|) - 1 \urcorner$ and finally
$\ulcorner |tr(A \wedge B)| \ \leq \ 3 \cdot (|A \wedge B|) - 1 \urcorner$.

$\vee$ 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 $|tr(A)| + |tr(B)|$ + one 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
$\ulcorner |tr(A \vee B)| \ \leq \ (3 \cdot |A| - 1) + (3 \cdot |B| - 1) + 4 \urcorner$. We can rearrange this to
$\ulcorner |tr(A \vee B)| \ \leq \ 3 \cdot (|A| + |B|) - 1 - 1) + 4 \urcorner$ and further to
$\ulcorner |tr(A \vee B)| \ \leq \ 3 \cdot (|A| + |B|) + 2 \urcorner$.

Now $|A| + |B| \ = \ulcorner |A \vee B| \urcorner - 1$ so we can substitute getting

$\ulcorner |tr(A \vee B)| \ \leq \ 3 \cdot (|A \vee B| - 1)) + 2 \urcorner$ and rearrange again to get

$\ulcorner |tr(A \vee B)| \ \leq \ 3 \cdot |A \vee B| - 1 \urcorner$ as desired.

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 14

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 Kfinite set is Nfinite.

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 Nfiniteness 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 Nfinite by induction hypothesis, because equality between its members is decidable) or the disjoint union of a and $\{x\}$, which is Nfinite by definition of Nfiniteness. That completes the proof."

Exercise 15

Remark 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.

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

$$X \subseteq f(A \setminus \{f(A)\}) \to X \in A \setminus \{f(A)\}$$

which is

$$X \subseteq f(A \setminus \{f(A)\}) \to X \in A \wedge X \neq f(A)$$

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:

$$X \subseteq f(A \setminus \{f(A)\}) \to X \in A$$

$f(A \setminus \{f(A)\}) \subseteq f(A)$ 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 $\mathcal{P} \circ f$: $(\forall x)(x \subseteq f(A) \to x \in A)$. Now specialise '$x$' to '$f(A)$' to obtain $f(A) \subseteq f(A) \to f(A) \in A$), which tells us that $f(A) \in A$ after all. ■

Note that the only set-theoretic axiom we have used is subscission.

Exercise 16

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 $M = \bigcap\{Y : m \in Y \wedge P\text{“}Y \subseteq Y\}$. (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 $m \in Y \wedge P\text{“}Y \subseteq Y$ 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 1

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:

Let
$XY = \forall xy(R'(x,y) \rightarrow R(x,y))$,
$Y = \forall y(R(y,x) \rightarrow \phi(y))$,
$Y' = \forall y(R'(y,x) \rightarrow \phi(y))$ *and*
$Y'_z = \forall y(R'(y,z) \rightarrow \phi(y))$.

We want to prove

$$(\forall x(Y \rightarrow \phi(x)) \rightarrow \forall z(\phi(z))) \wedge XY \rightarrow (\forall x(Y' \rightarrow \phi(x)) \rightarrow \forall z(\phi(z)))$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z), R(y,x), R'(y,x)}\,(b) \quad \dfrac{}{R(y,x), R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z), R(y,x)}\,(b)}{R'(y,x) \rightarrow R(y,x), R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z), R(y,x)}\,(\rightarrow l)}{\forall xy(R'(x,y) \rightarrow R(x,y)), R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z), R(y,x)}\,(2 \times \forall l)}{XY, R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z), R(y,x)}\,(expand)}{(****)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(***)}{XY, R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z), R(y,x)} \quad \dfrac{}{XY, \phi(y), R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z)}\,(b)}{XY, R(y,x) \to \phi(y), R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z)}\,(\to l)}{XY, \forall y(R(y,x) \to \phi(y)), R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z)}\,(\forall l)}{XY, Y, R'(y,x), R'(y,z) \vdash \phi(x), \phi(y), \phi(z)}\,(expand)}{XY, Y, \vdash \phi(x), \phi(z), R'(y,x) \to \phi(y), R'(y,z) \to \phi(y)}\,(2\times \to r)}{XY, Y, \vdash \phi(x), \phi(z), \forall y(R'(y,x) \to \phi(y)), \forall y(R'(y,z) \to \phi(y))}\,(2 \times \forall r)}{XY, Y, \vdash \phi(x), \phi(z), Y', Y'_z}\,(expand)}{(***)}$$

$$\dfrac{\dfrac{\dfrac{(***)}{XY, Y, \vdash \phi(x), \phi(z), Y', Y'_z} \quad \dfrac{}{XY, Y, \phi(x), \phi(z) \vdash \phi(x), \phi(z)}\,(b)}{XY, Y, Y'_z \to \phi(z) \vdash \phi(x), \phi(z), Y'}\,(\to l)}{(**)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(**)}{XY, Y, Y'_z \to \phi(z) \vdash \phi(x), \phi(z), Y'} \quad \dfrac{}{XY, Y, \phi(x), Y'_z \to \phi(z) \vdash \phi(x), \phi(z)}\,(b)}{XY, Y, Y' \to \phi(x), Y'_z \to \phi(z) \vdash \phi(x), \phi(z)}\,(\to l)}{XY, Y, \forall x(Y' \to \phi(x)) \vdash \phi(x), \phi(z)}\,(2 \times \forall l)}{XY, \forall x(Y' \to \phi(x)) \vdash \phi(z), Y \to \phi(x)}\,(\to r)}{XY, \forall x(Y' \to \phi(x)) \vdash \phi(z), \forall x(Y \to \phi(x))}\,(\forall r)}{(*)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(*)}{XY, \forall x(Y' \to \phi(x)) \vdash \phi(z), \forall x(Y \to \phi(x))} \quad \dfrac{\dfrac{}{\phi(z), XY, \forall x(Y' \to \phi(x)) \vdash \phi(z)}\,(b)}{\forall z(\phi(z)), XY, \forall x(Y' \to \phi(x)) \vdash \phi(z)}\,(\forall l)}{\forall x(Y \to \phi(x)) \to \forall z(\phi(z)), XY, \forall x(Y' \to \phi(x)) \vdash \phi(z)}\,(\to r)}{\forall x(Y \to \phi(x)) \to \forall z(\phi(z)), XY, \forall x(Y' \to \phi(x)) \vdash \forall z(\phi(z))}\,(\forall r)}{\forall x(Y \to \phi(x)) \to \forall z(\phi(z)), XY \vdash \forall x(Y' \to \phi(x)) \to \forall z(\phi(z))}\,(\to r)}{\forall x(Y \to \phi(x)) \to \forall z(\phi(z)) \wedge XY \vdash \forall x(Y' \to \phi(x)) \to \forall z(\phi(z))}\,(\wedge l)}{\vdash ((\forall x(Y \to \phi(x)) \to \forall z(\phi(z))) \wedge XY) \to (\forall x(Y' \to \phi(x)) \to \forall z(\phi(z)))}\,(\to r)$$

□

Exercise 6

(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 17

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 3 Functions

Exercise 19

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 22

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

$f(\texttt{null}, \vec{x}) = h(\vec{x})$;
$f(\alpha\texttt{::}l, \vec{x}) = g(\alpha, f(l, \vec{x}), l, \vec{x})$.

where the $\vec{x}$s can be anything, so certainly either αs or α-`lists`. More work to do here

Show how to define `tail` like predecessor.

$\texttt{tail}(\texttt{null}) = \texttt{null}$;
$\texttt{tail}(\alpha\texttt{::}l) = l$.

Comment on the rectype 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

$f(\emptyset, \vec{x}) = g(\vec{x})$;
$f(a \cup \{b\}, \vec{x}) = h(f(a), b, \vec{x})$

but we cannot be confident of a unique answer, since $a \cup \{b\}$ might be the same as $c \cup \{d\}$.

Exercise 24

Elementary, just a quick reality check. You do it by induction on 'i'.
$f_0(m, k) := m + k$;
$f_{n+1}(m, 0) := m$;[2]
$f_{n+1}(m + 1, k + 1) = f_n(m, f_{n+1}(m + 1, k))$.

[2]This 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.

$f_2(m+1,k+1) = f_1(m, f_2(m+1,k))$
$f_2(m+1,k+1) = m \cdot (f_2(m+1,k)+1)$.

Define an operation $\mathcal{DT} : (\mathbb{N}^2 \to \mathbb{N}) \to (\mathbb{N}^2 \to \mathbb{N})$ by

$(\mathcal{DT}f)(m,0) := m$;
$(\mathcal{DT}f)(0,m) := (\mathcal{DT}f)(m+1,0)$;
$(\mathcal{DT}f)(m+1,k+1) := f(m,(\mathcal{DT}f)(m+1,k))$.

Observe that if f is primitive recursive so is $\mathcal{DT}(f)$. That powers the induction.

Some tho'rts to sort out

What is the next operation (in the DT sequence) after exponentiation? People tend to think that it's "towers", so that $\alpha \uparrow\uparrow (\beta+1) = \alpha^{\alpha\uparrow\uparrow\beta}$ but it's easy to see that this cannot be right. For what is $\omega \uparrow\uparrow \omega$? Clearly it must be ϵ_0. But then $\omega \uparrow\uparrow (\omega+1)\omega^{\epsilon_0} = \epsilon_0$ and $\uparrow\uparrow$ is not strictly increasing. So we must have $\alpha \uparrow\uparrow (\beta+1) = (\alpha \uparrow\uparrow \beta)^\alpha$. So what, one might say? The point is that the tower function appears very early on in the Ackermann function—which behaves like a fixed point for DT. Surely there is something illuminating one can say about this..? see `minutes.tex`.

Observe that $\mathcal{DT} : (\mathbb{N}^2 \to \mathbb{N}) \to (\mathbb{N}^2 \to \mathbb{N})$ is monotone wrt dominance. [The proof will go as follows: suppose f is dominated by g; unpack $\mathcal{DT}(f)(x,y)$ to get some horrendous word $W(f,x,y)$ in 'f', 'x' and 'y'. Replace 'f' by 'g' in W to get an expression which, by assumption on f and g, must point to a number bigger than the number pointed to by $W(f,x,y)$. But this new expression is the result of evaluating $\mathcal{DT}(g)(x,y)$.]

Observe further that $\mathcal{DT}$ is just the Doner-Tarski operation minus the clause for limit ordinals. Isn't it...? Let's check...

The Doner-Tarski recursion for finite subscripts is:

$f_0(m,k) := m+k$;
$f_{n+1}(m,0) := m$;
$f_{n+1}(0,m) = f_n(m,1)$;
$f_{n+1}(m+1,k+1) = f_n(m, f_{n+1}(m+1,k))$.

If $f_{n+1} = \mathcal{DT}(f_n)$ then we must have

$f_{n+1}(m,0) := m$;
$f_{n+1}(0,m) := f_{n+1}(m+1,0)$;
$f_{n+1}(m+1,k+1) := f_n(m, f_{n+1}(m+1,k))$.

...and this last thing would look sort-of OK but for the fact that the first clause is missing and we have '$n+1$' instead of 'n' in the third. We saw this worry earlier.

Observe further that the Ackermann function is a fixed point for $\mathcal{DT}$, and is presumably the least fixed point above $\lambda(n,m).1$. I now find myself wondering if there might not be a simpler proof that Ackermann is not primitive recursive... one that procedes by showing that, for every primrec f, there is an n s.t. $\mathcal{DT}^n(\text{plus})$ dominates f.

How does it differ from the ωth Doner-Tarski function? Presumably the ωth Doner-Tarski function sends some pairs of naturals to infinite ordinals....

So I think we can characterise the Doner-Tarski hierarchy by saying: Do $\mathcal{DT}$ at successor stages and at limit stages take pointwise sups and then do $\mathcal{DT}$ again.

Exercise 25

It isn't what you think!

The next Doner-Tarski operation beyond exponentiation is declared by

$\beta \uparrow\uparrow 0 = \beta$;
$\beta \uparrow\uparrow (\alpha+1) = (\beta \uparrow\uparrow \alpha)^\beta$;
taking sups at limits.

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 $\beta = \omega$.

$\omega \uparrow\uparrow 0 = \omega$;
$\omega \uparrow\uparrow (\alpha+1) = (\omega \uparrow\uparrow \alpha)^\omega$;
taking sups at limits.

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 26

might need to check the indices

Show that, if f is primitive recursive, so are

1. the function $\sum_f(n) = \sum_{0\le x<n} f(x)$ 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 $\prod_f(n) = \prod_{0 \le x \le n} f(x)$ 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 27

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: $\alpha \to (\beta)^\gamma_\delta$)

We will start off by proving $\omega \to \omega_2^2$. (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.[3] This infinite branch either has infinitely many left turns in it, or infinitely many right turns.

Clearly the choice of a two-colouring was unneccessary: the same construction would have worked for any finite number of colours n. So we have proved $\omega \to (\omega)^2_n$.

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 $2^{2n} \to (n)^2_2$, or, if c is the number of colours, $c^{cn} \to (n)^2_c$. (As it happens this is not best possible: e.g., we know $6 \to (3)^2_2$.)

For higher exponents we reason by induction: assume $(\forall n)(\omega \to (\omega)^m_n)$ and try to prove $\omega \to (\omega)^{m+1}_n$. 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

[3] König'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 $f(x, y, z) \to (x)^y_z$.

I think it's probably best to write $f_y(x, z) \to (x)^y_z$ 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 $f, g : \mathbb{N} \to \mathbb{N}$, 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 $[\mathbb{N} \rightharpoonup \mathbb{N}]$ into acomplete metric space.

(ii) Show that the function $\Phi\ (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$ 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 $[A \rightharpoonup B]$. 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 $n \mathbin{\dot{-}} m$ represent $n < m$? Almost but not quite, because it isn't two-valued. We need an auxilliary function to zap all non-true (nonzero) truth values to 1.

$$\texttt{iszero}(0) := 0; \quad \texttt{iszero}(S(n)) := S(0).$$

`iszero` returns the truth-value of the assertion that its input is 0.

Then $\texttt{iszero}(n \dot{-} m)$ represents $m < n$ so of course $\texttt{iszero}(S(n) \dot{-} m)$ represents $m \leq n$ so of course $\texttt{iszero}(S(n) \dot{-} m) + \texttt{iszero}(n \dot{-} m)$ 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 $\texttt{pair}(n, m) = T(n + m) + m$ does the trick.

As for computing the unpairing functions ...

According to the definition, $\texttt{pair}(x, y)$ 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 $\leq \texttt{pair}(x, y)$. Thus `pair` is injective. To decode a number z as $\texttt{pair}(x, y)$, 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.

$$\texttt{fst}(x) \text{ is } \sum_{i<x}(\texttt{if } (\exists z < x)(\texttt{pair}(i, z) = x) \texttt{ then } i \texttt{ else } 0);$$
$$\texttt{snd}(x) \text{ is } \sum_{i<x}(\texttt{if } (\exists z < x)(\texttt{pair}(z, i) = x) \texttt{ then } i \texttt{ else } 0).$$

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

$$\sum_{m<n} \texttt{if } m \text{ and } n \text{ are coprime } \texttt{then } 1 \texttt{ else } 0.$$

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 $F(0) := \langle 1, 1\rangle$; $F(n+1) = \langle \mathtt{snd}(F(n)), \mathtt{fst}(F(n)) + \mathtt{snd}(F(n))\rangle$.
Then $\mathtt{Fib}(n) = \mathtt{fst}(F(n))$.

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 $\mathtt{Fib}(0)$ when computing $\mathtt{Fib}(n)$. If you program it in the primitive recursive way you end up with only one call to $\mathtt{Fib}(0)$.

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 $H(y) = \mathtt{fst}(F(y))$ and $y = \mathtt{snd}(F(y))$, so this is

$$\langle G(\mathtt{fst}(F(y)), \mathtt{snd}(F(y))),\ S(\mathtt{snd}(F(y)))\rangle,$$

and we notice that all occurrences of 'y' are wrapped up in F's, so this is

$$f(F(y)),$$

where f is

$$\lambda z.\langle G(\mathtt{fst}(z), \mathtt{snd}z), S(\mathtt{snd}(z))\rangle,$$

so $H(y) = \mathtt{fst}(Fy) = \mathtt{fst}(f^y(F0)) = \mathtt{fst}(f^y(b))$, where $b = \langle a, 0\rangle$.

