

Lecture
Models of computation
(DIT311, TDA184)

Nils Anders Danielsson

2018-11-05

Can every function be implemented?

- ▶ No (given some assumptions).
- ▶ This lecture: Two proofs (sketches).

General information

See the course web page.

Comparing sets' sizes

Injections

- ▶ Definition: $f \in A \rightarrow B$ is *injective* if $\forall x, y \in A. f x = f y$ implies $x = y$.
- ▶ If there is an injection from A to B , then B is at least as “large” as A .

Surjections

- ▶ Definition: $f \in A \rightarrow B$ is *surjective* if $\forall b \in B. \exists a \in A. f a = b$.
- ▶ If there is a surjection from A to B , then there is an injection from B to A (assuming the axiom of choice).
- ▶ Thus, if there is a surjection from A to B , then A is at least as “large” as B .

Left/right inverses

For functions $f \in A \rightarrow B$, $g \in B \rightarrow A$:

- ▶ Definition: g is a *left inverse* of f if $\forall a \in A. g(f a) = a$.
- ▶ Definition: g is a *right inverse* of f if $\forall b \in B. f(g b) = b$.
- ▶ If f has a left inverse, then it is injective.
- ▶ If f has a right inverse, then it is surjective.

Bijections

- ▶ Definition: $f \in A \rightarrow B$ is *bijective* if it is both injective and surjective.
- ▶ A function is bijective iff it has a left and right inverse.
- ▶ If there is a bijection from A to B , then A and B have the same “size”.

Quiz

Which of the following functions are injective? Surjective?

- ▶ $f \in \mathbb{N} \rightarrow \mathbb{N}, f n = n + 1.$
- ▶ $g \in \mathbb{Z} \rightarrow \mathbb{Z}, g i = i + 1.$
- ▶ $h \in \mathbb{N} \rightarrow Bool, h n = \begin{cases} \text{true,} & \text{if } n \text{ is even,} \\ \text{false,} & \text{otherwise.} \end{cases}$

Respond at <https://pingo.upb.de/>,
using a code that I provide.

Countable,
uncountable

Countable sets

- ▶ A is *countable* if there is an injection from A to \mathbb{N} .
- ▶ If there is no such injection, then A is *uncountable*.
- ▶ A is *countably infinite* if there is a bijection from A to \mathbb{N} .

Countable sets

- ▶ There is an injection from A to B iff $A = \emptyset$ or there is a surjection from B to A (assuming the axiom of choice).
- ▶ Thus A is countable iff $A = \emptyset$ or there is a surjection from \mathbb{N} to A .

Quiz

The set of finite strings of characters is infinite. Is it countable?

- ▶ Yes.
- ▶ No.

If A is countable, then $List A$ is countable.

Proof sketch:

- ▶ We are given an injection $f \in A \rightarrow \mathbb{N}$.
- ▶ Define $g \in List A \rightarrow \mathbb{N}$ by

$$g(x_1, x_2, \dots, x_n) = 2^{1+f x_1} 3^{1+f x_2} \dots p_n^{1+f x_n},$$

where p_n is the n -th prime number.

- ▶ By the fundamental theorem of arithmetic and the injectivity of f we get that g is injective.

Uncountable sets

- ▶ Is every set countable?
- ▶ No.
- ▶ *Diagonalisation* can be used to show that certain sets are uncountable.

$\mathbb{N} \rightarrow \mathbb{N}$ is uncountable

Proof (using the axiom of choice):

- ▶ Assume that $\mathbb{N} \rightarrow \mathbb{N}$ is countable.
- ▶ The set is non-empty, so we get a surjection $f \in \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$.
- ▶ Define $g \in \mathbb{N} \rightarrow \mathbb{N}$ by $g n = f n n + 1$.
- ▶ By surjectivity we get that $g = f i$ for some i .
- ▶ Thus $f i i = g i = f i i + 1$, which is impossible.

Diagonalisation

The function g differs from every function enumerated by f on the “diagonal”:

	0	1	2	3	...
f_0	+1				
f_1		+1			
f_2			+1		
f_3				+1	
\vdots					

Not every function is computable

Proof sketch (classical):

- ▶ The set of programs P of a typical programming language is countable and nonempty, thus there is a surjection from \mathbb{N} to P .
- ▶ There is no surjection from \mathbb{N} to $\mathbb{N} \rightarrow \mathbb{N}$.
- ▶ Thus there is no surjection from P to $\mathbb{N} \rightarrow \mathbb{N}$ (the composition of two surjections is surjective).
- ▶ Thus, however you give semantics to programs, it is not the case that every function is the semantics of some program.

