Computation Theory

Exercises

2021

Contents

1. Algorithmically undecidable problems

1. Two important concepts in the theory of computability are *enumerations* and *diagonalisation*. Intuitively, an enumeration of a set S is an ordered, "exhaustive" listing of all elements. While this intuition works for finite sets, we need to be more formal to handle infinite sets. Thus, an *enumeration* of a finite or infinite set S is a surjective function from the natural numbers $\mathbb{N}$ to S, if it exists. If it does, the set S is called *countable*; if it doesn't, it is *uncountable*.

   Prove or disprove the following statements:

   a) The set of natural numbers is countable.

   b) The set of integers is countable.

   c) The Cartesian product of two countable sets is countable.

   d) The set of rational numbers is countable.

   e) The finite n-ary product of countable sets is countable.

   f) The set of polynomials with coefficients from a countable set is countable.

   g) The powerset of a countable set is countable.

   h) The set of real numbers is countable.

Feel free to do some research if you are not familiar with these results.

2. Rephrase the proof of the undecidability of the Halting Problem (with an abstract definition of an algorithm) as a diagonal argument.

2. Register machines

1. Show that the following arithmetic functions are all register machine computable.

   a) First projection function $p \in \mathbb{N} \to \mathbb{N}$, where $p(x,y) \triangleq x$
   b) Constant function with value $n \in \mathbb{N}$, $c_n \in \mathbb{N} \to \mathbb{N}$ where $c(x) \triangleq n$
   c) Truncated subtraction function, $\_ \dot{-} \_ \in \mathbb{N}^2 \to \mathbb{N}$, where

   $$x \dot{-} y \triangleq \begin{cases} x - y & \text{if } y \leq x \\ 0 & \text{if } y > x \end{cases}$$

   d) Integer division function, $\_\mathrm{div}\_ \in \mathbb{N}^2 \to \mathbb{N}$ where

   $$x \text{ div } y \triangleq \begin{cases} \text{integer part of } x/y & \text{if } y > 0 \\ 0 & \text{if } y = 0 \end{cases}$$

   e) Integer remainder function, $\_\mathrm{mod}\_ \in \mathbb{N}^2 \to \mathbb{N}$ with $x \bmod y \triangleq x \dot{-} y \cdot (x \text{ div } y)$
   f) Exponentiation base 2, $e \in \mathbb{N} \to \mathbb{N}$, where $e(x) = 2^x$

g) Logarithm base 2, $\log_2 \in \mathbb{N} \to \mathbb{N}$, where

$$\log_2(x) \triangleq \begin{cases} \text{greatest } y \text{ such that } 2^y \leq x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases}$$

Hint: instead of defining everything from scratch, try implementing these machines with the help of general control flow components.

3. Coding programs as numbers

1. *Gödel numbering* is a general technique for assigning a natural number to some mathematical object (such as a well-formed formula in some formal language). The numbering is often computed by translating every symbol of a formula Φ to a natural number, then combining the codes to create a unique Gödel number $G(\Phi)$. For example, with the assignments $t(\text{‘}\forall\text{’}) = 1, t(\text{‘}x\text{’}) = 2, t(\text{‘}.\text{’}) = 3$, and $t(\text{‘}=\text{’}) = 4$, the Gödel number of the formula $\forall x.\ x = x$ with a particular combination function could be $G(\text{‘}\forall x.\ x = x\text{’}) = 272794772250$.

   a) Is the Gödel numbering of register machines described in the notes a bijection, an injection, a surjection, a total function, a partial function, or a relation? Justify your answer.

   b) In the example of first-order logic above, is a particular Gödel numbering a bijection, an injection, a surjection, a total function, a partial function, or a relation? Justify your answer.

   c) Suggest one or more ways of combining the symbol codes of a formula Φ to generate a *unique* Gödel number for Φ. Demonstrate your methods on the formula $\Phi = \text{‘}\forall x. x = x\text{’}$ used above.

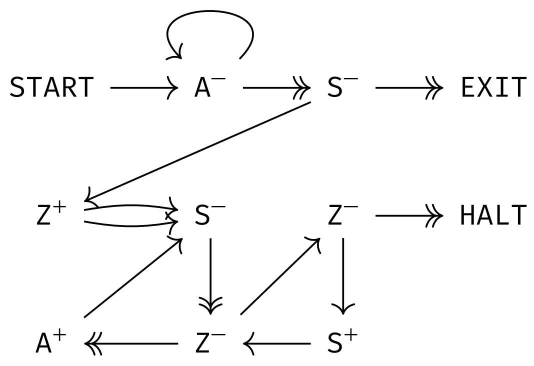
2. Let $\varphi_e \in \mathbb{N} \rightharpoonup \mathbb{N}$ denote the unary partial function from numbers to numbers computed by the register machine with code e. Show that for any given register machine computable unary partial function $f \in \mathbb{N} \rightharpoonup \mathbb{N}$, there are infinitely many numbers e such that $\varphi_e = f$. Two partial functions are equal if they are equal as sets of ordered pairs; equivalently, for all numbers $x \in \mathbb{N}$, $\varphi_e(n)$ is defined if and only if $f(x)$ is, and in that case they are equal numbers.