Chapter 4: Machines

Exercise 38

If we have a mutually recursive definition of two functions f and g we can turn this into a single primitive recursive declaration of the function $\lambda n.\langle f(n), g(n)\rangle$, from which we can then recover f and g by composition with the unpairing functions.

Exercise 41

1. The clause that says $A(m+1,0) = A(m,1)$ takes care of the cases where $n = 0$. For the remaining cases we use transfinite induction on the lexicographic product. $A(m+1,n+1) = A(m, A(m+1,n))$. $\langle m, A(m+1,n)\rangle <_{lex} \langle m+1, n+1\rangle$ so $A(m, A(m+1,n)) > A(m+1,n)$ by induction hypothesis. Also $\langle m+1, n\rangle <_{lex} \langle m+1, n+1\rangle$ so $A(m+1,n) > n$ by induction hypothesis. This gives us $A(m+1,n+1) > A(m+1,n) > n$ so certainly $A(m+1,n+1) > n+1$, which was what we wanted.

2. We prove by induction on a that $(\forall n,m)(n < m \rightarrow .A(a,n) < A(a,m))$

   When $a = 0$, $A(0,n) := n+1$ whence $n < m \rightarrow A(0,n) < A(0,m)$.

   For $a > 0$ we reason as follows. Assume true for a, prove it for $a+1$. $A(a+1,n) := A(a, A(a,n-1))$ and $A(a+1,m) := A(a, A(a,m-1))$. If $n < m$ then the induction hypothesis tells us that $A(a,n-1) < A(a,m-1)$, and another application of the induction hypothesis tells us that $A(a, A(a,n-1)) < A(a, A(a,m-1))$—which is to say $A(a+1,n) < A(a+1,m)$ as desired.

3. We prove by induction on n that $(\forall m)(A(m+1,n) \geq A(m,n+1))$

   When $n = 0$, we have that $A(m+1,n) = A(m+1,0) = A(m,1)$.

   Now suppose the result holds for n and consider $n+1$. Let m be arbitrary. We want $A(m+1,n+1) \geq A(m,n+2)$

   The recursion tells us that $A(m+1,n+1) = A(m, A(m+1,n))$. The induction hypothesis gives us that $A(m+1,n) \geq A(m,n+1)$. whence $A(m, A(m+1,n)) \geq A(m, A(m,n+1))$ by clause (2).

   Clause 1 gives us that $A(m,n+1) > n+1$ and consequently $A(m,n+1) \geq n+2$. But A is monotone in its second argument by clause 2, so $A(m, A(m+1,n)) \geq A(m,n+2)$. Now $A(m, A(m+1,n)) = A(m+1,n+1)$, whence $A(m+1,n+1) \geq A(m,n+2)$ as desired.

4. $A(m+1,n) > A(m,n+1)$ by clause 3; $A(m,n+1) \geq A(m,n)$ by clause 2.

5. We will show that

$$A(0,n) = n+1$$
$$A(1,n) = n+2$$
$$A(2,n) = 2n+3 > 2n$$

The result for $m = 0$ follows by definition. For $m = 1$, we have $A(1,0) = A(0,1) = 0+2$ as required. By induction,

$$A(1,n+1) = A(0,A(1,n)) = A(0,n+2) = n+2+1 = (n+1)+2,$$

as required. Similarly,

$$A(2,n+1) = A(1,A(2,n)) = A(1,2n+3) = 2n+3+2 = 2(n+1)+3.$$

Using this result and those previously obtained we have

$$A(m+2,S(n)) = A(m+1,\underbrace{A(m+2,n)}_{\geqslant A(m,n)\ (4)}) \overset{(2)}{\geqslant} A(m+1,\underbrace{A(2,n)}_{>2n}) \overset{(2)^*}{>} A(m+1,2n) \overset{(4)}{\geqslant} A(m,2n)$$

as required, provided we prove $(2)^*$: strictly monotone increasing. True for $m = 0$. By (1)

$$A(m+1,n+1) = A(m,A(m+1,n)) > A(m+1,n)$$

as required, and $n = 0$ is trivial.

■

Exercise 43

Define the terms *primitive recursive function; partial recursive function; total computable function.* Ackermann's function is defined as follows:

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

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?

You have to prove by induction on 'y' that this holds for all n.

Base case: $y = 0$. We want $(\forall n)(f_{n+1}(0) = f_n^1(1))$. Let n be arbitrary.
Want $f_{n+1}(0) = f_n^1(1)$. Expand using definition of f:
LHS: $f_{n+1}(0) = A(n+1,0) = A(n,1)$. RHS: $f_n^1(1) = A(n,1)$, as desired.

Now for the induction step.

Assume $(\forall n)(f_{n+1}(y) = f_n^{y+1}(1))$. We want $(\forall n)(f_{n+1}(y+1) = f_n^{y+2}(1))$. Let n be arbitrary as before and expand $f_{n+1}(y+1)$ as before to get $A(n+1, y+1)$, which is $A(n, A(n+1, y))$, which is $f_n(f_{n+1}(y))$. But by induction hypothesis on 'y', $f_{n+1}(y) = f_n^{y+1}(1)$, so $f_n(f_{n+1}(y)) = f_n(f_n^{y+1}(1))$, which of course is $f_n^{y+2}(1)$.

(This is a useful example of a general problem. There are two universally quantified variables in '$(\forall n)(\forall y)(n \geq 0 \rightarrow f_{n+1}(y) = f_n^{y+1}(1))$, and both of them range over a rectype. On the face of it, each quantifier can be dealt with by either a UG or an induction. In many cases, such as the one in hand, there is only one strategy that will work. You *have* to treat 'n' by UG and 'y' by induction.)

Finally, we show that f_n is primitive recursive for each n. We will use the fact we have proved, namely, that $f_{n+1}(y) = f_n^{y+1}(1)$. Consider the declaration:

$$g(0) := h(1); \quad g(n+1) := h(g(n)).$$

This is clearly primitive recursive: g will be primitive recursive if h is. But f_{n+1} is obtained by primitive recursion over f_n in precisely the way g is declared over h, so, as long as f_0 is primitive recursive, we can prove all the f_n to be primitive recursive by induction on n.

Exercise 44

1. Stan Wainer writes:

   Nested n-recursion is any definition $f(x_1, ...x_n, a) = T(f, x_1, ...x_n, a)$ where T is any term built up from given functions and applications of $f(t_1, ...t_n, a')$ where the vector $t_1, ...t_n$ is always lexicographically less than $x_1, ...x_n$. See [50].

2. The appropriate generalisations of the Ackermann function can be found in (e.g.) [53] p 28 exercise 19. (He calls them *Péter functions*.) When $n > 1$

$$\phi_n(0, y_1, \cdots y_n) = \phi_{n-1}(y_1, \cdots y_n)$$

$$\phi_n(y_0 + 1, 0, y_2 \cdots y_n) = \phi_n(y_0, 1, \cdots y_n)$$

$$\vdots$$

$$\phi_n(y_0 + 1, \cdots y_{n-1} + 1, 0) = \phi_n(y_0 + 1, 1, \cdots y_{n-2} + 1, y_{n-1}, 1)$$

$$\phi_n(y_0{+}1, \cdots y_n{+}1) = \phi_n(y_0{+}1, \cdots y_{n-2}{+}1, y_{n-1}, \phi_n(y_0{+}1, \cdots y_{n-2}{+}1, y_{n-1}, \cdots \phi_n(y_0{+}1, \cdots y_{n-1}{+}$$

I can't parse this last one. I think one of those '$y_0 + 1$' ought to be a mere y_0. My guess is that when $n = 2$ the last clause should be:

$$\phi_2(y_0 + 1, y_1 + 1, y_2 + 1) = \phi_2(y_0, y_1 + 1, \phi_2(y_0, y_1, \phi_2(y_0, y_1, y_2)))$$

If it were ϕ_3 in play we would have

$$\begin{aligned}&\phi_3(y_0 + 1, y_1 + 1, y_2 + 1, y_3 + 1)\\ &\quad = \phi_3(y_0, y_1{+}1, y_2{+}1, \phi_3(y_0, y_1, y_2{+}1, \phi_3(y_0, y_1, y_2, \phi_3(y_0, y_1, y_2, y_3))))\end{aligned}$$

Observe that we can prove the totality of ϕ_n by wellfounded induction on the lexicographic product ordering on $\mathbb{N}^n$. Or, in PA, by mathematical induction using n nested inductions.

Exercise46

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.

(i) If L_1 and L_2 are both regular must $L_1 \oplus L_2$ be regular?

(ii)

Suppose Σ_1 and Σ_2 are two alphabets with $|\Sigma_1|$ and $|\Sigma_2|$ both even natural numbers (so that both alphabets can be tho'rt of as a set of generators with their inverses). Let L_1 and L_2 be regular languages over Σ_1 and Σ_2 respectively and let G_1 be the group consisting of elements pointed to by words in L_1 and let G_2 be the group analogously indicated by words in L_2. (We say that the machines corresponding to L_1 and L_2 are **word acceptors** for G_1 and G_2.) What group corresponds to the interleaving of $L_1 \oplus L_2$?

Answer: (i) Yes. Suppose L_1 and L_2 are recognised by deterministic finite state machines M_1 and M_2. We will describe a nondeterministic machine that recognises $L_1 \oplus L_2$.

The new nondeterministic machine $M_1 \oplus M_2$ will have $|M_1| \cdot |M_2|$ states. Each state of $M_1 \oplus M_2$ represents a guess about how the string-seen-so-far is to be represented as an interleaving of a string from Σ_1^* and a string from Σ_2^*. Thus a state of $M_1 \oplus M_2$ will be an ordered pair $\langle m_1, m_2 \rangle$ of states of M_1 and M_2. When it receives a character a from $\Sigma_1 \setminus \Sigma_2$ it goes to the state $\langle m_1', m_2 \rangle$ where m_1' is the state whither M_1 would go were it to receive a when in state m_1. *Mutatis mutandis* when it receives a character b from $\Sigma_2 \setminus \Sigma_1$ it goes to the

state $\langle m_1, m_2' \rangle$ where m_2' is the state whither M_2 would go were it to receive b when in state m_2. When it receives a character a from $\Sigma_1 \cap \Sigma_2$ it goes to one of the two states $\langle m_1', m_2 \rangle$ and $\langle m_1, m_2' \rangle$.

The start state of $M_1 \oplus M_2$ is the ordered pair of the two start states, and the accepting states are ordered pairs of accepting states of M_1 and M_2.

Answer (ii)

Naturally one's first thought is the free product $G_1 * G_2$, but of course what one actually obtains is a quotient. Say v and u in $(\Sigma_1 \cup \Sigma_2)^*$ are *equivalent* if there are $w_1 \in L_1$ and $w_2 \in L_2$ such that both u and v are interleavings of w_1 with w_2. Then uv^{-1} belongs to $L_1 \oplus L_2$, and thus the homomorphism onto the quotient identifies any two equivalent words. But two words are equivalent precisely if they arise as interleavings from the same pair of elements of G_1 and G_2. So the quotient is precisely the ordinary (direct, cartesian) product $G_1 \times G_2$.

Exercise 51

(1) Let A be semidecidable (and nonempty), so that it is g"$\mathbb{N}$ for some computable g, and let f be the (gnumber of the) μ-recursive function that sends n to the nth output of g's volcano. Pick some arbitrary $a \in A$. Then define h by:

$h(n,k) =$ `if` $T(f,n,k)$ has final state `HALT`
`then` final register-contents of $T(f,n,k)$
`else` a.

[My *Doktorvater* Adrian Mathias calls this an "impatient" function: if it doesn't immediately get what it wants then it emits a default value.]

h is seen by inspection to be primitive recursive.

If $f(n) = y$, say, then $f(n)$ halts, so $h(n,k) = y$ for some k and $y \in h$"$(\mathbb{N}^2)$. Hence $A \subseteq h$"$(\mathbb{N}^2)$. Conversely, we always have $h(n,k) \in A$. So h"$(\mathbb{N}^2) = A$, with h primitive recursive.

(2) It's immediate that if X is the range of a μ-recursive function then $X = f$"Y for some computable f and semidecidable $Y \subseteq \mathbb{N}$ (on taking $Y = \mathbb{N}$). Suppose $X = f$"Y for some semidecidable $Y \subseteq \mathbb{N}$. If X is empty then we're done. Otherwise, let g be the primitive recursive function defined above with range Y, and set:

$$h(n,k) = f(g(n,k))$$

Then h is μ-recursive and has range f"Y.

Exercise 53

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.

(Jane Aston's answer)

Suppose A and B are semidecidable, with $A = \{n : f(n)\downarrow\}$ and $B = \{n : g(n)\downarrow\}$.

Define $h(n)$ as:

`if` (n is even) `then` $f((\mu k \leq n)(n \dot{-} 2k = 0))$ `else` $g((\mu k \leq n)(n \dot{-} (2k+1) = 0))$.

This h is certainly μ-recursive, and $h(2n) = f(n)$ and $h(2n+1) = g(n)$. So $\{n : h(n)\downarrow\} = \{n : n \text{ even and } f(n/2)\downarrow\} \cup \{n : n \text{ odd and } g((n-1)/2)\downarrow\}$, and this is the set we wanted.

For the other direction suppose the set $\{2n : n \in A\} \cup \{2n+1 : n \in B\}$ is semidecidable. So it is $\{n : h(n)\downarrow\}$ for some μ-recursive h. Define $h_1(n) =: h(2n)$ and $h_2(n) =: h(2n+1)$. (This is OK by composition). Then

$$\{2n : n \in A\} \cup \{2n+1 : n \in B\}$$

is

$$\underbrace{\{2n : h_1(2n)\downarrow\}}_{A} \cup \underbrace{\{2n+1 : h_2(2n+1)\downarrow\}}_{B}.$$

Thus $A = \{n : h_1(n)\downarrow\}$ and $B = \{n : h_2(n)\downarrow\}$, so A and B are both semidecidable.

Exercise 55

> *"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?"*

This is much easier than I thought when I set it!

For each i run the volcano for $\{i\}$ until it produces a number $> 2i$, then put the result into X. This construction ensures that, for every $n \in \mathbb{N}$, $|X \cap [0, 2n]| \leq n$. By spicing up the construction we can make X as thin as we like.

Another thing you can do is observe that $\mathcal{D} = \{i \in \mathbb{N} : \{i\}(i)\downarrow\}$ is semidecidable. Might it be the set we want ...? Let A be any infinite semidecidable set. Then $A = \{i\}$"$\mathbb{N}$ for some $\{i\}$. Then $\{i\}(i)$ is defined and is in both A and $\mathcal{D}$. But how then do we know that $|\mathbb{N} \setminus \mathcal{D}| = \aleph_0$? But this is easy: we know that there are infinitely many i such that $\{i\}$ is everywhere undefined—we can exhibit them by hand whatever the gnumbering system is.

Exercise 56

Suppose the graph of f is a $\exists\forall$ set. That is to say, there is a decidable set $A \subseteq \mathbb{N}^4$ s.t. $(\forall uv)(\langle u, v\rangle \in f \longleftrightarrow (\exists x)(\forall y)(\langle u, v, x, y\rangle \in A))$. If there is to be a g of the kind desired we must have $(\forall uv)(\langle u, v\rangle \in f \longleftrightarrow (\exists x)(\forall y > x)(g(u, y) = v)))$ whence $(\forall uv)[(\exists x)(\forall y)(\langle u, v, x, y\rangle \in A) \longleftrightarrow (\exists x)(\forall y > x)(g(u, y) = v)]$.

