CS 5002: Discrete Structures **Fall 2018**

Lecture 2: September 13, 2018

Instructors: Adrienne Slaughter, Tamara Bonaci

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

2.1 Overview

1. Review: logic and number representation
2. Representing negative numbers
3. Logic in Computers and Binary Strings
4. Variables and Functions
5. Some Important Integer Functions

2.2 Introduction

In today's lecture, we will quickly review the most important concepts we discussed last time - logical propositions, operators, and quantifiers, as well as the notion of logical eqiuvalence. We will then show how those might apply to bit strings.

We will then refresh our memory about number representation, and talk about possible ways to represent negative numbers.

Our big topic for today, however, are variables and functions. We will introduce those concepts, and see why and how do functions matter to computer scientists. We will finish by talking about special type of functions - integer functions, and we will consider some important examples.

2.3 Review

2.3.1 Logic: Propositions, Operators, Logical Equivalence and Quantifiers

In the last lecture, we defined a **proposition** as a statement that is true or false, but not both, and showed that such a proposition has a **truth value**. The ***truth value*** of a proposition is true (**T**) if it is a true proposition and false (**F**) if it is false.

We then introduced several logic operators:

- **Negation,** $\neg$ – the proposition $\neg p$ is false when p is true, and true when p is false.
- **Conjunction,** $\wedge$ – the proposition p and q is $p \wedge q$ and is *true* when p and q are true, and false otherwise.
- **Disjunction or inclusive OR,** $\vee$ – the proposition p or q (written by $p \vee q$) is *false* when p and q are false, and true otherwise.

- **Exclusive OR, or XOR,** $\oplus$ – the proposition p xor q (written by $p \oplus q$) is *true* when p is true OR q is true, but not when p and q are either both true or both false.
- **Implication or conditional,** $\rightarrow$ – the proposition $p \rightarrow q$ is false when p is true and q is false, and true otherwise.
- **Biconditional,** $\leftrightarrow$ – the proposition $p \leftrightarrow q$ is true when p is true and q is true, or p is false and q is false, false otherwise.

We then showed that we can create **compound propositions** by grouping and ordering propositions. We can then evaluate ***logical equivalence*** of two or more such (compound) propositions by comparing the truth tables of each of them.

We next talked about predicates and quantifiers, and introduced the **universe of discourse**, and two special quantifiers:

- **Universal quantifier,** $\forall$ – for every value in the *universe of discourse*, some predicate P is true.
- **Existential quantifier,** $\exists$ – There exists a value in the universe of discourse such that some predicate P is true.

2.3.2 Number Representations

We know that we generally use *decimal* notation to express integers. After the last lecture, however, we know that we don't always have to use base 10. We can actually use any base, such as, for example 5, and in the last lecture, we showed general rules to represent some number n into some arbitrary base b.

Theorem 2.1 *If b is a positive integer greater than 1, and n is a positive integer, it can be expressed in the form:*

$$n_{10} = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0$$

where k is a nonnegative integer, the number of digits in n

Some popular bases that we will be dealing with in computer sciences in general are:

- **Binary base** – base $b = 2$
- **Octal base** – base $b = 8$
- **Hexadecimal base** – base $b = 16$

To construct the base b expansion of integer n, first divide n by b to obtain a quotient and remainder. That is:

$$n = bq_0 + a_0, 0 \leq a_0 < b$$

The remainder (a_0) is the rightmost (least significant) digit in the base b expansion on n. Next, divide q_0 by b to obtain

$$q_0 = bq_1 + a_1, 0 \leq a_1 < b$$

a_1 is the second digit from the right in the base b expansion of n. Continue this until you obtain a quotient equal to 0.

We then briefly talked on the number of possible values that we can represent in some base. We observed that for some base b and digit length k, the maximum number of values that can be represented is b^k.

2.4 Representing Negative Numbers

So far in the course, we have only talked about positive numbers. In real life, however, numbers can, and and will be negative, and in math we have a simple way to deal with that - we just put a negative sign in front of a number.

The question, however, is: what do we do in machine world, to represent negative numbers. It turns out there are several approaches to that. For example, we can designate a particular bit, normally the left-most to indicate the sign. The problem with this approach, however, is that the procedures for adding the resulting numbers are somewhat complicated, and also the representation of zero si not unique.

One popular approach, that resolves these two issues, is referred to as **two's complement**, and in this approach, the negative sign is encoded within the number representation itself.