4. Universal register machine

1. What is the aim of the universal register machine U? How does it work? Annotate the diagram of the register machine with its major components, explaining what they accomplish in the bigger context of the operation of U.

2. Consider the list of register machine instructions whose graphical representation is shown below. Assuming that register Z holds 0 initially, describe what happens when the code is executed (both in terms of the effect on registers A and S and whether the code halts by

jumping to the label `EXIT` or `HALT`).



$$\begin{tikzcd}
\mathtt{START} \arrow[r] & \mathtt{A}^- \arrow[loop above] \arrow[r, two heads] & \mathtt{S}^- \arrow[r, two heads] \arrow[dll] & \mathtt{EXIT} \\
\mathtt{Z}^+ \arrow[r, bend left] \arrow[r, bend right, two heads] & \mathtt{S}^- \arrow[d, two heads] & \mathtt{Z}^- \arrow[r, two heads] \arrow[d] & \mathtt{HALT} \\
\mathtt{A}^+ \arrow[ur] & \mathtt{Z}^- \arrow[l, two heads] \arrow[ur] & \mathtt{S}^+ \arrow[l] &
\end{tikzcd}$$

Optional exercise

Write a register machine interpreter in a programming language you prefer (a functional language such as ML or Haskell is recommended). Implement a library of RM building blocks such as the ones appearing in the universal register machine or your answer for Ex. 2.1. You may try implementing the RM U as well, but don't worry if you run into resource constraints. The format of input and output is up to you but the RM representation and computation must conform to the theoretical definition.

5. The Halting Problem and undecidability

1. Show that decidable sets are closed under union, intersection, and complementation. Do all of these closure properties hold for undecidable languages?

2. Suppose S_1 and S_2 are subsets of $\mathbb{N}$. Suppose $r \in \mathbb{N} \to \mathbb{N}$ is a register machine computable function satisfying: for all n in $\mathbb{N}$, n is an element of S_1 if and only if $r(n)$ is an element of S_2. Show that S_1 is register machine decidable if S_2 is. Is the converse, inverse, or contrapositive of this statement true?

3. Show that the set E of codes $\langle e, e' \rangle$ of pairs of numbers satisfying $\varphi_e = \varphi_{e'}$ is undecidable.

4. Show that there is a register machine computable partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ such that both sets $\{ n \in \mathbb{N} \mid f(n){\downarrow} \}$ and $\{ y \in \mathbb{N} \mid \exists n \in \mathbb{N}.\ f(n) = y \}$ are register machine undecidable.

6. Turing machines

1. Compare and contrast register machines with Turing machines: how do they keep track of state, how are programs represented, what form do machine configurations and computations take?

2. Familiarise yourself with the Chomsky hierarchy and explain the connection between regular expressions and Turing machines.

7. Notions of computability

1. Before the formal development of the field of computation theory, mathematicians often used the term *effectively computable* to describe functions that can – in principle – be computed using mechanical, pen-and-paper methods.

   a) How was the notion of effective computability formalised by Church and Turing, and generalised to other models of computation?
   b) Suppose we invented a new model of computation. How can we establish that it is as "powerful" as mechanical methods? Make sure to formally explain what "power" means in this case.
   c) Can our new model be even more powerful?
2. Briefly describe of three Turing-complete models of computation not covered in the course.