What is $g(u, w)$? We seek x and v such that $(\forall y)(\langle u, v, x, y\rangle \in A)$. The idea is that when we find them, v will be the value of g. Of course we can't reliably identify such x and v in finite time, but we have some wiggle room because finitely many ws don't matter. Fix u; we enumerate the pairs $\langle x, v\rangle$ and examine them one-by-one until we find x and v s.t. $\langle u, v, x, 0\rangle \in A$. Then we return v as the value of $g(u, 0)$. What is $g(u, 1)$? We ask whether or not $\langle u, v, x, 1\rangle \in A$. If it is, we return v as the value of $g(u, 1)$ and we ask whether or not $\langle u, v, x, 2\rangle \in A$. If it is, we return v as the value of $g(u, 2)$, and so on. If $\langle u, v, x, 1\rangle \notin A$ we look for the next pair x' and v' s.t. $\langle u, v', x', 0\rangle \in A$ and we return v' as the value of $g(u, 1)$. At some point in the enumeration of the pairs (the nth, say) we will encounter x' and v' such that $(\forall y)(\langle u, v', x', y\rangle \in A)$. This encounter will happen because f is total, and the pair x', v' is unique because f is a function. Then $g(u, w) = v'$ for all $w \geq n$. It is true, of course, that when (at stage n) we encounter such a pair x', v', we have no way of telling that it is the last pair we will ever examine (n cannot be computed from u)—but that doesn't matter. Had we been able to compute n from u then f would have been computable.

Exercise 57

(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 prove or provide a counterexample.

For (i) We have a volcano that emits gnumbers of volcanoes. Every time it emits such a number we power up the corresponding volcano, so that at each finite stage we have finitely many volcanoes on the go. By the end of time the chorus of volcanoes has emitted every number in the union.

For (ii), let A be a semidecidable set that is not decidable. Then $\{A\}$ is decidable, but its sumset is not.

For (iii), let A be a semidecidable set that is not decidable. Then all the singletons $\{1\}$, $\{2\}$, ... of members of A are decidable, but $\bigcup_{i\in A}\{i\} = A$ is not.

This might remind you of Conway on Countable choice ("A counted union of counted sets is counted; a countable union of counted sets is countable ..." but a countable union of pairs can be uncountable.) but there are extra subtleties in this that are worth spelling out. By thinking of a countable family $\mathcal{F}$ of semidecidable sets as *itself* a semidecidable set we are perforce thinking of $\mathcal{F}$ as a set of indices of functions.

That is to say, we have—by equipping each set in $\mathcal{F}$ with a function (volcano) that emits it—done all that we would have wanted AC (the choice of a counting) to do. The countable/counted contrast is not at all like the decidable/semidecidable contrast.

For (iv), consider (thanks to Shoham Letzter[4]) the set A:

$$A = \{\{2^t, 3^n\} : \{t\}_n(t)\!\downarrow\}$$

(Beware annoying double use of '{}' notation for both functions-in-intension and sets!)

A is a decidable set of decidable sets. But $\bigcup A$ is

$$A = \{3^n : n \in \mathbb{N}\} \cup \{2^t : \{t\}(t)\!\downarrow\}$$

which is clearly not decidable.

Exercise 58

The first step towards discovering one is to observe that we can obtain the effect of the last paragraph by adopting a rule of inference

$$\frac{(\forall x)(\theta(x) \longleftrightarrow (\exists y)(\phi(x,y))), \quad (\forall x)(\theta(x) \longleftrightarrow (\forall y)(\psi(x,y)))}{(\exists y)(\forall x)(x \in y \longleftrightarrow \theta(x))}$$

where ϕ and ψ are Δ_1^0.

So we now have a decidable set of axioms, but the price we have paid for it is to have a new—awkward—rule of inference. Fortunately we can "internalise" this rule of inference by adopting, for each θ, a scheme

$$(\forall x)(\theta(x) \longleftrightarrow (\exists y)(\phi(x,y))) \wedge (\forall x)(\theta(x) \longleftrightarrow (\forall y)(\psi(x,y)))) \rightarrow (\exists y)(\forall x)(x \in y \longleftrightarrow \theta(x))$$

where ϕ and ψ are Δ_1^0 as before.

or

$$(\forall x) \bigwedge \left(\begin{array}{l} \theta(x) \longleftrightarrow (\exists y)(\phi(x,y)) \\ \theta(x) \longleftrightarrow (\forall y)(\psi(x,y)) \end{array} \right) \rightarrow (\exists y)(\forall x)(x \in y \longleftrightarrow \theta(x)) \qquad (9.1)$$

Exercise 61

Given a machine $\mathfrak{M}$ and an input k, form a machine $\mathfrak{M}^k$ so that:

$\mathfrak{M}^k(k) = \mathfrak{M}(k)$;
$\mathfrak{M}^k(n) = 0$ for all $n > k$;
$\mathfrak{M}^k(n)\!\uparrow$ for all $n < k$

Let $f(k,n)$ be the μ-recursive function which is zero if $\mathfrak{M}^k(n)$ halts and undefined otherwise. Then the least n such that $f(k,n)$ halts is k if $\mathfrak{M}(k)$ halts and $k+1$ otherwise, so the function g defined in the question is total and solves the halting problem.

[4]This proves—in case you ever doubted it—that a vegan diet is good for your brain!

Exercise 64

The relational product of two primitive recursive relations might not be a primitive recursive relation.

Try $R(x,y)$ iff $x = \langle m, i\rangle$ and $y = \langle t, o\rangle$ and $(\exists t' < t)(\{m\}_{t'}(i)\downarrow = o)$. Then consider the composite $(\exists z)(R(x,z) \wedge R(y,z))$. This is surely not a primitive recursive relation. It's not even decidable, by Rice's theorem.

There's probably a simpler demonstration.

Exercise 65

When setting the question I envisaged the following answer:

We build a transversal T in stages, T_n.

Put 0 into T_0. Compute $[[n \sim 0]]$ (the truth-value of $n \sim 0$) for all n in parallel. As soon as we discover a k such that $\neg(k \sim 0)$ we put k into T_1.

Subsequently at the nth stage we compute $[[k \sim m]]$ for all $m \in T_n$ and all $k \notin T_n$. As soon as this process reveals a k such that $(\forall m \in T_n)(\neg(k \sim m))$ we set $T_{n+1} := T_n \cup \{k\}$. Since $\sim$ is of infinite index, there is such an m and—since the graph of $\sim$ is the complement of a semidecidable set—we will find it. The idea is that $T_\omega = \bigcup_{i \in \mathbb{N}} T_i$ is a transversal.

However, I'm no longer happy with this approach. How can we be sure that T_ω meets every equivalence class? It *might* be a useful exercise to think about how one might modify the construction to get it to work but i'm not staking my life on it ... and Henk-Jaap Wagenaar tells me it can't be repaired. Anyway, a solution is easily found by observing that $\{n : (\forall m < n)(\neg(m \sim n))\}$ is semidecidable. It's also evidently a transversal.

Too easy for a tripos question, really.

Exercise 66

Suppose $f : \mathbb{N}^k \to \mathbb{N}$ is total computable and increasing: $f(\vec{x}) > \max(\vec{x})$. Show that there is a decidable $A \subseteq \mathbb{N}$ satisfying $f``A^k = \mathbb{N} \setminus A$.

(For the moment i'll just prove the special case where $k = 1$.)

Put 0 into A. Put $f(0)$ into $\mathbb{N} \setminus A$. Put into A any n s.t. $f(n) = f(0)$. So far so good. Now look at the least k that we haven't considered. It's not f of anything known already to be in A so we can safely put it into A. Put $f(k)$ into $\mathbb{N} \setminus A$ and of course also put into $\mathbb{N} \setminus A$ any k' s.t. $f(k') = f(k)$. Keep chugging on.

Exercise 67

see `http://vxheavens.com/lib/awd00.html`

Exercise 68

The answers are 'yes', 'yes' (i.e., it changes).

Exercise 70

The second function is a step function and is therefore computable: all step functions are, being finite objects. The fact that we do not know *which* step function it is merely means that we do not know *how* to compute it.

The first function might be computable. Nobody has a clue. My guess is that it is the computable function $\lambda n.\texttt{true}$.

The third function is in fact computable, in the sense that there is a computable function with the same graph, but you would not guess it from the declaration. Remember that computability is in the first instance a property of function declarations (functions-in-intension), not of functions-in-extension.

This is another opportunity to wheel out the expression 'self-validating' sometimes used by CompScis to describe functions that tell you what they are doing. If the function $\lambda n.\texttt{true}$ computes the second function in the exercise then we can't tell that just by looking at the code.

Exercise 73

If the set of gnumbers of boxes that cannot tile the plane is to be semidecidable then, whenever a box-of-tiles does not tile the plane then we will have to be able to detect this fact in finitely many steps. We rummage around in the box b and add tiles, one at a time, to a growing finite assembly of tiles on the plane. Every now and then we get stuck, so we backtrack and try something different. How do we know that we can't go on making ever bigger and bigger assemblies, but always getting stuck and having to backtrack?

Fix a box b of tiles and consider the set of finite (legal) assemblies of tiles: they form an obvious partial order. However what we want is a *tree*, so we consider instead the set of finite sequences of applications of tiles—the partial order is an obvious quotient of this set. There are only finitely many flavours of tiles in the box b, and only finitely many buds in any assembly where we can put a new tile, so this tree is finitely branching. If b cannot be used to tile the plane then this tree has no infinite paths. Now, by König's lemma, the tree must be actually finite, and that means that if we try adding new tiles in a particular order we will know when we have run out of possibilities.

The point is this: König's lemma tells us that if b will not tile the plane there will be what one might call a *cut*: a finite set of legal assemblies none of which can be legally enlarged, and such that every legal assembly is a subset of one of them. As long as we search systematically then if there is such a cut we will find it.

However (thank you Zhen Low and Lovkush Agarwal!) one has to be careful how one states this. You could consume an infinite amount of time putting tiles in a $\mathbb{Z}$-*line* without attempting to fill the space and thus fail to discover in finite time that your enterprise was doomed. (One thinks of space-filling curves in this connection) What you have to do is *spiral out from the origin*.

Mind you, this probably needs more discussion still. Spiral out from the origin so that the $(x^2 + y^2)$th day (or thereabouts) finds you attempting to put a tile on (x, y). It's a *weee* bit more complicated than that, because remember there is backtracking. So one doesn't count for this purpose time spent in blind alleys.

Exercise 79

Part (1) *Show that the range of an increasing total function* $f : \mathbb{N} \to \mathbb{N}$ *is a decidable set.*

Either the function f is eventually constant, in which case its range is finite and is therefore decidable, or it is unbounded. If it is unbounded, the way to test whether or not the candidate number is a value of f is to compute f of 0, 1, 2, 3 ... until the candidate number is either hit or overtaken.

Of course, if you do not know which of these two situations is the one you are in, you have no way of discovering the decision method in virtue of which this set is decidable, but that is your problem, not God's. This rams home the point that, for a problem to be solvable, what is necessary is that there should be a decision method for it–not for there to be a decision method for it *known to us*.

You might think this is an elementary point—and it is—but it is one that can be easily overlooked.

Another point worth taking away from this is that we have here a nonconstructive proof that something is recursive. Ironical, what?!

Worth emphasising to beginners that the way in which we (or at least many of us) think of the natural numbers, as a snake wandering through space ... has the potential to seriously mislead. The temptation is to think of a subset $X \subseteq \mathbb{N}$ as the snake with some of its nodes lit up. This is OK if X is decidable but not otherwise. The snake makes you think that $\mathbb{N}$ and all its subsets are random access devices, or at least (if you don't like that—and you mightn't) that it's a *sequential* access device. You can access members of a decidable subset $X \subseteq \mathbb{N}$ by using the enumeration in increasing order. However, if X is merely *semidecidable* then it's still a sequential access device all right, but the order in which you get access to the elements is not in order of *magnitude* but the order in which the volcano emits them. You must not attempt to visualise X in the way you visualise $\mathbb{N}$!

Part (2) *Show that every decidable subset of* $\mathbb{N}$ *is the range of an increasing total computable function* $\mathbb{N} \to \mathbb{N}$.

If A is decidable then $A = f$"$\mathbb{N}$ and $\mathbb{N} \setminus A = g$"$\mathbb{N}$ for two total computable functions f and g. Run the volcanos for f and for g (and we use the non-repeating style of volcano) until one of them emits 0 (they may emit other things, do not record them); *then restart them* and run them until one of them emits 1; and so on. We define the strictly increasing total computable function a by $a(n)$ is the nth member of A recorded as being emitted by this duet of the volcanos.

Part (3) *What if f is merely nondecreasing (but still total)?*

The technique of Part (2) works here too.

Part (4) *What if f is increasing but perhaps not everywhere defined? (i.e., $(\forall n)(\forall m)(((n < m) \wedge f(n)\downarrow \wedge f(m)\downarrow) \rightarrow f(n) < f(m))$?)*

Strictly-increasing and nondecreasing are equally good here, but if f is not total we are stymied. If f is undefined at only finitely many inputs then we can tweak it into a total computable function function with the same range. ("Hard-code" the missing bits).

Part (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 $\mathbb{N} \rightarrow \mathbb{N}^n$?*

You have to order $\mathbb{N}^k$ in order-type ω.

Exercise 80

Suppose X is a semidecidable set. Then there is a volcano that emits members of X, possibly with repetitions and not in increasing order. We define an increasing function $s : \mathbb{N} \rightarrow \mathbb{N}$ by $s(0) :=$ the first number emitted by the volcano, and thereafter $s(n+1)$ is the first number $> s(n)$ that the volcano emits. Evidently s"$\mathbb{N} \subseteq X$, s is an increasing computable function and—because X is infinite—s is *total*, so, by exercise 79, its range is a decidable subset of $\mathbb{N}$.

Exercise 81

The best way to answer this question is to draw lots of pictures.

Clearly we are going to have to execute a back-and-forth construction.

Think of the naturals in $\langle \mathbb{N}, <_A \rangle$ as $0_A, 1_A, 2_A \ldots$ and think of the naturals in $\langle \mathbb{N}, <_B \rangle$ similarly as $0_B, 1_B, 2_B \ldots$.

Clearly we wish to pair 0_A with 0_B. What do we do thereafter? In the routine back-and-forth construction we seek, at stage n, a mate in $\langle \mathbb{N}, <_B \rangle$ for the first n_A we have not already found a mate for. We examine $0_B, 1_B \ldots$ and so on until we find one that lives in the open interval that qualifies it to be a mate for n_A. This process of checking involves asking questions like "$x <_B y$?" all of which are *ex hypothesi* answerable, since the graphs of $<_A$ and $<_B$ are decidable sets of ordered pairs. Then we come back the other way. At the end of time we have a bijection as usual.

Duplication with p 222

Exercise 82

I can't do the first two parts!

By considering enumerations of the partial computable functions, find a computable partial function that cannot be extended to a computable total function.

Let $\{\{n\} : n \in \mathbb{N}\}$ be an enumeration of the partial computable functions of arity 1, that is, $\{n\} : \mathbb{N} \to \mathbb{N}$.

Now take $f : \mathbb{N} \to \mathbb{N}$ to be $f(m) = \{m\}(m) + 1$. Note that f is certainly not total, and it certainly is recursive. Suppose now that h is total computable and extends f to all of $\mathbb{N}$. Then we must have $h = \{n_0\}$ for some n_0, because h is total computable.

Then $h(n_0) = \{n_0\}(n_0)$, and as h total, the latter is defined. Therefore $f(n_0)$ is defined, and

$$\{n_0\}(n_0) = h(n_0) = f(n_0) = \{n_0\}(n_0) + 1.$$

Exercise 83

Given f as in (i), define g by picking $a_i \in A_i$ for each i (such that A_i is nonempty!) and then computing $g(x) = \sum_{i=0}^{n} i(1 - f(a_i, x))$. Incidentally, it is (as far as I know [says PTJ]) an open problem whether the equivalence of (i) and (ii) still holds for (countably) infinite families of sets—if anyone has any ideas about this, let us know! For the last part, take

$$A_0 = \{n : f_n(n) \textit{ is (defined and) odd}\}$$

and

$$A_1 = \{n : f_n(n) \textit{ is (defined and) even}\}$$

If we had a total function taking the value 0 on A_0 and 1 on A_1, it couldn't equal f_n for any n.

Exercise 84

[marked by PTJ as HARD]

A set $A \subseteq \mathbb{N}$ is called *Diophantine* if there exists a polynomial $p(x, y_1, \ldots, y_n)$ with integer coefficients such that $x \in A$ if and only if there exist $y_1, \ldots, y_n$ such that $p(x, y_1, \ldots, y_n) = 0$. 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.

