Chapter 5

Computability Theory

Bahar Aameri
Department of Computer Science
University of Toronto

Mar 27, 2015

Announcements

- **Additional Review Session (Bahar)**:
  - **Tues**, Mar 31, **2-3:45pm** in **MP203**.
- **Additional Instructor Office Hours (Bahar)**:
  - **Mar 30**, **Apr 01**, **Apr 08**, Time and location TBA.
  - If you cannot make any of the office hours email me to schedule for another time.
- **Additional TA Office Hours**: TBA
- **Final Exam**:
  - Chapter 1: Only Section 1.5 (Problem-solving techniques in Section 1.3 are useful in doing proofs. See Tutorial 5 and its solutions)
  - Chapter 2
  - Chapter 3
  - Chapter 4: Excluding Sections 4.2 and 4.3

Today's Topics

- **Computability of Functions**
- **Proving Uncomputability: The Halting Problem**
- **Proving Uncomputability: Reduction**
- **Countability of Sets**

Chapter 5

Computability Theory

Computability of Functions

Computability of Functions

Computable Functions

A function $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$ is **computable** if there exists a **computer function** (or **algorithm**) $\mathbf{F}$ such that for all $a \in A$, $\mathbf{F}(a)$ returns the value of $\mathbf{f}(a)$.

Examples of Computable Functions

- $f(x) = x^2$
- $f(x) = log(x)$

Chapter 5

Computability Theory

Proving Uncomputability: The Halting Problem

The Halting Problem

The Halt Function

- Is halt(f,i) computable?

$$halt(f,i) = \begin{cases} True & \text{if computer function } f(i) \text{ eventually halts,} \\ False & \text{otherwise} \end{cases}$$

- Is there an implementation for HALT(f,i)?

```
def HALT(f,i):
    """Return True iff f(i) eventually halts."""
    """Return False otherwise."""
```

Halting Function

Claim: `HALT(f,i)` is NOT computable

Assume `HALT` exists. # to derive contradiction

Then consider the following function.

```
def CONFUSED(f):
    def HALT(f,i):
        ...copy/paste the full code for HALT here...

        if HALT(f,f):                # line 1
            while True:  pass        # line 2
        else:
            return False             # line 3
```

Case 1: Assume `CONFUSED(CONFUSED)` halts.

Then `HALT(CONFUSED, CONFUSED)` returns `True`. # def of `HALT`

Then `CONFUSED(CONFUSED)` goes into an infinite loop. # on line 2

Then `CONFUSED(CONFUSED)` does not halt.

Contradiction!

Case 2: Assume `CONFUSED(CONFUSED)` does not halt.

Then `HALT(CONFUSED, CONFUSED)` returns `False`. # def. of `HALT`

Then `CONFUSED(CONFUSED)` returns `False` # on line 3.

Then `CONFUSED(CONFUSED)` halts.

Contradiction!

Contradiction! # both cases lead to contradiction

Then `HALT` does not exist! # assuming the negation leads to contradiction

Chapter 5

Computability Theory

Proving Uncomputability: Reduction

Proving Uncomputability by Reduction

Prove that a function `F` is uncomputable

1. Assume that `F` is computable.
2. Use `F` to implement `HALT`. (**Reducing** `F` to `HALT`)
3. Derive a **contradiction** as `HALT` is not computable.

Prove that a function `alltrue` is NOT computable

$$alltrue(P) = \begin{cases} True & \text{if } P(x) \text{ returns True for every value of } x, \\ False & \text{if } P(x) \text{ returns False or does not halt for at least one } x \end{cases}$$

1. Assume that `alltrue` is computable.
2. **Reduce** `alltrue` to `HALT` (Use `alltrue` to implement `HALT`).
3. Derive a **contradiction** as `HALT` is not computable.

Prove that a function `alltrue` is NOT computable

Assume that `alltrue` is computable. # to derive contradiction
Then the following function is computable # All parts of `H` are computable

```
def H(f,i):
    def RED(y):              # line 1
        f(i)                 # line 2
        return True          # line 3
    return alltrue(RED)      # line 4
```

Then, by construction of `RED(y)`

1. if `f(i)` **halts**, then `RED(y)` returns **True** for every `y`;
2. if `f(i)` **does not halt**, then `RED(y)` **does not halt** for any `y`.