8. Partial recursive functions

1. Show that the following functions are all primitive recursive. Make sure to give the final form of the function as a composition of primitive functions and projection.
   a) Truncated subtraction function, $minus: \mathbb{N}^2 \to \mathbb{N}$, where
   $$minus(x,y) \triangleq \begin{cases} x - y & \text{if } y \leq x \\ 0 & \text{if } y > x \end{cases}$$
   b) Exponentiation, $exp: \mathbb{N}^2 \to \mathbb{N}$, where $exp(x,y) = x^y$.
   c) Conditional branch on zero, $ifzero: \mathbb{N}^3 \to \mathbb{N}$, where
   $$ifzero(x,y,z) \triangleq \begin{cases} y & \text{if } x = 0 \\ z & \text{if } x > 0 \end{cases}$$
   d) Bounded summation: if $f : \mathbb{N}^{n+1} \to \mathbb{N}$ is primitive recursive, then so is $g : \mathbb{N}^{n+1} \to \mathbb{N}$ where where
   $$g(\vec{x},x) \triangleq \begin{cases} 0 & \text{if } x = 0 \\ f(\vec{x},0) & \text{if } x = 1 \\ f(\vec{x},0) + \cdots + f(\vec{x},x-1) & \text{if } x > 1 \end{cases}$$
2. Explain the motivation and intuition behind *minimisation*. How does it extend the set of functions computable using primitive recursion? Give three examples of computable partial functions that are not definable using primitive recursion, justifying your answer in each case.
3. Use minimisation to show that the following functions are partial recursive:
   a) the binary maximum function $max: \mathbb{N}^2 \to \mathbb{N}$.
   b) the integer square root function $sqrt: \mathbb{N} \rightharpoonup \mathbb{N}$ which is only defined if its argument is a perfect square.

Optional exercises

1. Write a Turing machine simulator in a programming language you prefer (a functional language such as ML or Haskell is recommended). Implement the machine described on Slide 64.

2. For the example Turing machine given on Slide 64, give the register machine program implementing $(S,T,D) := \delta(S,T)$, as described on Slide 70.

3. Recall the definition of Ackermann's function *ack*. Sketch how to build a register machine M that computes $ack(x_1,x_2)$ in R_0 when started with x_1 in R_1 and x_2 in R_2 and all other registers zero. (E9)

   Hint: Call a finite list $L = [(x_1,y_1,z_1),(x_2,y_2,z_2),\ldots]$ of triples of numbers *suitable* if it satisfies

   a) if $(0,y,z) \in L$, then $z = y+1$

   b) if $(x+1,0,z) \in L$, then $(x,1,z) \in L$

   c) if $(x+1,y+1,z) \in L$, then there is some u with $(x+1,y,u) \in L$ and $(x,u,z) \in L$.

   The idea is that if $(x,y,z) \in L$ and L is suitable then $z = ack(x,y)$ and L contains all the triples $(x',y',ack(x,y'))$ needed to calculate $ack(x,y)$. Show how to code lists of triples of numbers as numbers in such a way that we can (in principle, no need to do it explicitly!) build a register machine that recognises whether or not a number is the code for a suitable list of triples. Show how to use that machine to build a machine computing $ack(x,y)$ by searching for the code of a suitable list containing a triple with x and y in its first two components.

9. Lambda calculus

1. Given a set $\mathcal{V} = \{x,y,\ldots\}$ of variables, define the set $\mathcal{T}$ of λ-terms

   a) as an inductively defined set (see IA Formal Languages course).

   b) using Backus–Naur form (see IB Semantics course).

   c) using a recursive set comprehension (see Lecture 7 of the IB Logic course).

2. a) Simplify the following λ-terms (as much as possible, but without evaluating them) using the notational conventions described on Slide 105:

   $$(\lambda x.\,((ux)y)) \qquad (((\lambda u.\,(\lambda v.\,(vu)u))z)y) \qquad ((((\lambda x.\,(\lambda y.\,(\lambda z.\,((xz)(yz)))))u)v)w)$$

   b) Expand the following simplified λ-terms (as much as possible) using the notational conventions described on Slide 105, inserting all parentheses and λ's:

   $$xyz(yx) \qquad \lambda u.\,u(\lambda x.\,y) \qquad \lambda xy.\,ux(yz)(\lambda v.\,vy)$$

3. Give a recursive definition of the function $len(M)$ denoting the *length* of the λ-term M given by the total number of variables in M. For example, $len(x(\lambda y.\,yux)) = 5$.

4. a) Define the *subterm relation* $M \sqsubseteq N$ by recursion on N. For example,

   $$x \sqsubseteq \lambda y.\,ux \qquad \lambda x.\,y \sqsubseteq \lambda x.\,y \qquad xy \sqsubseteq (\lambda x.\,xy)z \qquad z \sqsubseteq x(\lambda z.\,y)$$

   but $uv \not\sqsubseteq \lambda x.\,xu(vy)$.

   b) We say there is *an occurrence of M in N* if $M \sqsubseteq N$.

(i) Mark all occurrences of xy in $(xy)(\lambda x.\ xy)$.
(ii) Mark all occurrences of x in $(xy)(\lambda x.\ xy)$.
(iii) Mark all occurrences of xy in $\lambda xy.\ xy$.
(iv) Mark all occurrences of uv in $x(uv)(\lambda u.\ v(uv))uv$.
(v) Does $\lambda u.\ u$ occur in $\lambda u.\ uv$?