[*The first part follows easily from question ??, and the (obvious) fact that the set of zeros of a polynomial is decidable. For the last part, given a polynomial $p(x, \vec{y})$ witnessing the fact that A is Diophantine, consider the polynomial*

$q(x, \vec{y}) = x - (x+1)((x, \vec{y}))^2$. *Note that, given Matiyasevich's result, this implies that there is a polynomial* $p(x_1, \ldots, x_n)$ *such that the non-negative values taken by* p *are exactly the primes. Incidentally, if students worry about whether Diophantine equations should be solved in integers or in natural numbers, point out to them that* $p(x) = 0$ *has a solution in* $\mathbb{N}$ *iff* $p(s^2 + t^2 + u^2 + v^2) = 0$ *has a solution in* $\mathbb{Z}^4$ *(since every natural number is a sum of four squares), and that* $p(y) = 0$ *has a solution in* $\mathbb{Z}$ *iff* $p(y).p(-y) = 0$ *has a solution in* $\mathbb{N}$.]

Exercise 85

> "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."

It's coming back to me, slowly. I think the answer must be that any spectrum (of a single sentence that is, not a theory) must be decidable. After all, for any formula ϕ and any n there are only finitely many structures of size n that are suitable for $\mathcal{L}(\phi)$ and it suffices to examine them exhaustively.

Another fact that swims into my mind as being vaguely relevant is that one can write down an expression in predicate calculus that says that there are precisely n things in the universe. This is useful if one is thinking about spectra of *theories* ... every subset of $\mathbb{N}$ is a spectrum of a theory, even if not a recursively axiomatisable one. It does at least show that every decidable subset of $\mathbb{N}$ is the spectrum of a recursively axiomatisable theory.

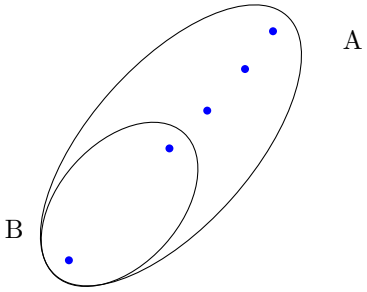
I have no idea about the last part. Most vexing.

Chapter 5: Lambda Calculus

Exercise 86

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: $((A \to B) \to A) \to A$ is not a constructive thesis.

Suppose *per impossibile* that there were a uniformly definable (and, accordingly, invariant) function P for Peirce's law. Let B be a two-membered set, and let A be obtained from B by adding three new elements.



A has five members and B has two, so any function $A \to B$ identifies a distinguished member of B, namely the one with larger preimage. This defines a function from $A \to B$ to B, which is to say (since $B \subseteq A$) a function from $A \to B$ to A. So what we have, in this rather special case, is a distinguished function $(A \to B) \to A$. Let us call this function F. F exists only because of the special circumstances we have here contrived, and it's not the sort of thing that P would normally expect to have to deal with, so we should expect P to experience difficulty with it ... which of course is what we want! But, if we have a term P, we can apply it to F to obtain a distinguished member of A. But clearly there is no way of picking an A in this way. The alleged existence of a uniformly definable P is trying to tell us that whenever we have a set of five things divided into two parts, one with two things in it and the other with three, then one of the five things is distinguished. And that's clearly not true.

On what features of A and B does this counterexample rely? A function $A \to B$ has to give us (*via* the pigeonhole principle) a distinguished element of B, so we need B to have two elements, and A (and therefore $A \setminus B$) to have an odd number. $|A \setminus B| = 1$ is no good, beco's then A has a distinguished element, which we don't want. $|A \setminus B| = 3$ is the smallest number that will do, and that is what Dana Scott gives us.

Exercise 87

One starts by noticing that $A \to (A \to A)$ and $(A \times A) \to A$ are naturally isomorphic, by `curry`ing and `un-curry`ing. So let's show that $(A \times A) \to A$ has only two definable inhabitants: left and right projection.

Any definable inhabitant of that type must commute with any permutation of A ... but what exactly do we mean by this? We need to get straight what it is in general for a permutation of A to act on some complex construct involving A and other things. We do this by recursion on the structure of the complex construct. For $\pi \in \mathrm{Symm}(A)$, π acts on A as itself, and on any other atom as the identity. How does π act on $X \to Y$? Clearly it must send $f \in X \to Y$ to f^π, by which we mean $\{\langle \pi(x), \pi(y)\rangle : \langle x, y\rangle \in f\}$ where $\pi(x)$ is what the induced action of π does to x, told us by the recursion.

We want to show that if $f : (A \times A) \to A$ commutes with all permutations of A then it is one of the two projections.

We start by showing that $f(\langle a, a'\rangle)$ must be a or a'. Suppose it weren't, and that a_1 and a_2 gave us a counterexample. Let π be a permutation that fixes a_1 and a_2 and moves $f(a_1, a_2)$. Then f doesn't commute with π. (We have assumed that A has enuff inhabitants but I think that's allowed ...?)

Now we have to show that a definable f must always "jump the same way". With a view to obtaining a contradiction let us suppose that there are a_1, b_1, a_2, b_2 all distinct s.t $f(a_1, a_2) = a_1$ but $f(b_1, b_2) = b_2$. Let π be the double transposition $(a_1, b_1)(a_2, b_2)$. We have $f(a_1, a_2) = a_1$ so we must have $f(\pi(a_1), \pi(a_2)) = \pi(a_1)$. But this is $f(b_1, b_2) = b_1$, contradicting $f(b_1, b_2) = b_2$.

Further thoughts to be worked up... show by the same means that the only definable inhabitants of $(A \to A) \to (A \to A)$ are I, KI and the Church numerals.

Fixed under all permutations of A. Permutations of A act on A by moving members of A around? How do they act on things obtained from A? If π is a permutation of A, it will, when acting on $A \times A$, send $\langle a_1, a_2\rangle$ to $\langle \pi(a_1), \pi(a_2)\rangle$. It acts on subsets of A by translation: π sends $X \subseteq A$ to $\{\pi(a) : a \in A\}$ also written π"A. What does π do to a function $f : A \to A$? f is a set of ordered pairs of members of A, so we send it to $\{\pi(\langle a_1, a_2\rangle) : \langle a_1, a_2\rangle \in f\}$ which is $\{\langle \pi(a_1), \pi(a_2)\rangle) : \langle a_1, a_2\rangle \in f\}$ which is $\pi \cdot f \cdot \pi^{-1}$, which we can write f^π. What is $\pi(g)$ when $g : A \to (A \to A)$? Well, g is a set of ordered pairs whose first components are members of a and whose second components are functions $A \to A$. So it's $\lambda a.(\pi^{-1}a)^\pi$.

Exercise 88

$\texttt{pair}{:=}\ \lambda xyf.fxy$
$\texttt{fst}{:=}\ \lambda p.p\ \texttt{true}$
$\texttt{snd}{:=}\ \lambda p.p\ \texttt{false}$
$\texttt{nil}{:=}\ \lambda x.\texttt{true}$

What are the types of these expressions?

`pair` is clearly of type $A \to (B \to (A \to (B \to C)) \to C)$ so (assuming naturally that a is of type A and b is of type B), `pair` a b is of type $(A \to (B \to C)) \to C$.

Now we compute some unpairing:

Consider what happens if one applies an object of type $(A \to (B \to C)) \to C$ to an object of type $X \to$ (not sure what i meant here)

Gödel's β-function trick: capturing recursion in the ring language (without exponentiation)

The ring language is not quite the same as the language for primitive recursive arithmetic, in that it has not only '0' and '=', but also '1', and the two operations '$+$' and '$\times$' (or $\cdot$). It does not have '$\leq$' or '$<$' but they can be defined. (I think the official answer is $x \leq z$ if $(\exists y)((x + y \cdot y) = z)$.) It doesn't have '$S$' either but it is clear that that, too, can be defined.

To capture recursion you need somehow to capture *certificates*, and this—on the face of it at least—means capturing *sequences*, since when θ is defined by recursion a certificate that $y = \theta(x)$ is *prima facie* a sequence of ordered pairs from an initial segment of the graph of θ. We can of course use the prime powers trick, but the prime powers trick uses exponentiation, and we do not have exponentiation in the ring language. It is a nontrivial fact that nevertheless for any primitive recursive function there is a formula in the ring language that captures it. To do it we need a trick, due to Gödel.

Gödel's idea is to represent a sequence by taking a fixed number modulo different numbers. For example, take the sequence 3, 5, 9, 6, 3, 0. We can generate this sequence by taking the number 19 modulo the bases 4, 7, 10, 13 and 16.

So, given a sequence $\langle a_0 \dots a_{k-1} \rangle$, how do we find a number, n, and an easily generatable set of k numbers—call them $g_0 \dots g_{k-1}$—to use as moduli? If the g_i are pairwise coprime then we can easily find n: The Chinese Remainder Theorem!

So the challenge is to find an easily generatable sequence. How much easier can we get than a linear progression? So if we decide that the moduli are in the form $g_i = a + bi$ then the work is cut down. We wish to apply the Chinese Remainder Theorem to it, so we need all the k terms to be pairwise coprime. The key observation is that $N! + 1$ is prime to every number $\leq N$.

We leave it to the reader to check that the numbers

$$N! + 1, \quad 2 \cdot N! + 1, \quad 3 \cdot N! + 1, \quad 4 \cdot N! + 1, \dots (k-1) \cdot N! + 1$$

are pairwise coprime, as long as $N \geq k$. The Chinese Remainder Theorem now tells us there is a solution n to the congruences:

$$\begin{gathered} n \equiv a_0 \pmod{N! + 1} \\ n \equiv a_1 \pmod{2 \cdot N! + 1} \\ \vdots \\ n \equiv a_{k-1} \pmod{(k-1) \cdot N! + 1} \end{gathered}$$

So now we have a number, n, such that modulo certain numbers in an arithmetic progression, we have the original numbers from the sequence. This

isn't quite what we want, since we want the *actual* numbers themselves, not things congruent to them.

The solution is as follows: If N is also greater than each a_i, then we would definitely have $a_i < N! + 1$ for each i. Now not only is n congruent to a_i modulo $N! + 1$, for each i, but furthermore a_i is the *least* positive number that is congruent to n modulo $N! + 1$.

We now define $[n]_p$ to be the least positive number that is congruent to n mod p. Observe that the relation $[n]_p = q$ can be captured in the ring language by

$$(\exists m)(n = m \cdot p + q) \wedge (0 \leq q < p)$$

(and indeed it is a primitive recursive relation). So, to summarise:

Lemma 1 *Given a sequence $a_1 \cdots a_k$ of naturals, there are natural numbers n and N such that:*

For each i with $0 < i \leq k$, $[n]_{i \cdot N+1} = a_i$

(The 'N' here is the earlier 'N!')

Not only can we code up finite sequences as naturals but the function that does it is primitive recursive.

So now there is no need to quantify over sequences. If we have defined $\theta(n, \vec{x})$ by recursion so that $\theta(0, \vec{x}) = \psi(\vec{x})$, and $\theta(n+1, \vec{x}) = \mu(n, \theta(n, \vec{x}), \vec{x})$, then instead of saying that there is a sequence s.t. ..., we can now say:

$$(\exists a)(\exists N)\left(\begin{array}{l} [a]_{N+1} = \psi(\vec{x}) \ \wedge \\ [a]_{nN+1} = y \quad \wedge \\ (\forall k < n)([a]_{(k+1)N+1} = \mu(k, [a]_{kN+1}, \vec{x})) \end{array}\right) \qquad ()$$

which says that $y = \theta(n, \vec{x})$.

And so we have represented recursion in the ring language!

So, for example, $n = 2^m$ can be represented as

$$(\exists a, A)([a]_{A+1} = 1 \wedge [a]_{mA+1} = n \wedge (\forall k < m)([a]_{A(k+1)+1} = 2 \cdot [n]_{kA+1}))$$

... and $x = y^z$ similarly. Of course once we have exponentiation we can, if we prefer, encode sequences by use of the prime powers trick and forget all about Gödel's trick.

"Beta function"? The upshot of all this is that there is a three-place function β such that whenever $\langle a_o \cdots a_n \rangle$ is a finite sequence of natural numbers, then there are $b, c \in \mathbb{N}$ such that $\beta(c, d, i) = a_i$ for each $0 \leq i \leq n$.

I am greatly endebted to Jack Webster and Imre Leader for explaining this material to me.

Things to check:

(i) I can't see offhand how to define factorials in the ring language, but that might not matter.

(ii) Presumably any multiple of big N will do. Have to think a bit about what happens if you think of a witness that is absurdly large. It presumably doesn't matter. (You have lots of numbers sitting around with nothing to do.)

(iii) There is something funny about the definability of exponentiation, as in exponential-diophantine as in Davis-Robinson-Matijasievich. What's the connection/difference? I think the answer is that in the Davis-Robinson-Matijasevich case what is at issue is whether or not $y = 2^x$ is definable in a much more impoverished *equational* language. (That is to say, the question is whether or not the relation is diophantine.) And that question is not answered by the device of Gödel used here.

Observe that the β function can be given a primitive recursive definition. (It wouldn't be much use to us if it didn't!)

E is the set of *elementary* functions: $\dot{-}$, zero, $+$, $\times$, bounded sum and product closed under composition but not primitive recursion. For every $f \in E$ there is $k \in \mathbb{N}$ s.t. $f(x)$ is dominated by $x \mapsto \beth_k x$. (That's a tower of 2s of height x.)

We can do this relative to f—giving us $E(f)$ the set of functions elementary in f. Sensible if f is superexponential. Then every function g in his new class is dominated by: $x \mapsto f^k(x)$.

Inductive definition of Kleene's $\mathcal{O}$, a set of notations for ordinals. It really is a set of ordinal notations. ($1 + \omega$ not the same as ω). If $a \in \mathcal{O}$ then $|a|$ is the ordinal denoted by a. Think of it as a set of strings or numbers, as you will.

Definition 24 *Put* 0 *in. (That's the numeral, which happens to be the number)*

If string s is in $\mathcal{O}$ add the string `succ`*::s. (This is often written $\langle 1, s\rangle$*

If $(\forall n)(\{e\}(n) \in \mathcal{O})$ (so we are thinking of $\mathcal{O}$ as a subset of $\mathbb{N}$) then put $\langle 1, e\rangle$ into $\mathcal{O}$. $\langle 1, e\rangle$ is a notation for the sup of (the denotations of) the notations (denoted by) values of the function $\{e\}$.

[*HOLE Aren't there more operations than this? Stan said that $1 + \omega$ is not the same string as $\omega + 1$ so presumably both these strings are in $\mathcal{O}$ ...*]

We think of $\mathcal{O}$ as a set of natural numbers. Membership of $\mathcal{O}$ is emphatically **not** decidable because of the universal quantifier. $\mathcal{O}$ is equipped with a kind of valuation function: $|-|$. $|s|$ is the ordinal denoted by the string s.

It turns out that $\mathcal{O}$ is Π^1_1 complete: every Π^1_1 set is many-one reducible to Kleene's $\mathcal{O}$. That is to say: For all Π^1_1 sets S there is a recursive g s.t. $(\forall x)(x \in S \longleftrightarrow g(x) \in \mathcal{O})$.

The point about Π^1_1 sets is that Π^1_1 means "inductively defined": an inductively defined is an intersection of all sets satisfying a conditon we can express using merely restricted quantifiers–namely, being closed under some operations.

We need the concept of a *path through Kleene's $\mathcal{O}$* (which Stan doesn't define). I think P is a path if it is a function $\mathbb{N} \to \mathcal{O}$ whose range is unbounded.