Quiz

If we define $g_n = f_n(2n) + 1$, does the diagonalisation argument still work? [BN]

	0	1	2	3	4	5	6	...
f_0	+1							
f_1			+1					
f_2					+1			
f_3							+1	
\vdots								

The halting problem

Uncomputable functions

- ▶ Can we find an explicit example of a function that cannot be computed?
- ▶ What does “can be computed” mean?
- ▶ Let us restrict attention to a “typical” programming language.
- ▶ In that case the answer is yes.
- ▶ A standard example is the halting problem.

The halting problem

Given the source code of a program and its input, determine whether the program will halt when run with the given input.

The halting problem is not computable

Proof sketch (with hidden assumptions):

- ▶ Assume that the halting problem is implemented by *halts*.
- ▶ Define $p\ x = \mathbf{if\ }halts\ x\ x\ \mathbf{then\ }loop\ \mathbf{else\ }skip$.
- ▶ Consider the application $p\ \ulcorner p \urcorner$, where $\ulcorner p \urcorner$ is the source code of p .
- ▶ The result of $halts\ \ulcorner p \urcorner\ \ulcorner p \urcorner$ must be true or false.

Quiz

Can the result of $halts \ulcorner p \urcorner \ulcorner p \urcorner$ be true?

- ▶ Yes.
- ▶ No.

The halting problem is not computable

Proof sketch (continued):

- ▶ If $\text{halts } \ulcorner p \urcorner \ulcorner p \urcorner = \text{true}$, then:
 - ▶ $p \ulcorner p \urcorner$ terminates (specification of *halts*).
 - ▶ $p \ulcorner p \urcorner = \text{loop}$, which does not terminate.
- ▶ If $\text{halts } \ulcorner p \urcorner \ulcorner p \urcorner = \text{false}$, then:
 - ▶ $p \ulcorner p \urcorner$ does not terminate.
 - ▶ $p \ulcorner p \urcorner = \text{skip}$, which does terminate.
- ▶ Either way, we get a contradiction.

Models of computation

Models of computation

- ▶ The proof is based on some assumptions.
- ▶ For instance, the programming language allows us to define **if–then–else** and *loop*, with the intended semantics.
- ▶ Later in the course we will be more precise.
- ▶ To make it easier to study questions of computability we will use idealised models of computation.

Models of computation

One model:

- ▶ The primitive recursive functions.
- ▶ Functional in character.
- ▶ All programs terminate.

Models of computation

Another model:

- ▶ A lambda calculus with pattern matching called χ .
- ▶ Functional in character.
- ▶ Some programs do not terminate.

Models of computation

Yet another model:

- ▶ Turing machines.
- ▶ Imperative in character.
- ▶ Some programs do not terminate.

The Church-Turing thesis

Models of computation

- ▶ How are these models related?
- ▶ Can one say anything about programming in general?
- ▶ It has been noted that many models of computation are, in some sense, equivalent:
 - ▶ Turing machines.
 - ▶ The (untyped) λ -calculus.
 - ▶ The recursive functions.
 - ▶ ...

The Church-Turing thesis

Every effectively calculable function on the positive integers can be computed using a Turing machine.

The Church-Turing thesis

Every effectively calculable function on the positive integers can be computed using a Turing machine.

- ▶ This is one variant of the thesis.
- ▶ We will define “can be computed using a Turing machine” more precisely later.

Effectively calculable

“Effectively calculable” means *roughly* that the function can be computed by a human being

- ▶ following exact instructions, with a finite description,
- ▶ in finite (but perhaps very long) time,
- ▶ using an unlimited amount of pencil and paper,
- ▶ and no ingenuity.

(See Copeland.)

The Church-Turing thesis

- ▶ The thesis is a conjecture.
- ▶ “Effectively calculable” is an intuitive notion, not a formal definition.
- ▶ However, the thesis is widely believed to be true.

Turing-complete

A programming language is *Turing-complete* if every Turing machine can be simulated using a program written in this language.