Definition 2.2 *Given a positive integer a, the* **two's complement** *of a relative to the fixed bit length n is the n-bit binary representation of $2^n - a$.*

Two's complement - explanation: we will cover the math behind why this works in several weeks, but two's complement relies on the fact that **under modular arithmethic**, given some modulus 2^n, every negative number has a "positive counterpart", computed as $2^n - a$. In the modular arithmetic, you can use this "positive counterpart" every time you would use the negative number, and the result will be correct.

This is convenient, because with two's complement, the representation of zero is unique, we can easily add and subtract positive and negative numbers, and we didn't reduce the size of the set of positive and negative numbers that we can represent.

Two's complement - Computation: It turns out that there exists a convenient way to compute two's complements that involves less arithmeitc then the direct application of the definition. For example, let's consider an 8-bit representation. That representation is based on three facts:

1. $2^8 - a = [(2^8 - 1) - a] + 1$
2. The binary representation of $2^8 - 1$ is 11111111_2
3. Subtracting an 8-bit binary number a from 11111111_2 is equivalent to just switching all 0's to 1's, and vice versa (flipping the bits). The resulting number is called the **one's complement** of the given number.

Example 1: Let's find an 8-bit two's complement of $a = 27$. To do so, let's represent $a = 27$ in the binary base as $a = 27_{10} = 00011011_2$.

We can write:

$$\begin{array}{ll} & 11111111 = 2^8 - 1 \\ - & 00011011 = 27_2 \\ & 11100100 = (2^8 - 1) - 27 \\ + & 00000001 = 1 \\ \hline \end{array}$$

$$11100101 = 2^8 - 27 \tag{2.1}$$

So, in general, to find an n-bit complement of a positive integer a:

1. Write an n-bit representation of a.
2. Flip the bits (that is, switch all 1's to 0's, and all 0's to 1's).
3. Add 1 to binary representation.

2.4.1 Logic in Computers

Definition 2.3 *A* **Boolean variable** *is a variable that can take on only two values, either true or false, and such a variable can be represented using a bit.*

Computers represent information using **bits**, where a bit is defined as a symbol with two possible values, 0 (zero) and 1 (one). Luckily for us, a bit can also be used to represent a truth value, and typically, we use a 1 bit to represent true, and a 0 bit to represent false.

This is convenient because it allows us to establish the correspondence between computer **bit operations**, as follows.

p	q	NOT p	p AND q	p OR q	p XOR q
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0

An interesting question one can now ask is – what if we have more than one bit of information to represent?

Once again – not a problem for us. We will just use a **bit string (binary string)**, defined as follows.

Definition 2.4 *A* **bit string** *is a sequence of zero or more bits, and its length is equal to the number of bits in this string.*

We can now use bit strings, and extend binary operations to bit strings as follows.

Bitwise OR, bitwise AND, and **BITWISE XOR** of two strings of the same length are the resulting bit strings, of the same length, that have as their bits the OR, AND or XOR of the corresponding bits in the starting strings.

Let's take a look at an example.

Example 2: Consider bit strings 0110 1101 1000 and 1100 0111 0110. Find the bitwise OR, bitwise AND and bitwise XOR of these strings.

$$\begin{array}{ll} 011011011000 & \\ 110001110110 & \\ \hline 111011111110 & (\text{bitwise } OR) \\ 000001010000 & (\text{bitwise } AND) \\ 111011101110 & (\text{bitwise } XOR) \end{array}$$

2.5 Variables and Functions

Mathematics is a language. *Josiah Willard Gibbs(1839–1903)*

In this section, we are staring our deep dive into discrete math, and we will do so by asking how does one express mathematical statements?

It turns out there exist several different ways to express mathematical statements, but three of the most important mathematical statements are:

- **Universal statements,** stating the property that is true for all elements in some set. (For example: *All students in this class are ALIGN students.*)
- **Conditional statement,** stating that if one thing is true, then it has to be the case that some other thing is also true. (For example *If a student is required to take CS 5002, then tht student is an ALIGN student.*)
- **Existential statement,** which states that, given some property, there exists at least one element of the set for which the given property is true. (For example: *It will be sunny at least one day this week.*)

Such mathematical statements can be expressed in a variety of ways, but we are interested in a way that is efficient and practical for computer systems. As it turns out, one such a way relies on variables and functions.