If P is a Π^1_1 path through $\mathcal{O}$ you can't get all the recursive functions:

Remark 17
If P is a Π^1_1 path through $\mathcal{O}$ then $\bigcup_{a \in P} E(F_\alpha)$ is not the set of all recursive functions

Do we mean recursive or total recursive? The latter, surely ...
Proof: Suppose P were such a path. Then

$$(\forall c)(c \in \mathcal{O} \longleftrightarrow (\exists \text{ recursive } F)(\forall a)(f \in E(F_a) \wedge a \in P \rightarrow |a| > |c|))$$

But this would make $\mathcal{O}$ into a Σ^1_1 set. (The occurrence of '$a \in P$' is negative; '$|a| > |c|$' is decidable—once you know that a and c are in $\mathcal{O}$. Not entirely sure why.). ■

So what kind of classes *can* be characterised as $\bigcup_{a \in P} E(F_\alpha)$ for a Π^1_1 path P through $\mathcal{O}$?

Answer: If T is a recursively axiomatisable arithmetic theory then the class of functions provably total in T may well be of this form.

For example:

- The functions provably total in the fragment of PA with Σ_1 induction are precisely the primitive recursive functions, and these are the functions in $O(F_\omega)$.

- A much more far-reaching result is Buchholz-Wainer:

  Every computable function that is provable total in PA is dominated by f_{ϵ_0}.[8])

From: pax0@seznam.cz
To: fom@cs.nyu.edu
Sent: Fri, Apr 8, 2011 11:03 am
Subject: [FOM] Chaitin's Omega/P vs. NP

Please does someone know if we knew the well-known Chaitin's Omega(the probability of halting a chosen universal Turing machine on a random input) to enough bits, then we could settle the P vs. NP problem?Thank you, Jan Pax

Joe Shipman says:

Not necessarily. With an oracle for the halting problem we would know whether there was a proof in ZFC or P = NP, a proof in ZFC of P not= NP, or neither, or both (in which case there would be bigger issues than P-NP to worry about). But "neither" would mean the P-NP problem was still unsettled.

We can ALMOST settle it though. There is a universal algorithm X to solve SAT which has the property that, if P=NP, X runs in polynomial time. We create a "clocked" version X' which runs X for (input length+1)^{googleplex} steps and then halts if X failed to halt, and another program X" which sequentially runs X' on all instances of SAT, and ask the oracle for the halting problem if X'' ever halts. If the answer is "no", then P=NP, and if the answer is "yes", then any polynomial-time algorithm for SAT is ridiculously infeasible. – JS

Exercise 96

Let J be Jockusch's partition of $[\mathbb{N}]^3$ from exercise 95, and fix a colour, χ. The elements of the tree will be [some] finite strictly ascending sequences of natural numbers, and the tree ordering will be end-extension. We use an ascending sequences s for the tree iff every increasing [?] triple from s is coloured χ. All sequences of length 1 or 2 are allowed in free. (In the case of length 2, one demands that the sequence be strictly ascending).

This condition on the sequences is decidable. If s is of length n one performs $\binom{n}{3}$ calculations (at most). If p is an infinite path through the tree, let P be the set of integers along it. Then P is monochromatic for J. Hence P is not decidable, and therefore p is not either.

Another answer supplied by Zachiri McKenzie, doctored by me.

I write '$2^{<\omega}$' for the set of all finite sequences from $\{0,1\}$ (we think of finite sequences as functions with domain an initial segment of $\mathbb{N}$) and '2^{ω}' for the set of all functions with domain $\mathbb{N}$ and range a subset of $\{0,1\}$. Define

$$d(n) = \begin{cases} 0 & \text{if } \varphi_n(n)\downarrow \neq 0 \\ 1 & \text{if } \varphi_n(n)\downarrow = 0 \\ \texttt{undefined} & \text{if } \varphi_n(n)\uparrow \end{cases}$$

It is clear that d is a partial computable function with domain $\{0,1\}$. Moreover, if φ_e is total then $d(e)\downarrow \neq e$.
For all $k \in \mathbb{N}$, define

$$d^{(k)}(n) = \begin{cases} d(n) & \text{if } d(n)\downarrow_k \\ \star & \text{otherwise} \end{cases}$$

The binary function $d^{(\cdot)}(\cdot)$ is total computable. Define

$$T = \{\sigma \in 2^{<\omega} \mid \text{for all } 0 \leq i < |\sigma|, (d^{(|\sigma|)}(i) = \star) \vee (\sigma(i) = d^{(|\sigma|)}(i))\}.$$

Now, $\langle T, \subseteq\rangle$ is a computable tree. Since $T \subseteq 2^{<\omega}$, T is finitely branching.
Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be such that for all $n \in \mathbb{N}$, if $d(n)\downarrow$ then $f(n) = d(n)$. Now, for all $k \in \mathbb{N}$, $f \upharpoonright k \in T$ (f restricted to k is in the tree). So, T is indeed infinite (and has a path f). Now, suppose $f = \varphi_e$. Then $d(e)\downarrow$ and $f(e) = \varphi_e(e) = d(e)$, which is a contradiction. Therefore f is not computable.

Exercise 99

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'?

Answer:

He meant that $\{n : \{n\}(n)\uparrow\}$ is productive. Let $\{n\}$ be a function whose domain of definition is a subset of this set. Is n a member of $\{n : \{n\}(n)\uparrow\}$? Can't be! so $\{n\}(n)\uparrow$.

Exercise ??

B is many-one reducible to A (written $B \leq_m A$) if there is a total computable f s.t $(\forall n)(n \in B \longleftrightarrow f(n) \in A)$.

Turing-reducibility is slightly more subtle.

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 A?" 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.

We can also do it as follows: the program for f is allowed to ask for $g(n)$, and it will be given a value or 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 $\lambda n.(\texttt{if } n \in A \texttt{ then } 1 \texttt{ else fail})$ or $\lambda n.(\texttt{if } n \in A \texttt{ then } 1 \texttt{ else } 0)$.)

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 $\{0,1\}$. In this setting, where we are considering recursion relative to an oracle, we let $\{e\}$ be the eth 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 eth program that calls the oracle C".

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 $\{e\}^{\eta_a}(x)\uparrow$.

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 $A \not\leq_T B \not\leq_T A$. 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's length(α_{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 $\{0,1\} \setminus \{\{s\}^{\eta_a}(x)\}$. And β_{2s+1} is set to be $\beta_{2s+1}{::}0$. 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's length(β_{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 $\{0,1\} \setminus \{\{s\}^{\eta_b}(x)\}$. And α_{2s+2} is set to be $\alpha_{2s+1}::0$. If there is no such η_b then set $\alpha_{2s+2} := \alpha_{2s+1}::0$ and $\beta_{2s+2} := \beta_{2s+1}::0$.

9.1 Questions for Tripos 2013

Oquestion 1 *What is a regular expression?*

State and prove Kleene's theorem that the set of strings accepted by a deterministic finite state machine is captured by a regular expression.

Oquestion 2 *Let $\mathfrak{M}$ be a machine (architecture unspecified) that halts on all inputs. Is there a computable function f such that $f(n)$ bounds the time taken by $\mathfrak{M}$ to halt on input n? Suppose the function computed by $\mathfrak{M}$ is not total ... what then? What happens if $\mathfrak{M}$ is the function that enumerates the output of a volcano for the halting set? Let $G(n)$ be the time taken by the volcano for the halting set to emit n numbers. Is $G(n)$ computable? [obviously!!]*

Oquestion 3 *What is a typed λ-term? Explain the connection with constructive propositional logic. What is a Church numeral? Supply λ-terms for successor, addition, multiplication and exponentiation on Church numerals. What is the Y combinator? Sketch how it can be used to find a λ term for every computable function.*

Need some questions on automatic structures.

Oquestion 4 *State and prove Kruskal's theorem on wellquasiordering finite trees.*

Oquestion 5 *State and prove the extended omitting types theorem for propositional logic*

9.2 Questions for Tripos 2014

Oquestion 6 *Prove Tennenbaum's theorem that there is no recursive nonstandard model of Peano Arithmetic.*

Oquestion 7 *What is an automatic group? Prove that a product of finitely many automatic groups is automatic.*

Oquestion 8 *Prove Scott's Isomorphism theorem for $\mathcal{L}_{\omega_1,\omega}$*

Perhaps a question on Quine's trick

Oquestion 9 *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?*

Oquestion 10 *Prove cut elimination for first-order logic*

Oquestion 11 *What is a primitive recursive function $\mathbb{N}^k \to \mathbb{N}$? Under what operations in this class of functions closed? Is every total computable function $\mathbb{N}^k \to \mathbb{N}$ primitive recursive?*

Oquestion 12 *State and prove the Myhill-Nerode theorem.*

Oquestion 13 *Find a proof for $((((A \to B) \to A) \to A) \to B) \to B$ using only the rules for $\to$ and the "identity rule"*

$$A \quad B$$

$$\overline{A}$$

Then decorate your proof appropriately with λ-terms.

9.2.1 Answers

Question 11 Bookwork, but if it is done properly it will take a while, and it will be strictly marked. They will need to define primitive recursive relations, and if-the-else, show that bounded quantification and bounded sum and product preserve primitive recursiveness.

My guess is that most of them will do this question, and that as a result we might be a decent spread of marks.

9.3 Questions for Tripos 2015

State and prove Friedberg-Muchnik

State and prove Myhill-Nerode

State and prove Tennenbaum's theorem

question about immune sets

State and prove Kruskal's theorem on the wellquasiordering of trees. Deduce Friedman's Finite form and explain its significance.

Chapter 10

Dear Thomas,

The packing is going well so I have stolen a few more moments for mathematics. The ***file at the end of this email*** is from

www.geom.uiuc.edu/docs/forum/automaticgroups

It says (I think) that all discrete (finitely generated?) Euclidean groups (hence frieze and wallpaper groups) are automatic groups.

Please keep me informed if you find anything interesting. best wishes, Alan

==========

On Jun 13 2012, Thomas Forster wrote:

Dear Alan,

Thanks for prompt and detailed reply!

Benson Farb sent me this link:

`http://retro.seals.ch/digbib/view?rid=ensmat-001:1992:38::528&id=&id2=&id3=`

On Wed, 13 Jun 2012, A.F.Beardon@dpmms.cam.ac.uk wrote:

Dear Thomas,

I am just 'clearing' some emails before leaving (in a few hours time) for South Africa - hence the (for me) unusually prompt reply!

I do not know much about automatic groups, and have just looked them up on the web. Clearly I do not have time to absorb the material before my flight today, but I will return to it later (I will be 'doing maths' for five weeks in South Africa). Can you give me a brief account of automatic groups by email in return for my thoughts about their relation with frieze groups?

I will send you a summary of everything I know (I am writing this stuff up for Part III) plus all the links I have.

Presumably you have internet etc in S.A.?

More later..

Thanks *very* much!!!

Thomas

Frieze groups are interesting: here are two things which do not seem to be (well) known; I am trying to get the second one published.

(1) The best way to show there are only seven frieze groups is to consider a frieze group G and form the quotient group G/T, where T is the subgroup of translations. This is a finite Euclidean isometry group of a cylinder (imagine

wrapping a paper frieze around a cylinder whose circumference is the minimal translation length in the group). This leads to an almost trivial - and certainly efficient - division into seven classes. The problem is that most people want to study frieze groups before they study quotient groups!

(2) The problem of classification generalises. Let T be a non-trivial group of real translations of the complex plane. We do NOT assume that T is discrete (for example, T could be the group of all rational translations); if T is discrete (and therefore cyclic) the discussion below simply repeats the theory for frieze groups.

Now consider any group G of isometries of the complex plane which (i) leaves the real axis invariant, and (ii) has T as its subgroup of translations. QUESTION: how many conjugacy classes for G are there (the answer will, of course, depend on T; if T is cyclic, the well known answer is seven).

I have answered this; if, for example, T is the group of all rational real translations then there are exactly five conjugacy classes.

The point is that the discreteness (as in the frieze groups)is ONLY necessary to draw pictures; if one abandons the urge to draw pictures then the algebra takes over (although the work is a little harder).

I can send you accounts of either, or both, of these if you wish. Clearly the problem about automatic groups and frieze groups could be asked about what I call generalised frieze groups as in (2) above.

best wishes, Alan

On Jun 13 2012, Thomas Forster wrote:

Alan, I have just discovered automatic groups and i am trying to learn about them double-quick so i can say something illuminating about them in my Part III computability course next year. You know about frieze groups, i remember.. What is the connection (if any) between frieze groups and automatic groups? Presumably every frieze group is automatic but not vice versa..? Presumably every Cayley graph of a frieze group is a frieze but not vice versa..?

I apologise for peppering you with these questions, doubly so if you had been counting on a peaceful retirement!

Thomas

10.0.1 Friedberg-Muchnik

When we introduced volcanos we left it open whether the volcanos store the partial computations, their work-in-progress, so that when a volcano looks again at the computation with input k it has ready to hand the earlier stages of the computation for this input. For the Friedberg-Muchnik theorem it is important that our volcanos do *not* do this, but instead start again from the beginning each time. It is worth noting (thank-you, Adrian Mathias!) that our volcanoes repeat themselves. If $\{i\}(3)\downarrow$ we keep on computing it infinitely often.

We have to construct two semidecidable sets A and B with the property that χ_A is not recursive in B and χ_B is not recursive in A. Another way of putting it: neither of χ_A and χ_B can be computed from the other.

This means that we have to ensure that, for each i, χ_A is not the ith total function recursive in B, and that for each i, χ_B is not the ith total function recursive in A.

We try to build two sets A and B by starting with two empty bins, and surrounding each with volcanoes for the functions-that-consult-an-oracle, one collection of volcanoes for functions that will be told to consult A and the other for functions that will be told to consult B. If a volcano finds that it needs to know whether or not $15 \in A$ and A doesn't yet know, then it goes to the next calculation.

Initially the two bins are empty. However at least some two-valued $\{i\}$ can produce some output even without consulting an oracle and as soon as one of them (consulting the B bin) announces (say) "$\{i\}^B(17) = $ No!" we put 17 into the A bin in order to ensure that $A \neq \{i\}^B$"$\mathbb{N}$. Volcanoes consulting the A bin analogously put numbers into the B bin—or insist that they be left out. How do we interpret the utterances of a volcano? When a volcano emits a number, it always says what the corresponding input was. We interpret '0' as "Input is not in my set" and 1 "Input is in my set"; any other output is a warning that the function being computed by the volcano is not a characteristic function, so we ignore it. In fact, we dowse the volcano altogether.

What happens—in our illustration above—if, for some $j \neq i$, the volcano for $\{j\}^B(17)$ at some later time says "Yes"... thereby apparently committing us to *not* putting 17 into A after all? What we do depends on whether $j > i$ or $j < i$. If $j > i$ we simply ignore it for the moment, with a view to giving that volcano another chance. [we will explain later why this is OK, he says optimistically]. However if $j < i$ we allow the volcano for $\{j\}^B(17)$ to win the argument, and decide to not put 17 into A after all. This of course also means that we have to discard all inferences about membership in B made on the basis that $17 \in A$. Since there are only finitely many $j < i$ then we can change our minds about whether or not $17 \in A$ only finitely often.

Two things to sort out. (i) the postponement alluded to in the square bracketed passage above and (ii) the question of what happens if no $\{i\}^B$ ever expresses a view about whether or not 17 should be in A.

(i) is not a problem for the following reason. If $\{j\}^B$ is a volcano for a characteristic function it will emit opinions about whether or not $n \in A$ for *all* n, not just 17, whereas it can be overruled or anticipated only *finitely* often. On the other hand, if it is not a volcano of that kind then we don't care about it.

(ii) appears to be more serious. It appears to be entirely possible that none of the $\{i\}^B$ ever express a view on whether or not 0 should be in A, and this is because, for every one of these $\{i\}^B$, the computation of $\{i\}^B(0)$ asks whether or not (say) $17 \in B$; and none of the $\{i\}^A$ ever express a view on whether or not 17 should be in B, and this is because, for every one of these $\{i\}^A$, the computation of $\{i\}^A(17)$ asks whether or not $0 \in B$!

However it's easy to see that this deadlock cannot occur. At least some of the $\{i\}^B$ do not consult B when trying to decide what to do to 0. If any reader disputes this, I can write such a programme in front of their eyes.

We also need to check, for A and for B, that—for any i— $A \neq (\{i\}^B)^{-1}$"B.