Then, for every `f` and `i`,

`H(f,i)` returns **True** if `alltrue(RED)` returns True. # line 4 in `H`
if `RED(y)` returns True for every y. # by def of `alltrue(RED)`
if `f(i)` **halts**; # by #1 above

`H(f,i)` returns **False** if `alltrue(RED)` returns False # line 4 in `H`
if `RED(y)` fails to halt for every y # by def of `alltrue(RED)`
if `f(i)` **does not halt**. # by #2 above

Then `H(f,i)` computes `HALT(f,i)`. # contradiction! since `HALT` is not computable
Then `alltrue` is not computable. # Assuming the negation lead to contradiction

Chapter 5

Computability Theory

Countability of Sets

Countability

Countable Sets: Informal Definition

A set S is **countable** iff each element of S can be mapped to a (natural) number.

Examples

- **Countable Sets**: All finite sets, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$.
- **Uncountable Sets**: $\mathbb{R}$, the set of all functions from $\mathbb{R}$ to $\mathbb{R}$.

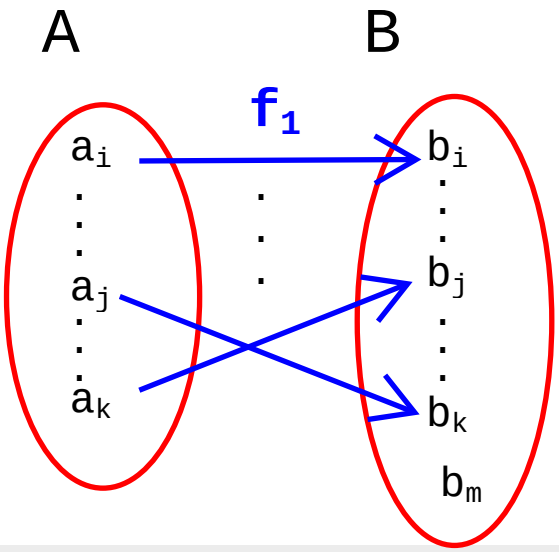
Countability

One-to-One and Onto Functions

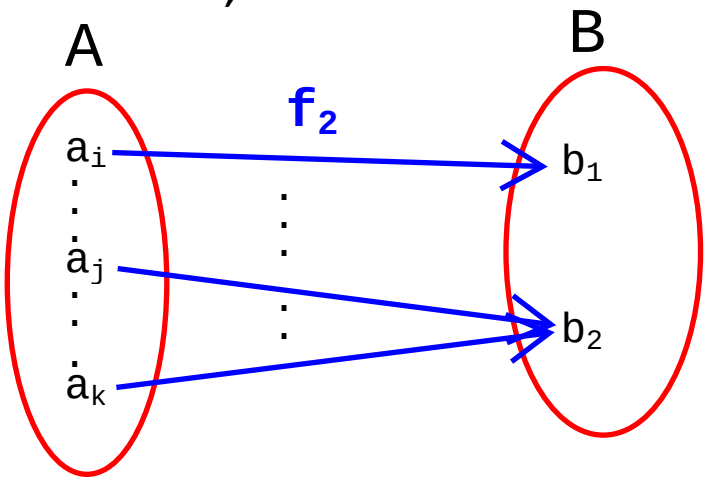
Suppose $f : A \to B$ is a function that associates an element $f(a) \in B$ to each element $a \in A$.

- f is **one-to-one** if f maps each element of A to a distinct value in B
  $\forall a_1 \in A, \forall a_2 \in A, f(a_1) = f(a_2) \implies a_1 = a_2$;
- f is **onto** if every element in B is mapped to at least one element of A
  $\forall b \in B, \exists a \in A, f(a) = b$.