2.5.1 Variables

Definition 2.5 *A* **variable** *is any characteristic, number or quantity whose value:*

- Is arbitrary, or
- Is not fully specified, or
- Is unknow, or
- Changes over time

Let's briefly analyze this definition of a variable a bit more. Intuitively, we can think of a variable as a mathematical "John Doe", a **placeholder** which we can use whenever we want to talk about two cases [Epp, 2011]:

- **Case 1:** There exists an entity that can have more possible values, and we don't exactly know what its current value is, or
- **Case 2**: There exists two or more entities that could possibly satisfy some condition that we care about, and we want to consider all such entities, and do not want to be restricted to consider only one.

This notion of a variable as a placeholder is very powerful. It allows us to create a variety of algebraic computations with variables as if they were explicit numbers, which, in turns, allows us to represent, analyze and solve a variety of problems.

When thinking about variables, we typically distinguish between two types:

- **Independent variables** - variables that can take on different values independent of other variables we may care to measure
- **Dependent variables** - variables that change value in relation (response) to other variables in our system

Example 3: Let's consider some examples of independent and dependent variables.

- **Effect of study time on the success in some course** Let's assume we are interested in observing the effect that the time a student investes into studying has on their success in this course. Under two

simplifying assumptions, that all of the student's study time is perfectly productive time, and that the student has an unconstricted time they could decide to devote to studying, in this example, the study time is the independent variable, and the final grade in the course, as a representation of the success in the course is the dependent variable.

- **Effect of a prescribed medication on a disease severity** Independent variable: the dose of a prescribed medication. Dependent variables: frequency and severity of the symptoms of a disease
- **Effect of weekly exercise on a person's fitness**: Independent variables: frequency, type and intensitiy of a person's workout. Dependent variables: a person's weight, stamina, etc...

Based upon this classification, we notice that mathematical objects, such as variables, can be related in various ways, and we typically represent that relationship as a **function**.

2.5.2 Functions

One of the core concepts in discrete math is that of the **function**. In it's most simple form, an function is simply a mapping from one set of numbers (or items) to another set of numbers (or items).

You might be familiar with the concept of a function in algebra, where you can calculate a variable y based on an input x.

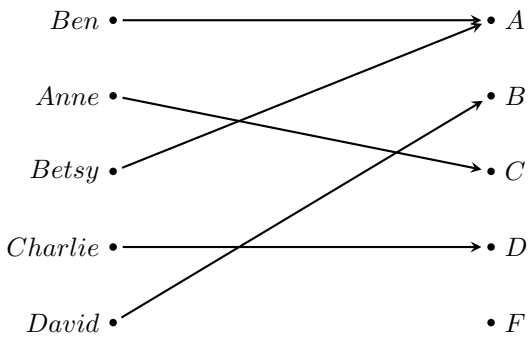

Figure 2.1: Example: Assigning grades

In algebra, a function is seen a relationship between one or more **inputs (input variables)**, and one or more **outputs (output variables)**.

$$\underbrace{f}_{\text{name of a function}}\ (\ \underbrace{x}_{\text{input variable}}\ ) = \underbrace{y}_{\text{output variable}} \tag{2.2}$$

Example 4: Let's consider some examples, and analyze whether or not those are functions:

- **Identity function:** Let's consider some function $f_1(x)$, defined as:

  $$f_1(x) = x, \forall x \text{ our universe of discourse}$$

  The given function $f_1(x)$ takes some input x, and just propagates it to the output (maps input into the output). Such a functions is typically called the **identity function**.
- **Eraser function:** Let's consider function $f_2(x)$ that takes some integer x as an input, and maps every such integer into a 0. Is this function $f_2(x)$ a valid function?

- **Computing an average value in the list of numbers**: Let's consider the folowing Python code below, that takes some list of numbers as an input, and finds the average value of the elements in the list. Is this a valid function?

```
# Python program to get average of a list
  def Average(lst):
    return sum(lst) / len(lst)

  # Driver Code
    lst = [15, 9, 55, 41, 35, 20, 62, 49]
    average = Average(lst)

  # Printing average of the list
    print("Average of the list =", round(average, 2))
```

Extending this perspective, we can think of a function as a **mapping** or a **transformation** between inputs and output, i.e., **a function is a mathematical statement that maps input into outputs**.

This perspective immediately triggers a question: "Is there a way for us to characterize such mappings a bit more formaly, for example by providing more information about inputs and outputs?"

The answer to that questions is "yes", but to show it, we need to introduce another discrete structure we will rely heavily in this course, a **set.**[1]

Definition 2.6 *A set is an* **unordered collection of objects**, *called* **elements** *or* **members** *of a set. A set is said to* **contain** *its elements. We write $a \in A$ to denote that some element a is an element of set A, and $a \notin A$ to denote that a is not an element of the set A.*

Given the definition of a set, we can formally define a function as follows.