How can we ensure that every i gets avoided? Well, we only need to avoid those i that are characteristic functions. Every volcano that corresponds to a total function $\mathbb{N} \to \{0,1\}$ gets attended to.

The only trouble with this construction is that the sets that it constructs are not semidecidable! This is easy to see. $\mathbb{N} \backslash A$ has the same status as A, so if A were semidecidable so too would $\mathbb{N} \backslash A$. A and B are in fact both Δ_2. How so? We know that, for every $n \in \mathbb{N}$ there is either a stage s s.t. for all $s' > s$ n is in A at stage s' or there is a stage s s.t. for all $s' > s$ n is not in A at stage s'. So $n \in A$ iff it is in A for all sufficiently late stages s or (equivalently) for arbitrarily late stages s.

All of which reminds me that we should have a proof that every $\Pi_1 \cap \Sigma_1$ expression is equivalent to a quantifier-free expression.

A reply from Richard K

This is the story from the point of view of PA(L) i.e. PA with the full induction scheme for all of L in a language L containing + S and times and a few basic properties of these (I usually take the axioms of a discretely ordered semiring). Obviously we will be looking at subtheories of PA(L).

A formula is Δ_0 if all its quantifiers are restricted in your sense. It is Σ_n if it is $\exists x_1 \forall x_2 ... Q x_n) \phi$ where ϕ is Δ_0. The x_1, ... x_n may be tuples of variables or omitted. Restricted and unrestricted quantifiers cannot be mixed unless you count the restricted ones that are outside unrestricted ones as if they are unrestricted.

Dually Π_n

$I\Delta_0$, $I\Sigma_n$, $I\Pi_n$, etc are the theories consisting of the base theory plus induction on formulas with parameters from the formula class.

For $n \geq 1$, $B\Sigma_n$ is usually taken to be the base theory "discretely ordered semiring plus $I\Delta_0$" plus your collection scheme for formulas with parameters that are Σ_n

$\forall (parameters)[(\forall x < y)(\exists z)\phi(x,z,y) implies (\exists w)(\forall x < y)(\exists z < w)\phi(x,y,z)]$

the other direction follows from the axioms of an ordered semiring.

Let's write Coll(Σ_n) for the same, except that $I\Delta_0$ is NOT included in the base theory.

Then the old results (not too difficult) are, for $n \geq 1$

(*) $B\Sigma_{n+1} \to I\Sigma_n \to B\Sigma_n$

In other words, you need induction to prove the collection scheme, and (modulo a very weak system, $I\Delta_0$) the collection scheme implies some induction as well. In fact (modulo $I\Delta_0$) the full collection scheme implies full PA. Now that you know what you need to prove it, the first implication should be easy.

Slightly harder, the implications in (*) cannot be reversed.

If you are worried about Π_n the results are

$I\Sigma_n \longleftrightarrow I\Pi_n$

$B\Sigma_n \longleftrightarrow B\Pi_{n-1}$

The first of these sometimes causes problems, but remember parameters are allowed in the induction scheme. (Proof theory people often have parameter-free versions for which $I\Sigma_n \longleftrightarrow I\Pi_n$ is false.) The second is easy—treat a whole block of quantifiers like a single quantifier with a pairing function.

The Coll() scheme, i.e. without the $I\Delta_0$ part of the base theory for $B\Sigma_n$ is quite interesting. In fact it is proof theoretically very weak on its own (without any induction added to it) but it is still nontrivial. I certainly don't claim to know all about it, nor do I think anyone does.

Take a countable model $\mathfrak{M}$ of the base theory "discretely ordered semiring". Any model–it could satisfy something very bad such as having an irreducible element that isn't prime (this is a Σ_1 sentence). Then by easy algebraic means there is an end extension that is also a "discretely ordered semiring". (Take the ring of polynomials over this and order it suitably.) And we repeat the process taking unions at limits. After ω_1 times we get a model of size $\aleph_1$ for which all proper initial segments are countable. (It is an "ω_1-like model".) Then this ω_1-like model obviously satisfies the full collection scheme, but not the $I\Delta_0$ (because $I\Delta_0$ says an irreducible is prime). And it is still very weak—it still has an irreducible that is not prime.

Conversely, there is a straightforward model theoretic result that any *countable* model of the full collection scheme, whether or not it satisfies induction, has an elementary end-extension. (This is not the MacDowell Specker theorem which says that any (possibly un)countable model of PA has an elementary end-extension. It's a lot easier than MD-S and just needs the omitting types theorem.) Repeating this ω_1 times we get the same thing: Collection is precisely the theory of ω_1 models, and such things can be very very weak, but if they satisfy a little bit of induction ($I\Delta_0$ is enough, actually much less is needed) then they satisfy full induction.

All the above is in my models of PA book.

An old result of mine says a κ-like model of $I\Delta_0$ need not satisfy full induction if κ is not regular.

How much induction is needed to add to collection to PA is an interesting research problem. Induction on formulas that are "restricted existential" would do. (That's not so much...)

There's a dual to the collection scheme, called "Keisler's Axiom 4" (or is it Axiom 5? I forget) which you might find out about too. But it's the same.

I am not 100% sure what happens without multiplication. It might be similar, but there could be some basic trick to do with QE for presburger that makes things look rather different for very special reasons that are specific to presburger and usually unreproducible and unhelpful elsewhere (i.e. for any other base theory/language). If you really want this for presburger let me know.

Finally

If T is a theory of arithmetic (e.g. a sub theory of PA) and we have a formula $\theta(\bar{x})$ which is T-equivalent both to a Π_1 formula $(\forall x)\phi(x,\vec{z})$ and to a Σ_1 formula $(\exists y)\psi(y,\vec{z})$ we say for obvious reasons that the formula $\theta(\bar{x})$ is "provably recursive in T".

Lots of people have done lots of work trying to identify what "provably recursive in T" is for various T.

In the particular case $T = I\Sigma_1$ "provably recursive in T" is precisely "primitive recursive" i.e. all PR predicates can be so-represented in $I\Sigma_1$, and conversely.

In the case T=PA, "provably recursive in T" is precisely "primitive recursive" except you are allowed to do primitive recursions along the canonical well-orderings of length α, for any fixed $\alpha < \epsilon_0$. You are not allowed to do primitive recursions along ϵ_0.

And so on for other theories. But the details might be messy.

As it happens, "provably recursive in $I\Sigma_n$" is the same as "provably recursive in $B\Sigma_{n+1}$" for each n. I don't know if it helps you here, but it might. So sometimes working in $I\Sigma_n$ you can WLOG assume the $B\Sigma_{n+1}$ scheme.

Richard

From: T.Forster@dpmms.cam.ac.uk [T.Forster@dpmms.cam.ac.uk]

Sent: 08 November 2012 10:04

To: ps218@cam.ac.uk

Cc: Richard Kaye

Subject: don't dob me in

Gentlemen,

It occurs to me that you two (a) might know the answer to this and (b) won't expose my ignorance to the world.

OK, the question concerns expressions in arithmetic with S, $+$ and perhaps $\times$. (Probably doesn't matter) And restricted quantifiers. The question that's bothering me is:

Can we push restricted quantifiers in past unrestricted quantifiers.

That is:

is

$(\forall x < y)(\exists z)\phi(x, z, y)$

equivalent to

$(\exists w)(\forall x < y)(\exists z < w)\phi(x, y, z)$

?

How do we prove it, and in what system of arithmetic can this be proved? Primitive recursive arithmetic..?

Also: if we have a formula which is T-equivalent both to

$(\forall x)\phi(x, \vec{z})$

and to

$(\exists y)\psi(y, \vec{z})$

where psi and phi both altogether lack unrestricted quantifiers, then is it also equivalent to

$\theta(\vec{z}, \vec{a})$

for some θ, (the $\vec{a}$ are constants, which I suspect we must need)

****

I know I should know this. It prompts yet again the reflection that announcing a course of lectures on a topic of which one's mastery is incomplete

concentrates the mind like nothing else. (I am lecturing part III computability next term!)

All enlightenment received with suitable displays of snivelling gratitude.

****

I am very happy to announce the official opening of the Patterns of Resemblance Ordinal Calculator.

`http://www.xamuel.com/patterns/`

The POR Ordinal Calculator does arithmetic on ordinals below the ordinal of Π^1_1-CA$_0$, notated using first-order additive Patterns of Resemblance, a beautiful combinatorial notation system discovered by Timothy J. Carlson.

Currently implemented operations are addition, comparison, multiplication, and base-omega exponentiation and logarithm. Future operations (epsilon, Veblen, etc.) are planned.

The calculator requires no download or installation. It works on any computer with an internet connection and a modern browser (including smartphones and tablets). Notations generated by the calculator come with unique, permanent, public IDs which can be used to easily share patterns with other researchers or simply return to your own calculations at a later time.

A few samples:

The ordinal ϵ_0: `http://www.xamuel.com/patterns/du/`

The ordinal Γ_0: `http://www.xamuel.com/patterns/dy/`

The Howard-Bachmann ordinal: `http://www.xamuel.com/patterns/jz/`

Suppose you want to calculate the product $\Gamma_0 * \epsilon_0$. Simply enter the command: mult dy du

The calculator has an interactive tutorial which can be started by entering the command, "tutorial".

I hope you will find the calculator fun and useful.

-Samuel Alexander

Lemma 2 *If $\mathcal{F}$ is Schmidt-coherent, λ is limit and $n \in \mathbb{N}$ then $\mathcal{F}^{\beta,n}$, defined by*

$$\mathcal{F}^{\lambda,n}\ \lambda\ m =: \mathcal{F}\ \lambda\ (m+n);\quad \mathcal{F}^{\lambda,n}\ \beta\ m = \mathcal{F}\ \beta\ m \text{ for other } \beta$$

... is also Schmidt-coherent.

Proof: It will suffice to show that $\mathcal{F}\ \lambda\ 0 <_{\mathcal{F}^{\beta,n}} \lambda$. But—since $\mathcal{F}$ is Schmidt-coherent we have $\mathcal{F}\ \lambda\ 0 <_{\mathcal{F}} \lambda$. Hence—by the definition of $\mathcal{F}^{\beta,n}$—we have $\mathcal{F}^{\beta,n}\ \lambda\ 0 <_{\mathcal{F}^{\beta,n}} \mathcal{F}^{\beta,n}\ \lambda\ n$. But this last ordinal is the $<_{\mathcal{F}^{\beta,n}}$-predecessor of λ, whence $\mathcal{F}\ \lambda\ 0 <_{\mathcal{F}} \mathcal{F}\ \lambda\ n <_{\mathcal{F}^{\beta,n}} \lambda$.

Lemma 3 *Let $\mathcal{F}$ be a Schmidt-coherent system of fundamental sequences for Δ an initial segment of the second number class, and suppose $\alpha < \beta \in \Delta$. Then there is a system $\mathcal{F}^{(\alpha,\beta)}$ of fundamental sequences*[1] *for Δ such that*

[1] This is my notation not hers, and i've put in the brackets to make it less likely that readers will confuse it with the "$\mathcal{F}^{\alpha,\beta}$"

1. *$\mathcal{F}^{(\alpha,\beta)}$ is Schmidt-coherent;*
2. *$\alpha <_{\mathcal{F}^{(\alpha,\beta)}} \beta$ and*
3. *for all $\delta \leq \alpha$ we have $\mathcal{F}^{(\alpha,\beta)}\delta = \mathcal{F}\delta$.*

Proof:
(lifted brazenly from [55])
We define a sequence $\langle \gamma_n, \mathcal{F}_n \rangle$ as follows.
$\gamma_0 =: \beta$, $\mathcal{F}_0 =: \mathcal{F}$;
Thereafter

- if $\gamma_n = \alpha$ then $\gamma_{n+1} =: \alpha$ too, and $\mathcal{F}_{n+1} =: \mathcal{F}_n$;
- if $\gamma_n = \delta + 1 > \alpha$ then $\gamma_{n+1} =: \delta$ and $\mathcal{F}_{n+1} =: \mathcal{F}_n$;
- if $\gamma_n > \alpha$ and is a limit, and m is minimal such that $\mathcal{F}\gamma_n m \geq \alpha$ then $\gamma_{n+1} =: \mathcal{F}\gamma_n m$ and
- – if $\beta \neq \gamma_n$ then $\mathcal{F}_{n+1}\gamma q =: \mathcal{F}_n \gamma q$, and
  – if $\beta = \gamma_n$ then $\mathcal{F}_{n+1}\gamma q =: \mathcal{F}_n \gamma(q + m)$.

Using lemma 2 it is easy to show that

- $\mathcal{F}_n$ is Schmidt-coherent,
- $\gamma_n <_{\mathcal{F}_n} \beta$ or $\gamma_n = \beta$,
- $\mathcal{F}_n\delta = \mathcal{F}\delta$ for all $\delta < \alpha$,
- $\gamma_n \geq \alpha$.

Now $\langle \gamma_n : n < \omega \rangle$ is a nonincreasing sequence, so is eventually constant, so there is $n_0 \in \mathbb{N}$ such that $\gamma_{n_0} = \alpha$. Set $\mathcal{F}^{(\alpha,\beta)} =: \mathcal{F}_{n_0}$. ■

Lemma 4
Let $\mathcal{F}$ be a Schmidt-coherent system of fundamental sequences for Δ an initial segment of the second number class, and let λ be the smallest limit ordinal not in Δ. Then there is a Schmidt-coherent system $\mathcal{F}'$ of fundamental sequences for $\Delta \cup \{\lambda\}$.

Proof: Let $\langle \lambda_n : n \in \mathbb{N} \rangle$ be a fundamental sequence for λ. We define a sequence $\langle \mathcal{F}_n : n \in \mathbb{N} \rangle$ by recursion as follows. $\mathcal{F}_0 =: \mathcal{F}$ and thereafter $\mathcal{F}_{n+1} =: (\mathcal{F}_n)^{(\lambda_n, \lambda_{n+1})}$ as in lemma 3. Now—by that lemma (itemwise!)—for each $n \in \mathbb{N}$ we have

1. $\mathcal{F}_n$ is Schmidt-coherent;
2. $\lambda_n <_{\mathcal{F}_{n+1}} \lambda_{n+1}$;
3. $\mathcal{F}_n\delta = \mathcal{F}_{n+m}\delta$ for all $\delta \leq \lambda_n$ and $m \in \mathbb{N}$.

We can now set $\mathcal{F}'\beta$ to be

- $\langle \lambda_n : n \in \mathbb{N} \rangle$ if $\beta = \lambda$;
- $\mathcal{F}\beta$ if $\beta \leq \lambda_0$;
- $\mathcal{F}_{m+1}\beta$ if $\lambda_m < \beta \leq \lambda_{m+1}$.

$\mathcal{F}'$ obviously assigns fundamental sequences to everything in $\Delta \cup \{\lambda\}$.

■

THEOREM 23 *([55] theorem 2)*
Every proper initial segment of the second number class admits a Schmidt-coherent family of fundamental sequences.

Proof:
We prove by induction on 'α' that the countable ordinals strictly below α admit a Schmidt-coherent family.

The successor case is easy: if α is a successor of a successor, the assertion follows from the induction hypothesis; if α is the successor of a limit it follows from lemma 4 and the induction hypothesis.

So consider the case where α is limit.

Let $\langle \alpha_n : n \in \mathbb{N} \rangle$ be a fundamental sequence for α, and for each $n \in \mathbb{N}$ set $\sigma_n =: \Sigma_{m<n}\alpha_m$. Clearly $\alpha \leq \mathrm{Sup}(\{\sigma_n : n \in \mathbb{N}\})$.

By the induction hypothesis for each $n \in \mathbb{N}$ there is a Schmidt-coherent family $\mathcal{F}_n$ for the ordinals below $\alpha_n + 1$. We now define a family $\mathcal{F}$ as follows:

$\mathcal{F}\gamma m =:$

- 0 if γ is zero or a successor;
- $\sigma_n + (\mathcal{F}(\gamma - \sigma_n)m)$ otherwise, where n is maximal so that $\sigma_n < \gamma$.