5. Let M be the λ-term $\lambda xy.\ x(\lambda z.\ zu)y$.

   a) What is the β-normal form of the term $N = M(\lambda vw.\ v(wb))(\lambda xy.\ yaz)$?

   b) Apply the simultaneous substitution $\sigma = [x/y, (\lambda xy.\ zy)/u]$ to M and N, and find the β-normal form of $N[\sigma]$.

   c) Give 2 terms α-equivalent to M. Give 2 other terms β-equivalent to M.

   h. We define η-equivalence as: $M =_\eta \lambda x.\ Mx$ for any λ-term M. Give a shorter and a longer term η-equivalent to M. What is the use of η-equivalence in functional programming?

6. What are some differences between the lambda calculus as defined in this course, and the functional subset of L2 from the IB Semantics course?

10. Lambda-definable functions

1. Give a complete proof of the correctness of Church addition from Slide 119. *Hint:* a formal justification of one of the steps will require mathematical induction.

$$\mathbf{Plus}\,\underline{m}\,\underline{n} =_\beta \underline{m+n}$$

2. Define the λ-terms **Times** and **Exp** representing multiplication and exponentiation of Church numerals respectively. Prove the correctness of your definitions.

$$\mathbf{Times}\,\underline{m}\,\underline{n} =_\beta \underline{m \times n} \qquad \mathbf{Exp}\,\underline{m}\,\underline{n} =_\beta \underline{m^n}$$

3. Show that the λ-term $\mathbf{Ack} \triangleq \lambda x.\ x\,T\,\mathbf{Succ}$, where $T \triangleq (\lambda f\,y.\ y f(f\,\underline{1}))$ represents Ackermann's function $ack \in \mathbb{N}^2 \to \mathbb{N}$. *Hint*: you will need to use nested induction; consider deriving a simplified form for the outer inductive case before starting the nested proof.

4. Consider the following λ-terms:

$$\mathbf{I} \triangleq \lambda x.\ x \qquad \mathbf{B} \triangleq \lambda g f x.\ f\,x\,\mathbf{I}(g(f\,x))$$

   a) Show that $\underline{n}\,\mathbf{I} =_\beta \mathbf{I}$ for every $n \in \mathbb{N}$.

   b) Assuming the fact about normal order reduction mentioned on Slide 115, show that if partial functions $f, g : \mathbb{N} \rightharpoonup \mathbb{N}$ are represented by closed λ-terms F and G respectively, then their composition $(g \circ f)(x) \triangleq g(f(x))$ is represented by $\mathbf{B}\,G\,F$. Explain how this avoids the discrepancy with partial functions mentioned in Slide 125.

5. In the following questions you may use all of the λ-definable functions presented in the notes, as well as the terms you define as part of this exercise. You should explain your answers

(possibly using some examples), but don't need to prove their correctness.

a) Give a λ-term **Not** representing Boolean negation.

b) Give λ-terms **And** and **Or** representing Boolean conjunction and disjunction.

c) Give a λ-term **Minus** representing truncated subtraction (i.e. $\mathbf{Minus}\,\underline{m}\,\underline{n} = 0$ if $m < n$).

d) Give λ-terms **Eq**, **NEq**, **LT**, **LEq**, **GT**, **GEq**, representing the numeric comparison operations $=, \neq, <, \leq, >, \geq$ respectively. You can define them in any order you find most convenient.

e) Define the λ-term **UCr** that represents the uncurrying higher-order function: if $\langle M, N \rangle$ denotes $\mathbf{Pair}\,M\,N$, then $\mathbf{UCr}\;F\;\langle M, N\rangle =_\beta F\,M\,N$.

f) Give a λ-term **MapPair** that applies a function to both elements of a pair: that is, $\mathbf{MapPair}\,F\;\langle M, N\rangle =_\beta \langle F\;M, F\;N\rangle$.

g) Give a λ-term **SqSum** which represents the function $\langle m, n\rangle \mapsto m^2 + n^2$.

6. a) Explain why Curry's **Y** combinator is needed and how it works.

b) Give a λ-term which is β-equivalent to the **Y** combinator, but only uses it's f argument once. *Hint*: see if you can exploit the symmetry of the **Y** combinator.

c) Define the λ-term **Fact** that computes the factorial of a Church numeral.

d) Define the λ-term **Fib** such that $\mathbf{Fib}\,\underline{n} =_\beta \underline{F_n}$ where F_n is the n^{th} Fibonacci number defined recursively as $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$.