Definition 2.7 *Let A and B be some nonempty sets (sets containing at least one element). A function $f : A \to B$ from set A to set B is an assignment of exactly one element of B to each element of A. We write $f(a) = b$ if b is a unique element of B, assigned by the function f to the element $a \in A$.*

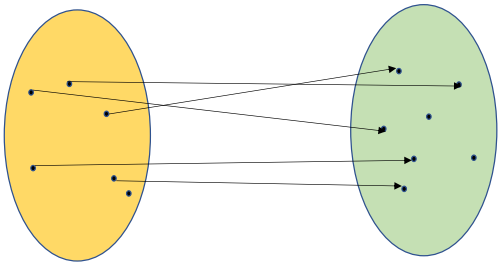

Figure 2.2: A graphical representation of a function as a mapping between two sets

2.5.2.1 Domain and Codomain of a Function

Definition 2.8 *If f is a function from A to B, we refer to A as a* **domain** *of f and B as a* **codomain**.

[1] A whole lecture on sets and set theory is coming soon.

Example 5: Let's consider some example functions, and find their domains and codomains:

- Let function f be a function $f : A \to B$, where $A = \{1, 3, 5\}$ and $B = \{2, 4, 6\}$. This function maps odd numbers 1, 3, 5 into their even counterparts, $1 \to 2$, $3 \to 4$ and $5 \to 6$. Its domain is $A = \{1, 3, 5\}$, and its codomain set $B = \{2, 4, 6\}$.
- Let f be the function that takes a student's name as input, and returns that student's final grade in some course. For example, $f(\text{Jane Smith}) = A$, $f(\text{Burt Beghs}) = \text{F}$. The domain of this function is the set of all student names, and its codomain is the set of possible grades, $\{A, B, C, D, E\}$.
- Let f be the function that assigns the last two bits of some bit string of length of at least 2 bits to that string. For examle, $f(101100) = 00$. The domain of that function is the set of all bit strings of length 2 or more, and its codomain is the set $\{00, 01, 10, 11\}$.

A function is called **real-valued** if its codomain is the set of real numbers, and it is called **integer-valued** if its codomain is the set of integers. Two real-valued or integer-valued functions with the same domain can be added, as well as multiplied as follows:

Definition 2.9 *Let f_1 and f_2 be functions from A to $\mathbb{R}$. Then $f_1 + f_2$ and $f_1 f_2$ are also functions from A to $\mathbb{R}$, defined for all $x \in A$ as follows:*

$$(f_1 + f_2)(x) = f_1(x) + f_2(x) \tag{2.3}$$
$$(f_1 f_2)(x) = f_1(x) f_2(x) \tag{2.4}$$

2.5.2.2 Range of a Function

Definition 2.10 *If $f(a) = b$, we say that b is the* **image** *of a, and a is the* **preimage** *of b. The* **range** *or* **image** *of f is the set of all images of elements of A.*

Example 6: Let's consider some example functions, and find their domains, codomains and ranges:

- Let f be a function that, given a student's ID, defined as a 5-digit integer, for example 12345, returns that students first and last name. The domain of this function is the set of all possible student ID, i.e., the set of all allowed 5-digit integers, and its codomain is the set of possible students' first and last names. The range of f is the set of the first and last names of all of the active/enrolled students.

2.5.2.3 Partial Functions

You must have exprienced cases where some program you wrote, or interacted with may not have produced the correct value of the function for all elements in the domain of the function. For example, a program may fail because the provided input would make it stuck in the infinite loop, lead to an overflow, or something similar.

In mathematics, we allow for something similar to happen. We allow there to exist cases where some elements of domain do not produce the expected output. To account for such cases, we often want to deal with functions that are defined only for a subset of the domain. In order to do that, we introduce the idea of a partial function.

Definition 2.11 *A* **partial function** *f from a set A to a set B is an assignment of each element a in a subset of A, called the* **domain of definiiton of** *f, of a unique element $b \in B$. The sets A and B are still called the domain and the codomain of f, but we say that f is* **undefined** *for elements in A that are not in the domain fo definition of f.*

Remark: When the domain of definition of f is equal to A, we say that f is a total function.