Now for all μ and ν such that $\sigma_n < \mu \leq \sigma_{n+1}$ and $\sigma_n < \nu \leq \sigma_{n+1}$ we have $\mu <_{\mathcal{F}} \mu \longleftrightarrow (\mu - \sigma_n) <_{\mathcal{F}_n} (\nu - \sigma_n)$. Hence if γ is a limit ordinal and $\sigma_n < \gamma \leq \sigma_{n+1}$ then $\gamma - \sigma_n$ is also a limit, and since $\mathcal{F}_n$ is Schmidt-coherent we have $\mathcal{F}(\gamma - \sigma_n)m <_{\mathcal{F}_n} \mathcal{F}(\gamma - \sigma_n)(m+1)$ for each $m \in \mathbb{N}$. Thus $\mathcal{F}\gamma m = \sigma_n + (\mathcal{F}(\gamma - \sigma_n)m) <_{\mathcal{F}} \sigma_n + (\mathcal{F}(\gamma - \sigma_n)(m+1)) = \mathcal{F}\gamma(m+1)$. So $\mathcal{F}$ is Schmidt-coherent.

■

Can we omit 'proper' from the statement of theorem 23? We proved it without any use of AC.

First I tell 'em what i'm going to tell 'em ...

The plot runs as follows. Clearly any finite set $X \subset \mathbb{N}$ can be encoded as a natural number. We show how, in any nonstandard model, the *the standard part* of any *decidable* $X \subseteq \mathbb{N}$ can be similarly encoded by a (nonstandard) natural. Then, using the existence of a pair of recursively inseparable semidecidable

sets, we find an *un*decidable $X \subset \mathbb{N}$ whose standard part can be coded by a [nonstandard] natural. Then we show that if the addition relation of the model were decidable then this X would be decidable too. And it ain't.

. . . then I tells 'em:

Clearly any finite set $X \subset \mathbb{N}$ can be encoded by a single natural number using a prearranged scheme, as it might be $\Sigma_{n\in X} 2^n$. [You saw this in a question on the third example sheet from last year's Part II Set theory and Logic: define E so that $\langle \mathbb{N}, E\rangle \simeq \langle V_\omega, \in\rangle \ldots$]. Let $x\,E\,y$ be the relation "the xth bit of the binary expansion of y is 1". Clearly the graph of E is a decidable set.

So what we can do is the following.

state this properly

Next we show that there is an undecidable X that is coded . . .

Suppose $\mathfrak{A} \subseteq \mathbb{N}$ and $\mathfrak{B} \subseteq \mathbb{N}$ are semidecidable and recursively inseparable. (There are such sets, by remark 10 on page 112.)

By remark 6 on page 100 (that says that every semidecidable subset of $\mathbb{N}^k$ is a projection of a decidable subset of $\mathbb{N}^{k+1}$) we can safely assume that $\mathfrak{A} = \{n : (\exists x)A(n,x)\}$ and $\mathfrak{B} = \{n : (\exists x)B(n,x)\}$ where the predicates A and B are decidable.

Since $\mathfrak{A}$ and $\mathfrak{B}$ are recursively inseparable they are at least disjoint, so
$(\forall n)\neg((\exists x)A(n,x) \wedge (\exists x)B(n,x))$
$(\forall n)(\neg(\exists x)A(n,x) \vee \neg(\exists x)B(n,x))$
$(\forall n)(\forall x)\neg A(n,x) \vee (\forall x)\neg B(n,x))$
$(\forall n)(\forall x)(\forall x')(\neg A(n,x) \vee \neg B(n,x'))$
$(\forall n)(\forall x)(\forall x')(\neg(A(n,x) \wedge B(n,x')))$
So, in particular, for any standard m

$$(\forall n < m)(\forall x < m)(\forall x' < m)(\neg(A(n,x) \wedge B(n,x'))) \qquad \text{(O)}$$

Here we next have to use an *overspill* argument. Expression O is a formula with one free variable, and its extension cannot be precisely the standard naturals, co's o/w we could prove by induction that every natural was standard. So there must be a nonstandard natural—e, say—of which it holds.

$$(\forall n < e)(\forall x < e)(\forall x' < e)(\neg(A(n,x) \wedge B(n,x'))) \qquad \text{(O')}$$

. . . where we can take the variables 'n', 'x' and 'x'' to range over standard naturals.

Now let X be the standard part of $\{n : (\exists m < e)A(m,n)\}$. The stuff after the colon is decidable, so there will be $\mathfrak{x}$ such that $(\forall m < e)(y\,E\,\mathfrak{x}. \longleftrightarrow .y \in X)$

An old exam

Answer FOUR questions

Oquestion 14 *What is a regular expression? State and prove Kleene's theorem that the set of strings accepted by a deterministic finite state machine is captured by a regular expression.*

What is it for a nondeterministic machine to accept a string? Is there a version of Kleene's theorem for nondeterministic finite state machines?

This is a standard bookwork question, the meat of which is a complex induction. The second part of the question invites the candidate to explain why (by the power set construction) for any nondeterministic finite state machine there is a deterministic machine that recognises the same language.

OQUESTION 15 *(i) Let $\mathfrak{M}$ be a machine that halts on all inputs. Is there a computable function f such that $f(n)$ bounds the time taken by $\mathfrak{M}$ to halt on input n? Suppose the function computed by $\mathfrak{M}$ is not total ... what then?*

(ii) State and prove the Extended Omitting Types theorem for Propositional Logic

Clearly the answer to the first part is 'yes': attach a counter to $\mathfrak{M}$. If such a function could reliably be found even when the function computed by $\mathfrak{M}$ is not total then we would be able to solve the halting problem. To ascertain whether or not $\mathfrak{M}$ halts on an input i, compute the bound b for the time it would take to halt on input i. Then fire it up; if it hasn't halted within b steps it never will.

The second part is bookwork. Here is the relevant extract from the notes that were made available to the students.

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 $v(\sigma) = \texttt{true}$ 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 $\phi \to \sigma$ for every $\sigma \in \Sigma$, then $T \vdash \neg\phi$.

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 $T \vdash \phi \to \sigma$ for every $\sigma \in \Sigma$ but $T \nvdash \neg\phi$. But of course $T \vdash \top$. Contraposing, we infer that if $T \vdash \neg\phi$ for every ϕ such that $T \vdash \phi \to \sigma$ for every $\sigma \in \Sigma$ then there is a T-valuation omitting Σ. ■

However, we can prove something stronger.

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 $T \cup \{\neg A_1, \ldots \neg A_i\}$ 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 $T \cup \{\neg A_1, \ldots \neg A_i, \neg A_{i+1}\}$ 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 $T \cup \{\neg A_1, \ldots \neg A_i\}$ 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 $T \cup \{\neg A_1, \ldots \neg A_i, \neg A_{i+1}\}$ 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.

■

OQUESTION 16 *Give a direct definition of the factorial function* $\mathbb{N} \rightarrow \mathbb{N}$ *in the language of ordered rings. How long is your formula?*

Suppose we have defined f by primitive recursion: $f(0, \vec{s}) := g(\vec{s})$; $f(S(n), \vec{s}) := h(f(n), n, \vec{s})$. When we assert that $y = f(x, \vec{s})$ we are committing ourselves to being able to produce a chain of equations, which we can think of as a list of ordered pairs.

If f is a function $\mathbb{N}^k \rightarrow \mathbb{N}$ declared by primitive recursion then there is a formula $\phi(y, x_1 \ldots x_k, \vec{z})$ in the language with 0, 1, +, × and = ("the ring language") containing no unrestricted quantifiers such that $y = f(x_1 \ldots x_k)$ iff $(\exists z)\phi(x, \vec{s}, z)$

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', the s variables and 'y'. Recall that a *certificate that* $y = f(x, \vec{s})$ is a list of tuples .... Clearly $y = f(x, \vec{s})$ holds iff there is a certificate to that effect. So we need to be able to say that 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. This is straightforward if we have exponentiation in the language (we can use the prime powers trick, as noted above) but less straightforward if we don't, and we will need a new idea. However 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: '$(\exists I)(\exists O)(\exists p)(\ldots)$'. The prime p will be chosen big enough so that the following picture makes sense.

$\ldots$	p^2	p^1	p^0	
$\ldots$	2	1	0	I
$\ldots$	$f(2)$	$f(1)$	$f(0)$	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.

Some Local Definitions

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

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 $(\forall w < z)(w|z \to p|w)$—at least when p is prime. (And the task in hand does 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 remainder[2] then divide that remainder by (z/p), we find that the quotient is o_z."

In symbols:

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

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

(x `DIV` y is the largest integer z s.t. $y \cdot z \leq x$ and x `rem` y is the remainder when x is divided by y.)

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 $R(I, z, i)$) and 'o_z' for the O-entry at the place corresponding to z (i.e., the unique o such that $R(O, z, o)$).

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 nth 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

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

Oquestion 17 *What is a typed λ-term? Explain the connection with constructive propositional logic. What is a Church numeral? Supply λ-terms for successor, addition, multiplication and exponentiation on Church numerals. What is the Y combinator? Sketch how it can be used to find a λ term for every computable function.*

This is pure bookwork

Oquestion 18 *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 $\mathbb{N}$, with infinitely many equivalence classes, 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.

We build a transversal T in stages, T_n.

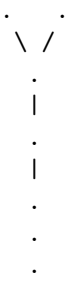
Put 0 into T_0. Compute $n \sim 0$ for all n in parallel. As soon as we discover a k such that $\neg(k \sim 0)$ we put k into T_1.

Subsequently at the nth stage we compute $[[k \sim m]]$ (the truth-value of $k \sim m$) for all $m \in T_n$ and all $k \notin T_n$. As soon as this process reveals a k such that $(\forall m \in T_n)(\neg(k \sim m))$ we set $T_{n+1} := T_n \cup \{k\}$. Since $\sim$ is of infinite index, there is such an m and—since the graph of $\sim$ is the complement of a semidecidable set—we will find it.

Oquestion 19 *State and prove Kruskal's theorem on wellquasiordering finite trees.*

Suppose we quasiorder finite trees as follows: $T \leq T'$ if there is an injection from the vertex set of T to the vertex set of T' that preserves adjacency. Is this a WQO?

The first part is bookwork. The answer to the second part is 'no': let T_i be a chain of length i with two endpoints—a forked tongue of length i.



Then the $T_i : i \in \mathbb{N}$ constitute an infinite antichain.

H I A T U S

One has to distinguish between the two situations:

1. We have a method which, when we are presented with a widget, enables us to construct an algorithm which recognises gagdets in polynomial time.

2. We have an algorithm which recognises gadgets in time which—if there is a widget—is only polynomial.

An example of the first is a result of Seymour-Robertson to the effect that there is a polynomial-time algorithm for recognising graphs not embeddable in surfaces of genus k. Richard Pinch has supplied me with the following example of something rather like the second

> Thomas,
>
> The precise result is that it is possible to prove that a number N is composite, using the well-known Miller-Rabin ('strong') test, if one can find a primitive root g of one of its prime factors p. The ERH implies that the least such g is $< (log(p))^2$, otherwise the best one can say at present is that $g < p^{1/3}$. So without the ERH one cannot prove that MR proves compositeness in polynomial time. (It might be that it does so even if ERH is false, of course.)
>
> Richard

A message from Ian Grant

Dear Thomas

I'm enjoying this. I needed some new input. I haven't read much yet, but there are a couple of typos on page 51. You mean η-reduces, not beta reduces in the times and exp lambda definitions.

Regarding the sentence that follows them: if `\n.\m.n m` is the identity function, then why is it extensionally equal to EXP m n? The identity function certainly isn't. It's true that `\m.\n.m n` η-reduces to the identity function, but Church numerals are not inert data, they are better thought of as iterated abstract operations. I.e. `\f.\x.f x` should be called "once", `\f.\x.f (f x)` "twice" etc.

The problem with LC is that it works too easily and we "read in" the semantics to Church numerals, pairs etc. In combinatory logic one is less inclined to do this because many different weak normal forms correspond to the same Church numeral. There it is very clear that you only have a representation of the naturals by extensional equality. So you have to apply each numerical result to the successor function, and apply the result of that to zero, before you can see the answer. Often it takes more reductions to reduce a weak normal form to a constant than it takes to reach the weak normal form, so it is much more clear that the Church numerals are processes, not data.

Of course extensional equality is not decidable in general, but if you know the types of the terms in question it is. However not all numerical functions have simple types. The λ term you give for the Ackermann function doesn't have a simple type. The usual subtraction function `\m.\n.n PRED m`. doesn't have one either. This is because the predecessor function does not return a value of the same type as the one it is given, it returns one one or two levels up, so one cannot iterate it arbitrarily many times as one does when one applies $\underline{n}$ to it. The Ackermann function using an iterator has no simple type for the same reason. Godel's system T is the simplest type system I know of that can type higher order functions like ACK and SUB. At the end of this mail there is an implementation of system-T in a few dozen lines of standard ML.

Another way to see what I mean is to look at the fully parenthesised expressions:

$$((\lambda f.\lambda x.fx)S)O \rightarrow (\lambda x.Sx) \rightarrow SO$$

So the term $\lambda f.\lambda x.fx$ is actually doing something: namely applying S to O which isn't happening in the original expression.

To get a denotation for lambda calculus we need to fix some constants in the domain: S and O for the successor and zero, say. Then what we mean by the 'object' two is the term $(\lambda f.\lambda x.f(fx))SO$ which reduces to $S(SO)$. The Church numeral $\lambda f.\lambda x.f(fx)$ is something distinct from the number two. It works much better in combinatory logic because there you *have* to evaluate values: for example if using Church numerals in SKI combinatory logic, when

you compute 2^2 using 22 you get the weak normal form

$$S(K(S(S(KS)(S(KK)I))(S(S(KS)(S(KK)I))(KI))))(S(K(S(S(KS)(S(KK)I))(S(S(KS)(S(KK)I))(KI))))I)$$

and to see extensional equality with the Church numeral 4:

$$S(S(KS)(S(KK)I))(S(S(KS)(S(KK)I))(S(S(KS)(S(KK)I))(S(S(KS)(S(KK)I))(KI))))$$

takes some more reductions: applying each to s and z, say, gives $s(s(s(sz)))$ in both cases.

And I think that's all there is to computation: it's just constructing left-folded forms (by application) so that, as they fold up again under conversion, they construct the right-folded values that represent the results. There is surely a connection with Lawvere's left and right adjoint functors and universality. I love reading about category theory, but it must be like taking psychedelic drugs, afterwards I find it hard to say anything more than "Wow, man, what a trip!" I'll feel I understand comma categories and adjuncts etc. when I can apply them to something concrete like a "Schonfinkel algebra": the PTJ exam question you set as an exercise is a lovely one. If you can see any way to go with something like this then let me know.

LC is full of gotchas. One that caught me was: β conversion preserves the set of free variables in a formula. If there are no free variables in a formula, then—since β conversion won't change this fact—there is never any need to do α conversion before substitution because there are no free variables to be captured. I am embarrassed to say that I actually believed this until only about a month ago. But it is surprising how far you can get without needing to α convert before substitution. ADD, PRED, SUB, MUL, ACK, FACT can all be done with naïve substitution. But applying 1 to 1, as you do in EXP m n, needs an alpha conversion. I imagine one could do exponentiation with Y though and not need capture-avoiding substitution.

> So you think I might just wave my arms over it?

Larry handwaves it very eloquently in his FoFP lecture notes. Pasting strips together is quite intuitive, and I believe all the tedious case analysis in the strip lemma. Here is something I once wrote about the strips. You'll need to run it through LaTeX (with the amsmath package loaded) to see what I mean:

An alternative definition of equality is in terms of multi-step reduction: Two terms M and M' are equal iff there exists a sequence of forward and backward reductions from M to M'. This is famously pictured:

$$\begin{array}{ccccccccc} M & & M_1 & & M_2 & \cdots & M_{k-1} & & M_k \equiv M' \\ & \searrow^* \quad {}^*\swarrow & & \searrow^* \quad {}^*\swarrow & & & & \searrow^* \quad {}^*\swarrow & \\ & N_1 & & N_2 & & \cdots & & N_k & \end{array}$$

From this diagram we can read off various properties of equality, such as

(i) If $M \to_* N$ or $N \to_* M$ then $M = N$.