Turing-complete

A programming language is *Turing-complete* if every Turing machine can be simulated using a program written in this language.

- ▶ This is one variant of the definition.
- ▶ We have not specified what it means to simulate a Turing machine.

Only
terminating
programs?

Only terminating programs?

- ▶ Every primitive recursive function terminates.
- ▶ Easy to solve the halting problem!
- ▶ Can we have a model of computation that includes exactly those functions on the natural numbers that can be implemented using Turing machines that always halt?

Only terminating programs?

- ▶ Every primitive recursive function terminates.
- ▶ Easy to solve the halting problem!
- ▶ Can we have a model of computation that includes exactly those functions on the natural numbers that can be implemented using Turing machines that always halt?
- ▶ No (given some assumptions).

Only terminating programs?

The following assumptions are contradictory:

- ▶ The set of valid programs $Prog \subseteq \mathbb{N}$.
- ▶ For every computable function $f \in \mathbb{N} \rightarrow \mathbb{N}$ there is a program $\ulcorner f \urcorner \in Prog$.
- ▶ There is a computable function $eval \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ satisfying $eval \ulcorner f \urcorner n = f n$.

(See Brown and Palsberg.)

Only terminating programs?

Proof sketch:

- ▶ Define the computable function $f \in \mathbb{N} \rightarrow \mathbb{N}$ by
 $f\ n = \text{eval}\ n\ n + 1$.
- ▶ We get

$$\begin{aligned} & f\ \ulcorner f \urcorner \\ &= \text{eval}\ \ulcorner f \urcorner\ \ulcorner f \urcorner + 1 \\ &= f\ \ulcorner f \urcorner + 1, \end{aligned}$$

which is impossible.

A variant of the previous argument

Assumptions:

- ▶ Programs: $Prog$.
- ▶ Computable semantics:

$$\llbracket - \rrbracket \in Prog \times \mathbb{N} \rightarrow \mathbb{N}$$

- ▶ A coding function:

$$code \in Prog \rightarrow \mathbb{N}$$

- ▶ A computable left inverse of $code$:

$$decode \in \mathbb{N} \rightarrow Prog$$

A variant of the previous argument

Goal: Prove that the following statement is false:

$$\forall g \in \mathbb{N} \rightarrow \mathbb{N}. g \text{ is computable} \Rightarrow \\ \exists \underline{g} \in \text{Prog}. \forall n \in \mathbb{N}. \llbracket (\underline{g}, n) \rrbracket = g n$$

A variant of the previous argument

Goal: Prove that the following statement is true:

$$\exists g \in \mathbb{N} \rightarrow \mathbb{N}. g \text{ is computable} \wedge \\ (\forall \underline{g} \in Prog. (\forall n \in \mathbb{N}. \llbracket (\underline{g}, n) \rrbracket = g \ n) \rightarrow \perp)$$

A variant of the previous argument

- ▶ Define $g \in \mathbb{N} \rightarrow \mathbb{N}$ by

$$g \ n = \llbracket (\text{decode } n, n) \rrbracket + 1.$$

Note that g is computable.

- ▶ Assume that we have $\underline{g} \in \text{Prog}$, with

$$\forall n \in \mathbb{N}. \llbracket (\underline{g}, n) \rrbracket = g \ n.$$

- ▶ We get a contradiction:

$$\begin{aligned} g(\text{code } \underline{g}) &= \\ \llbracket (\text{decode}(\text{code } \underline{g}), \text{code } \underline{g}) \rrbracket + 1 &= \\ \llbracket (\underline{g}, \text{code } \underline{g}) \rrbracket + 1 &= \\ g(\text{code } \underline{g}) + 1 & \end{aligned}$$

Summary

- ▶ Injections, surjections, bijections.
- ▶ Countable and uncountable sets.
- ▶ Diagonalisation.
- ▶ The halting problem.
- ▶ Models of computation.
- ▶ The Church-Turing thesis.

Summary

- ▶ Injections, surjections, bijections.
- ▶ Countable and uncountable sets.
- ▶ Diagonalisation.
- ▶ The halting problem.
- ▶ Models of computation.
- ▶ The Church-Turing thesis.

Please try to solve the recommended exercises before coming to the tutorial on Wednesday.