Example 7: Let $f : \mathbb{Z} \to \mathbb{R}$ be a function that computes a square root of some integer number, $f(x) = \sqrt{x}$. This function is a partial function. Its domain of definition is the set of nonnegative integers, its codomain are all real numbers, and its range is the set of real number that are the square roots of the integers in the domain.

Definition 2.12 *Let f be a function from A to B, and let S be a subset from A. The image of S under the function f is the subset of B that consists of images of elements of S. We typically denote the image of S as $f(S)$*

$$f(S) = \{t | \exists s \in S(t = f(s))\} \tag{2.5}$$

2.5.2.4 Equality Between Functions

Two functions are said to be **equal** if the have:

- The same domain,
- The same codomain, and
- Map each element of their common domain to the same element of their common codomain.

2.5.3 One-to-One and Onto Functions

When thinking about mapping between domains and codomains, we distinguish between several different types of functions, namely, injection, surjection and bijection. Let's see what are those.

Definition 2.13 *Some function f is said to be an* **injection** *or* **one-to-one** *if and only if $f(a) = f(b)$ implies that $a = b$ for all a and b in the domain of f.*

Remark: We can express that f is one-to-one using quantifiers:

$$\forall a \forall b \ (f(a) = f(b)) \to a = b$$

or equivalently:

$$\forall a \forall b \ (a \neq b \to f(a) \neq f(b))$$

where the universe of discourse if the domain of the function.

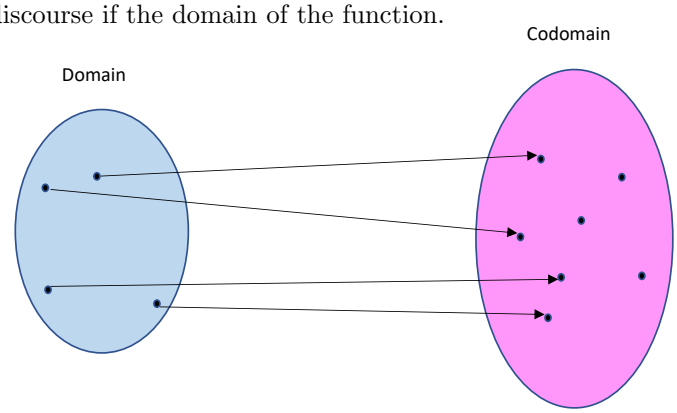

Figure 2.3: A graphical representation of a one-to-one (injection) function.

Example 8: Let consider the following functions. Are they injections?

- Function f from $\{w, x, y, z\}$ to $\{100, 200, 300, 400, 500\}$, where:

$$\begin{aligned} f(w) &= 100 \\ f(x) &= 200 \\ f(y) &= 300 \\ f(z) &= 500 \end{aligned}$$

  Yes, the given function is an injection because f mapes every element of its domain into a different value in its codomain.
- Function $f : \mathbb{Z} \to \mathbb{Z}$, defined as $f(x) = x^2$.
  No, the given function is not an injection, because two different integers from the domain, for example -5 and 5 map into the same number in the codomain, 25.

Definition 2.14 *Some function f is said to be an* **surjection** *or* **onto** *if and only if for every element $b \in B$ there is an element $a \in A$, such that $f(a) = b$.*

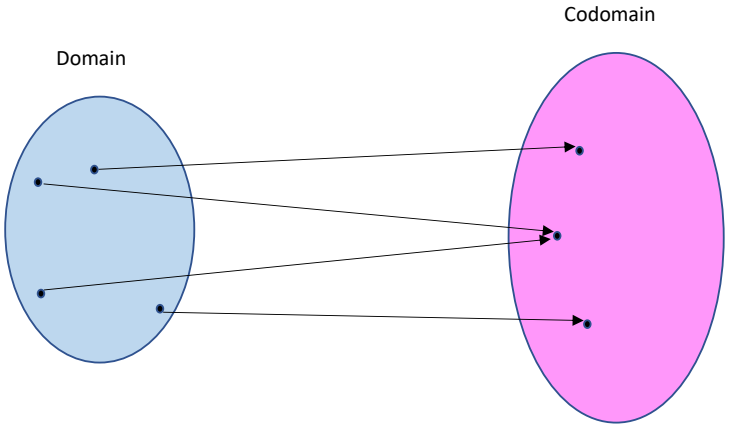

Figure 2.4: A graphical representation of a onto (surjection) function.

Example 9: Let consider the following functions. Are they surjections?