Take $k = 1$ and put $M \equiv N_1$ or $N_1 \equiv M_1$.

(ii) If $L \to_* M$ and $L \to_* N$ then $M = N$.

Take $k = 2$ and put $L = M_1$ with $N_1 \equiv M$ and $N_2 \equiv M_2 \equiv N$.

(iii) If $M \to_* L$ and $N \to_* L$ then $M = N$.

Take $k = 1$ and put $N_1 \equiv L$ and $M_1 \equiv N$.

Best wishes
Ian

Question 23

Part (i) is pretty routine. You can't do it unless you have achieved a certain level of familiarity with the material, but once you are confident and fluent, you can answer a question like this (which the student will *not* have seen during lectures) merely by doing the obvious and following your nose.

Part (ii) is actually quite hard: altho' the understanding it requires is basic, it also requires persistence and a calm head.

My feeling is 10 marks for part (i) and 15 marks for part (ii). In part (ii) I would say 8 marks for the proof tree and 7 for the decoration.

I have bundled these two questions together because the students will like part (i) (looks like bookwork) and will prefer not to attempt part (ii) (which looks as if they might have to be creative). That way they have to do something demanding.

Part (i)

The best way to answer this question is to draw lots of pictures.

Clearly we are going to have to execute a back-and-forth construction.

Think of the naturals in $\langle \mathbb{N}, <_A \rangle$ as $0_A, 1_A, 2_A \ldots$ and think of the naturals in $\langle \mathbb{N}, <_B \rangle$ similarly as $0_B, 1_B, 2_B \ldots$.

Clearly we wish to pair 0_A with 0_B. What do we do thereafter? In the routine back-and-forth construction we seek, at stage n, a mate in $\langle \mathbb{N}, <_B \rangle$ for the first n_A we have not already found a mate for. We examine $0_B, 1_B \ldots$ and so on until we find one that lives in the open interval that qualifies it to be a mate for n_A. This process of checking involves asking questions like "$x <_B y$?" all of which are *ex hypothesi* answerable, since the graphs of $<_A$ and $<_B$ are decidable sets of ordered pairs. Then we come back the other way. At the end of time we have a bijection as usual.

Duplication with p 191

Part (ii)

Clearly the first step must be to decide to infer B from $(((A \to B) \to A) \to A) \to B$.

The only way to use the assumption $(((A \to B) \to A) \to A) \to B$ is to exploit it as the major premiss of $\to$-elimination, which means that we have to somehow obtain the minor premiss, namely $(((A \to B) \to A) \to A$. Now this last formula is not a constructive thesis (tho' it is a truth-table tautology) we will have to derive it from the assumption we already have, namely $(((A \to B) \to A) \to A) \to B$.

If we are to infer $((A \to B) \to A) \to A$ from $(((A \to B) \to A) \to A) \to B$, then clearly it means we have to infer A from $(((A \to B) \to A) \to A) \to B$ and $(A \to B) \to A$.

If we can deduce $A \to B$ from $(((A \to B) \to A) \to A) \to B$ we'll be all right. But this is easy, because A implies $((A \to B) \to A) \to A$ (use K). Thus we can obtain the following proof (The decorations are easy once one has the proof.)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x:A]^2 \qquad \dfrac{y:[(A\to B)\to A]^1}{x:A}\ \text{identity rule}}{\lambda y.x:((A\to B)\to A)\to A}\ \to\text{-int (1)} \qquad [z:(((A\to B)\to A)\to A)\to B]^4}{z(\lambda y.x):B}\ \to\text{-elim}}{\lambda x.z(\lambda y.x):A\to B}\ \to\text{-int (2)} \qquad [w:(A\to B)\to A]^3}{w(\lambda x.z(\lambda y.x)):A}\ \to\text{-elim}}{\lambda w.w(\lambda x.z(\lambda y.x)):(((A\to B)\to A)\to A)\to A}\ \to\text{-int (3)} \qquad [z:(((A\to B)\to A)\to A)\to B]^4}{z(\lambda w.w(\lambda x.z(\lambda y.x))):B}\ \to\text{-elim}}{\lambda z.(\lambda w.w(\lambda x.z(\lambda y.x))):((((A\to B)\to A)\to A)\to B)\to B}\ \to\text{-int (4)}$$

(10.1)

OQUESTION 20 *What are many-one reducibility and Turing-reducibility? State and prove the Friedberg-Muchnik theorem.*

OQUESTION 21 *(i)*

What is an immune set? What are incompressible strings? Show that the set of incompressible strings is immune.

(ii)

Explain how Friedman's finite form (FFF) of Kruskal's theorem follows from Kruskal's theorem. What is the significance of FFF?

OQUESTION 22 *Prove carefully that if f is a function $\mathbb{N}^k \to \mathbb{N}$ declared by primitive recursion then there is a formula $\phi(y, \vec{x}, \vec{z})$ in the language with 0, 1, $+$, $\times$, $<$ and $=$ ("the language of ordered rings"), containing no unrestricted quantifiers, such that $y = f(x_1 \ldots x_k)$ iff $(\exists \vec{z})\phi(y, \vec{x}, \vec{z})$*

OQUESTION 23 *Show that every partial computable function can be represented by a λ-term acting on Church numerals.*

OQUESTION 24 *What is the Goodstein function? Prove that it is total. Explain why this gives rise to a decidable wellordering of $\mathbb{N}$ of length ϵ_0.*

OQUESTION 25 *What is a recursively axiomatisable theory? Establish that any sufficiently strong sound recursively axiomatised arithmetic of the natural numbers is incomplete. What is a productive set? Establish that the set of arithmetic truths is productive.*

10.1 Questions for Tripos 2016

A question from Maurice about automatic groups

[6 marks]
Let G be a group, and (A, L) an automatic structure for G. Show that there exists a constant N such that, if $w \in L$ and $g \in G$ satisfy $g = \overline{wx}$ or $\overline{w} = g\overline{x}$ for some $x \in A \cup \{\epsilon\}$, then:
(i) g has some representative $v \in L$ of length $|v| \leq |w| + N$; and
(ii) If $u \in L$ is a representative of g with $|u| > |w| + N$, then there are infinitely many representatives of g in L.

Henceforth let G be a group, and (A, L) an automatic structure for G with A a symmetric set (that is, if $a \in A$, then $a^{-1} \in A$). Assume that all multiplier automata M_x ($x \in A \cup \{\epsilon\}$) are normalised; they have no inaccessible states, and all dead states are merged into one.

[8 marks]
Show that, given $u \in L$ and $x \in A$, we can algorithmically construct $u \in L$ with $\overline{v} = \overline{ux}$ in G.

[4 marks]
Using the result of Question 10.1, show that the algorithm you described to compute v in part 10.1 can be carried out in time $O(|u|)$, and moreover that v can always be constructed such that $|v| \leq |u| + N$ for some fixed N which depends only on the automatic structure (A, L).

[2 marks]
Show that, from (A, L), we can construct a word $\gamma \in L$ such that $\overline{\gamma} = 1$ in G.

[3 marks]
Let γ be as in Question 10.1. Given a word $w \in A^*$, show that we can construct a word $z \in L$ with $\overline{z} = \overline{w}$ in G, in time $O(|w|^2)$, with $|z| \leq |w|N + \gamma$.

[2 marks]
Conclude that there is an algorithm that, on input of any word $w \in A^*$ decides, in time $O(|w|^2)$, whether or not $\overline{w} = 1$ in G.

Oquestion 26 *Let A be an infinite set, G a subset of A^ω, the set of ω-sequences of elements of A. Players* `I` *and* `II` *alternately pick members from A (with replacement, so repetitions are allowed) thereby generating a play $p \in A^\omega$.* `I` *wins iff $p \in G$. Give A the discrete topology, A^ω the product topology. By using a fixed-point theorem or otherwise show that if $G \subseteq A^\omega$ is open then one of the two players must have a winning strategy.*

Oquestion 27 *(i) What is a primitive recursive function? Define an ω-sequence of functions $\mathbb{N}^2 \to \mathbb{N}$ of which the first three members are addition, multiplication and exponentiation. Prove that every function in your sequence is primitive recursive.*

(ii) Is every semidecidable set $X \subseteq \mathbb{N}$ the range of a primitive recursive $f : \mathbb{N} \to \mathbb{N}$?

Can one say anything about the fibres of f? I think every fibre has to be a decidable set.

Oquestion 28 *State and prove Friedberg-Muchnik*

Oquestion 29 *What are many-one reducibility and Turing-reducibility? State and prove the Friedberg-Muchnik theorem.*

Two marks for an explanation of computation relative to an oracle; Two marks each for the two definitions, leaving 21 marks for F-M.

I will certainly be looking for an appreciation that one only ever adds elements to a set-under-construction, and never deletes any—after all, the final products have to be semidecidable. An explanation of why this inability to delete elements doesn't conflict with the need to correct *injuries* will reassure the examiner that the student understands what is going on.

One should keep some marks up one's sleeve should any students wish to contrast the situation with Kleene-Post and present a proof of that. (It was in the handouts but was not lectured).

More specifically

Two marks for explaining "requirement";
Two marks for explaining "witness";
Two marks for explaining "injury";
Two marks for explaining "barrier";
Two marks for explaining "merits attention";
Two marks for explaining the list of candidates maintained by each requirement and a comment about why they don't need to be disjoint.

The lecturer lectured from the following notes. None of the diagrams on the blackboard are reproduced in the notes.

B is many-one reducible to A (written $B \leq_m A$) if there is a total computable f s.t $(\forall n)(n \in B \longleftrightarrow f(n) \in A)$.

We need the notion of computation relative to an oracle. Just 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 O?" [where 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 eth 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 eth program that calls the oracle C".

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. 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 $\lambda n.$(`if` $n \in A$ `then` 1 `else fail`) or $\lambda n.$(`if` $n \in A$ `then` 1 `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 give 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 the halting set and its complement) 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.

THEOREM 24 *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, beco's that would mean A and B could be enumerated in increasing order and that would make them decidable. 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 nonemepty intersection.

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 $A \neq \{e\}^B$"$\mathbb{N}$; $B \neq \{e\}^A$"$\mathbb{N}$, 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 eth 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 eth 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.

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 beco's 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 nth 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 dæmon 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 priority[3] 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 dæmon 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 beco's it could be that your decision that $\{17\}(x)\downarrow = 0$ happened beco's 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.

This will be a question for 2017

Let Σ be a finite alphabet, and $f : \mathbb{N} \to \Sigma$ an infinite string of elements of Show that f is eventually periodic iff $\{f \restriction n : n \in \mathbb{N}\}$ (thought of as a subset of $\Sigma^{<\omega}$ is a regular language over Σ.

The situation is subtle and complex, and there are rectypes of bounded character that might not be sets, even if we have replacement, for example the largest initial segment of the ordinals consisting entirely of ordinals of cofinality ω at most. [This sounds coinductive!] We need (a bit of) AC to show that this collection is not the paradoxical collection of all ordinals.

Exercise 102 *(*)*

Suppose that ϕ is a computable partial function of two arguments.

[3] Why stop there? Why not raise the barriers on *all* requirements? beco's you might end up going round in circles!

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 $\psi(m) = \mu x.(\phi(x, m) = 0)$.*

10.1.1 Finite Trees

I've sent you this material earlier but it could make sense to collate it here.

It is a standard fact that the class of Dedekind-finite sets is closed under "finite sequences without repetitions". (John Truss—from whom i learnt this fact—tells me that it was, indeed, proved by Tarski. See footnote below.) I don't remember ever seeing a proof, tho' i presumably must've, and in any case it's not hard to find one. I want to set it in a slightly more general context.

Lemma 5 *(No use of AC!)*

Given a repetition-free ω-sequence of repetition-free [finite] lists-from-X we can recover a repetition-free ω-sequence of members of X.

Proof:

We are going to describe an algorithm. Given the sequence of lists, look at the sequence of heads of the lists. If there are infinitely many that are distinct we obtain an ω-sequence of distinct members of X by ordering the elements by first appearance. If there are only finitely many distinct heads, then at least one element x_0 of X turns up as the head of infinitely many lists in the sequence, and there will be a first such element. Discard any list that does *not* have x_0 as its head. Now look at the second elements of the surviving lists (all but at most one of the surviving lists have a second element). Do the same, this time obtaining x_1. Iterate. At each step we either have infinitely many distinct elements (in which case we stop) or we procede to the next stage. If we never stop we end up with an ω-sequence of members of X. And it is without repetitions, because every initial segment of it is an initial segment of one (well, infinitely many) of the lists in our collection.

■

It is important that the proof we have just given is effective. It doesn't claim to be constructive (it uses excluded middle—infinitely often indeed) but at least it doesn't use AC.

Corollary 4 *(Tarski)*

If X is Dedekind-finite then the set of repetition-free finite lists from X is also Dedekind-finite[4].

[4]Dear Thomas,

Nice to hear from you. In my (very old!) paper, [3], this is given as Lemma 6. In the proof I say it is due to Tarski, and I refer to Levy's paper [2]. There Levy says that this was conveyed to him by Tarski ('oral communication').

I think that's the best I can do.

All the best, John

Proof: Contrapose lemma 5. Since X is Dedekind-finite the process cannot halt at a finite stage, and the infinite run constructs the ω-sequence for us.

■

Two questions come up here which i am not planning to pursue. (i) Can we do the same for sets lacking countably infinite partitions? (ii) Observe that even tho' this tree construction gives us larger Dedekind-finite sets it's not going to give us Dedekind-finite sets with large uncountable wellordered partitions.

10.1.2 A topological angle. . . ?

There is probably something helpful to be said about how the construction of the ω-sequence relies on the fact that the things we are trying to construct form a closed subset of a product space.

Something analogous to corollary 4 holds for Dedekind-finite trees . . . that are repetition-free (in the appropriate sense). We will need a carefully crafted definition.

Definition 25 *Define D-trees inductively as follows. A D-tree has a root $d \in D$ and the children form a repetition-free finite list of $(D \setminus \{d\})$-trees.*

This definition doesn't prevent a D-tree having multiple occurrences of an element but it does have the effect that no *branch* of a D-tree can have two occurrences of any one element. Indeed it may even be equivalent to that condition. No repetitions on any branch.

This following remark may be new, i don't know.

Remark 18 *If D is Dedekind-finite then the class of D-trees is also Dedekind-finite.*

Proof:

Suppose we have an ω-sequence of D-trees; we will show that they cannot all be distinct.

Start by looking at the roots. At least one d in D appears infinitely often as the root of a tree in our sequence. Put this d on one side and call it d_0; it's going to be the first member of a repetition-free ω-sequence of members of D.

Discard all the trees that have roots other than d_0. Look at the sequence of litters of the roots of the surviving trees. This is an ω-sequence of repetition-free finite lists of $(D \setminus \{d\})$-trees. Now we use the construction of lemma 5 to obtain an ω-sequence of $(D \setminus \{d_0\})$-trees. That is to say, from an ω-sequence of D-trees we have obtained both a member d_0 of D and an ω-sequence of $(D \setminus \{d_0\})$-trees.

In some sense we are in the situation we started with, or very nearly. We can repeat what we have just done on the repetition-free ω-sequence of $(D \setminus \{d_0\})$-trees. When we have done that we will have d_0, d_1 and a repetition-free ω-sequence of $(D \setminus \{d_0, d_1\})$-trees. By iterating we obtain an infinite (repetition-free) sequence $\langle d_i : i \in \mathbb{N} \rangle$ of elements from D.

■

I don't think i am being fanciful in saying that this proof provides an anticipation of Nash-Williams' proof of Kruskal's theorem.

And we should remember that Simpson's application of topological ideas to BQO theory was a huge liberation.

10.1.3 Multisets?

Is there anything useful to be said about multisets over a finite set?

Bibliography

[1] Aigner and Ziegler, "Proofs from THE BOOK", Springer, Berlin 1998, ISBN 3-540-63698-6

[2] Azriel Levy, 'The Fraenkel-Mostowski method for independence proofs in set theory', in 'The theory of models' North-Holland 1965, page 225 lines 16−20.

[3] John Truss Classes of Dedekind finite cardinals (Fund Math 84 (1974) 187-208)