One-to-one, but not onto:



Onto, but not one-to-one:



Countability

Countable Sets: Formal Definition

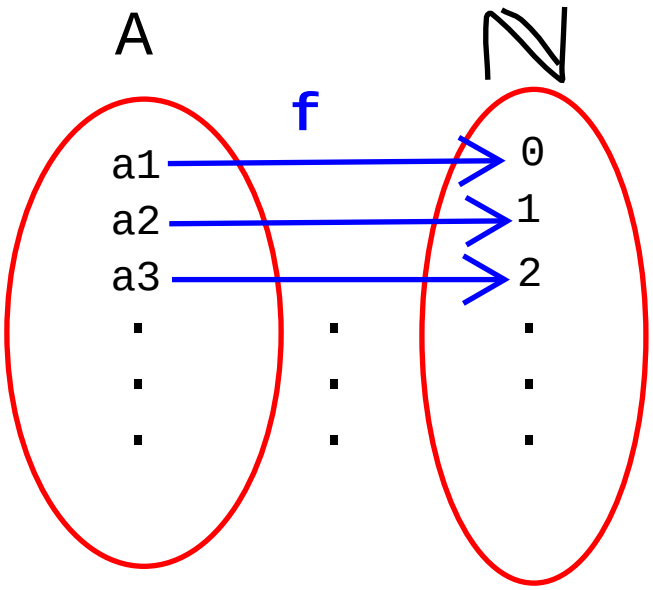
Set A is **countable** iff

1. there is a function $\mathbf{f} : \mathbf{A} \to \mathbb{N}$ that is **one-to-one**.

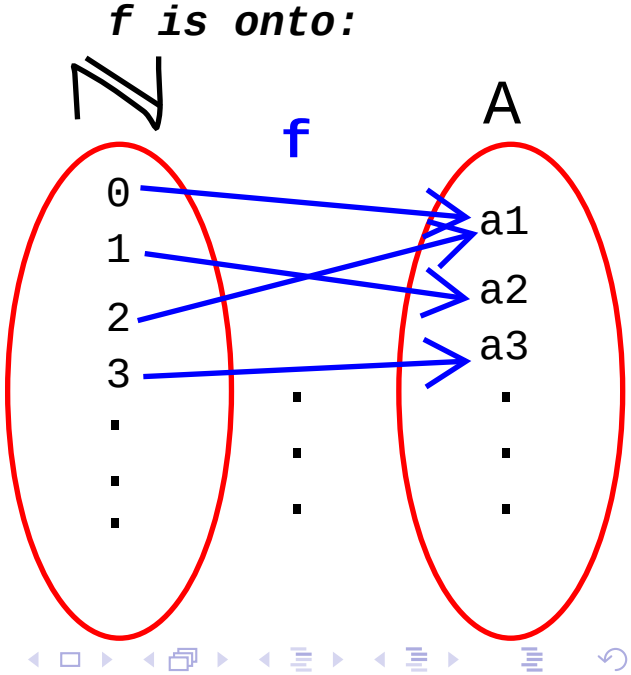
   or equivalently,

2. there is a function $\mathbf{f} : \mathbb{N} \to \mathbf{A}$ that is **onto**.

f is one-to-one:



f is onto:



Countability of $\mathbb{Z}$

Claim: $\mathbb{Z}$ is countable

Let $f(n) = \begin{cases} n/2 & \text{if } n \text{ is even,} \\ (1-n)/2 & \text{if } n \text{ is odd.} \end{cases}$

Then $f : \mathbb{N} \to \mathbb{Z}$ is a function. # needs proof, but we skip it

Assume $x \in \mathbb{Z}$. # x is a typical integer

Then $x < 0$ or $x \geq 0$.

Case 1: Assume $x < 0$.

Let $n' = 1 - 2x$.

Then $x < 0$, so $-2x > 0$, so $n' > 0$ so $n' \in \mathbb{N}$.

Also, $n' = 2(-x) + 1$, so n' is odd.

Then $f(n') = (1 - n')/2 = (1 - (1 - 2x))/2 = 2x/2 = x$.

Hence $\exists n \in \mathbb{N}, f(n) = x$.

Case 2: Assume $x \geq 0$.

Let $n' = 2x$.

Then $x \geq 0$, so $n' \geq 0$ so $n' \in \mathbb{N}$.

Also, $n' = 2x$ is even.

Then $f(n') = n'/2 = 2x/2 = x$.

Hence $\exists n \in \mathbb{N}, f(n) = x$.

In all cases, $\exists n \in \mathbb{N}, f(n) = x$.

$\forall x \in \mathbb{Z}, \exists n \in \mathbb{N}, f(n) = x$. # intro $\forall$

Then f is onto. # by definition of onto functions

Then $\mathbb{Z}$ is countable. # there is a function $f : \mathbb{N} \to \mathbb{Z}$ that is onto