- Let f be a function from set $\{p, r, s, t\}$ to $\{a, b, c\}$, defined as:

$$\begin{aligned} f(p) &= a \\ f(r) &= b \\ f(s) &= c \\ f(t) &= c \end{aligned}$$

  Yes, the given function is a surjection becase all three elements of its codomain are images of the elements in the domain.
- Let $f : \mathbb{Z} \to \mathbb{Z}$ be a function defined as $f(x) = x^2$.
  No, the given function is not a surjection, because there doesn't exist an integer whose squared value is negative. For example, there doesn't exist x such that $f(x) = -5$.

Definition 2.15 *Some function f is said to be an* **bijection** *or* **one-to-one correspondance** *if and only if it is both injection (one-to-one) and surjection (onto).*

Example 10: Let consider the following functions. Are they bijections?

- Let A be a set. The **identitiy function** on A is a function $\iota_A : A \to A$, where $\iota_A(x) = x$ for all $x \in A$. **Yes,** this function is a bijection, because it is both one-to-one and onto.

Summary: Suppose that $f : A \to B$

- **To show that f is injective:** Show that if $f(x) = f(y)$ for arbitrary $x, y \in A$ with $x \neq y$, then $x = y$.
- **To show that f is not injective:** Find particular elements $x, y \in A$ such that $x \neq y$, but $f(x) = f(y)$.
- **To show that f is surjective:** Consider an arbitrary $y \in B$, and find an element x inA such that $f(x) = y$.
- **To show that f is not surjective:** Find a particular element $y \in B$ such that $f(x) \neq y \forall x \in A$.

2.5.3.1 Increasing, Strictly increasing, Decreasing and Strictly Decreasing Functions

Definition 2.16 *A function f whose domain and codomain are subsets of the set of real numbers is called:*

- **Increasing function**, *if $f(x) \geq f(y)$ whenever $x < y$ and x and y are both in the domain of f.*
- **Strictly increasing function**, *if $f(x) < f(y)$ whenever $x < y$ and x and y are both in the domain of f.*
- **Decreasing function**, *if $f(x) \leq f(y)$ whenever xy and x and y are both in the domain of f.*
- **Strictly decreasing function**, *if $f(x) > f(y)$ whenever $x < y$ and x and y are both in the domain of f.*

When reasoning about increasing and decreasing functions, it is typically helpful to represent them graphically, and as it turns out, we can easily do that, by defining a graph of a function.

Definition 2.17 *Let f be a function from the set A to the set B. The* **graph** *of the function f is the set of ordered pairs $\{(a, b)|a \in A$ and $f(a) = b\}$.*

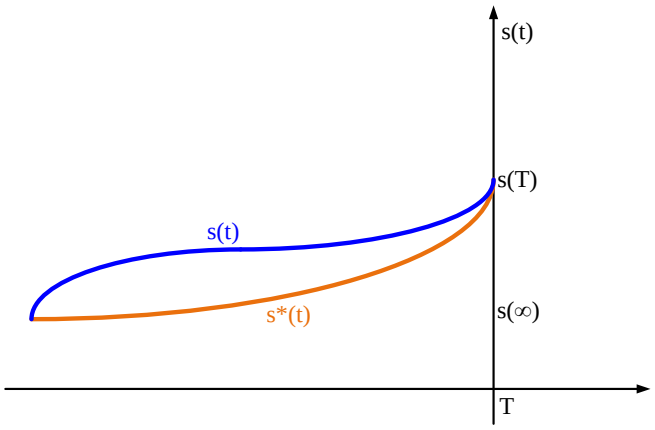

Figure 2.5: An example graphical representation of some functions $s(t)$ and $s^*(t)$.

2.5.4 Inverse Functions and Compositions of Functions

Let's now consider a one-to-one correspondence (a bijection) f from set A to the set B. Because f is an onto function (a surjection), every element in B is an image of some element in A. Because f is also a one-to-one function, every element in B is the image of a *unique* element in A. These facts allow us to define a new function from B to A that reverses the correspondence given by f. This leads to the definition of an inverse function.

Definition 2.18 *Let f be a one-to-one correspondance (bijection) from some set A to some set B. The* **inverse function** *of f is the function that assigns to an element b belonging to B the unique element $a \in A$ such that $f(a) = b$. The inverse function of f is typically denoted as f^{-1}.*

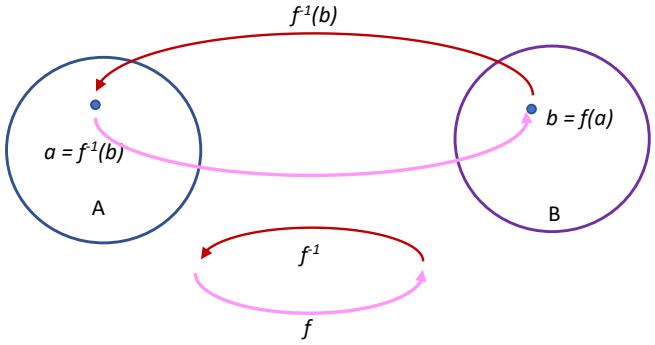

Figure 2.6: A graphical example of function f and its inverse f^{-1}.

Example 11: Let's consider some function $f : \mathbb{R} \to \mathbb{R}$, defined as:

$$f(x) = 5x + 7 \tag{2.6}$$

- Is the given function invertible? **Yes,** the given function is a one-to-one correspondence (bijection), and therefore has an inverse.
- If the function is invertible, find its inverse. **To find thr inverse,** we proceed as follows:

$$\begin{aligned} \text{f is a bijection} \quad &\to \quad f(x) = y \forall x, y \in \mathbb{R} \\ &\leftrightarrow \quad 5x + 7 = y \\ &\leftrightarrow \quad x = \frac{y-7}{5} \end{aligned} \tag{2.7}$$

  By the definition of an inverse function, it follows that:

$$f^{-1}(y) = \frac{y-7}{5} \tag{2.8}$$

Example 12:[2] Given a function $g : T \to T$, defined as:

`For all string` t `in` T`,` $g(t)$ `= the string obtained by writing the characters of` t `in reverse order` determine whether or not $g(t)$ is invertible, and if it is, find its inverse.

To solve this problem, we observe that if the characters of t are written in reverse order, and then written in the reverse order again, the origina string is recovered. Therefore, the given function is invertible, and given any string $t \in T$, $g^{-1}(t)$ `= the unique string that, when written in reverse orhder, equals` t `= the strings obtained by writing the characters of` t `in reverse order =` $g(t)$ So, in this example, the function and its inverse are the same, $g = g^{-1}$.

[2]Example originally presented in [Epp, 2011].

2.5.4.1 Composition of Functions

Definition 2.19 *Let g be a function from the set A to the set B, and let f be a function from the set B to the set C. The composition of the functions f and g, denoted fro all $a \in A$ by $f \circ g$, is defined as:*

$$(f \circ g)(a) = f(g(a)) \tag{2.9}$$

Remark: In other words, a composition $f \circ g$ is the function that assigns to the element $a \in A$ the element assigned by f to $g(a)$. In doing so, it is important to note that a composition $f \circ g$ cannot be defined unless the range of g is the subset of the domain of f.

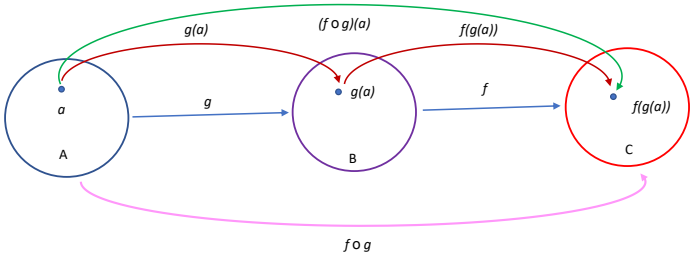

Figure 2.7: A graphical example of the composition of functions f and g.

Example 13:[3] Consider two functions $f : \mathbb{Z} \to \mathbb{Z}$, defined as $f(x) = x + 1$ and $g : \mathbb{Z} \to \mathbb{Z}$, defined as $g(x) = x^2$ for all $x \in \mathbb{Z}$.

- Find $f \circ g$?
- Is $f \circ g = g \circ f$?

Let's start solving this problem by first finding $f \circ g$ as follows:

$$\begin{aligned} f \circ g &= f(g(x)) \\ &= f(\underbrace{x^2}_{y}) \\ &= f(y) = y + 1 \\ &= x^2 + 1 \end{aligned} \tag{2.10}$$

Let's now find $g \circ f$ as follows:

$$\begin{aligned} g \circ f &= g(f(x)) \\ &= g(\underbrace{x + 1}_{y}) \\ &= g(y) \\ &= y^2 = (x + 1)^2 \end{aligned} \tag{2.11}$$

As we can see, the two compositions are not the same, $f \circ g \neq g \circ f$

2.6 Some Important Integer Functions

We have already seen many examples where our programs expects integers, yet real life values that we are dealing with are real numbers or fractions. This issue exposes a question of conversion from fractions or arbitrarty real number into integers.

[3] Example originally presented in [Epp, 2011].

In this part of the lecture, we will explore several such conversion functions and analyze their properties.

2.6.1 Floor and Ceiling Functions and Their Properties

The first two functions used to convert real numbers into integers are **floor (greatest integer)**, and **ceiling (least integer)**, defined as follows.

Definition 2.20 *The floor (greatest integer) function is defined as:*

$$\lfloor x \rfloor = \textit{the greatest integer less than or equal to } x \tag{2.12}$$

Similarly, the ceiling (least integer) function is defined as:

$$\lceil x \rceil = \textit{the least integer greater than or equal to } x \tag{2.13}$$

Graphs of the floor and the ceiling functions are represented in Figure 2.8, and we can observe that, for both functions, their graphical representations form staircase-like patterns above and below the line $f(x) = x$.

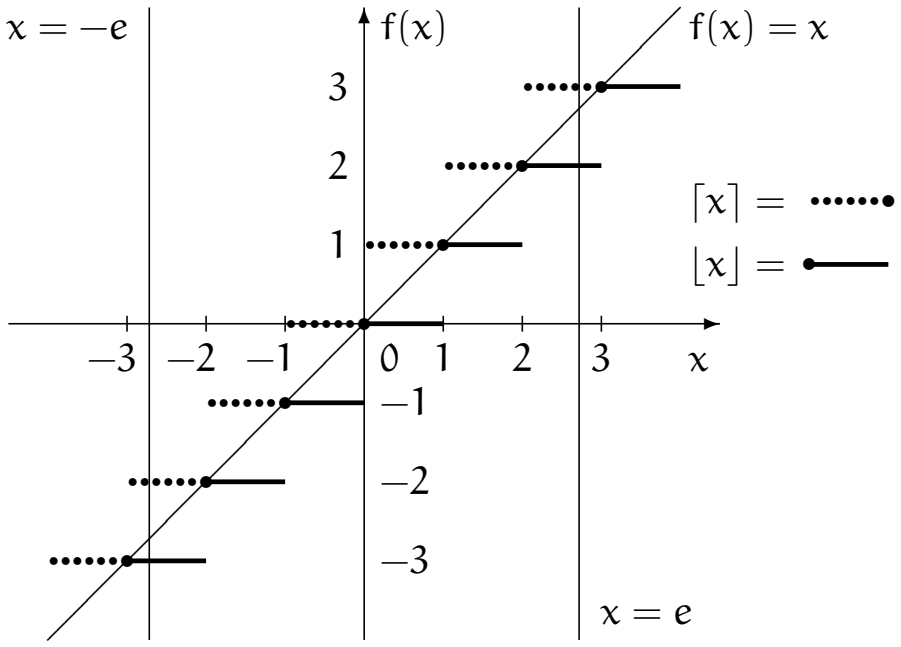

Figure 2.8: A graphical representation of the floor and the ceiling functions. Graph credit: *Knuth et al., Concrete Mathematics, 2nd Edition, 1994*

From graph 2.8, we see that:

$$\begin{aligned} \lfloor e \rfloor &= 2 \ \lfloor -e \rfloor = -3 \\ \lceil e \rceil &= 3 \ \lceil -e \rceil = -2 \end{aligned}$$

We can observe several facts about floors and ceiling from their graphical representation [Knuth, 1994]:

- Since the floor lies on or below the diagonal line $f(x) = x$, and similarly ceil on or above the same line, it follows that:

$$\lfloor x \rfloor \leq X \text{ and } \lceil x \rceil \geq x \tag{2.14}$$

- The floor and the ceil function are equal precisely at the integer points:

$$\lfloor x \rfloor = x \iff \lceil x \rceil \iff x \in \mathbb{Z} \tag{2.15}$$

- When floor and ceiling differ, then the ceiling is exactly 1 higher than the floor:

$$\lceil x \rceil - \lfloor x \rfloor = 1 \implies x \notin \mathbb{Z} \tag{2.16}$$

- If we shift the diagonal line down one unit, it lies completely below the floor function, so $x - 1 < \lfloor x \rfloor$. Combining this with a similar observation for floor, we can write:

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1 \tag{2.17}$$

- The functions are reflections of each other about both access:

$$\begin{aligned} \lfloor -x \rfloor &= -\lceil x \rceil \\ \lceil -x \rceil &= -\lfloor x \rfloor \end{aligned}$$