Structures and Operations

ian.mcloughlin@atu.ie

January 23, 2023

Are all programming languages equally capable? What are the limits of computation? How can we discuss computation without worrying about the details of specific machines? To answer these questions, we need a simple way of describing problems. In the following article we will define the basic building blocks of computing.

References

We will use two reference texts: Norman Biggs' *Discrete Mathematics*[1] and Michael Sipser's *Introduction to the Theory of Computation*.[2] Another good resource is the Open Logic Text.[3] On the practical side, I recommend also use The Python Tutorial,[4] The Python Software Foundation's official tutorial for Python.

[1] Biggs, Norman L. *Discrete Mathematics*. revised Edition. Oxford Science Publ., 1989.

[2] Sipser, Michael. *Introduction to the Theory of Computation*. Third. Boston, MA: Course Technology, 2013. ISBN: 113318779X.

[3] Open Logic Project. *Open Logic Project Builds*. Dec. 21, 2022. URL: `https://builds.openlogicproject.org/` (visited on 01/23/2023).

[4] *The Python Tutorial — Python 3.11.1 documentation*. Jan. 22, 2023. URL: `https://docs.python.org/3/tutorial/` (visited on 01/22/2023).

Sets

A set is a collection of objects, usually denoted using curly braces.[5] For example, the set A below contains the three objects 1, 2, and 3. We call these objects elements of the set.

$$A = \{1, 2, 3\}$$

Sets can be infinite, in which case the elements can be identified by an algorithm or property. In this case, we usually assume the infinite set of counting numbers[6] $\mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$ is a given[7].

In the below, we generate the set T of even positive natural numbers with an algorithm. The algorithm says start with the natural numbers and multiply each of them by two.

$$T = \{2n \mid n \in \mathbb{N}\}$$

Similarly, the set P is given by the property that each element is prime. Of course, we could define them by an algorithm such as the brute force algorithms for primes.

$$P = \{p \in \mathbb{N} \mid p \text{ is prime}\}$$

Two important properties of sets are that they are unordered and that each element is distinct. Note there is no mention of order in the definition *collection of objects*[8]. Likewise, the idea of an *object* is that it is unique — we did not say an instance of an object or anything like that.

We say B is a subset of A if all the elements in B are also in A. When B has k elements, we sometimes say B is a k-subset of A. Under this definition, the empty set and A itself are always subsets of a set A.

[5] Sipser, Michael, *Introduction to the Theory of Computation*, p. 3.

[6] We call these the natural numbers and $\mathbb{N}_0$ is the set of natural numbers including zero.

[7] Sometimes it is convenient to not include the element 0, in which case we denote the set $\mathbb{N}$.

[8] We can create an order or ordering of a set if we wish, but that is something we must treat alongside the set.

Note that a set B is an object itself, and so might be an element of a set A. In this case, we are not saying that the elements of B are individually in A, although that could also be the case. The distinction is important[9].

[9] Bertrand Russel is known for Russell's paradox about a set R which is the set of all sets that do not contain themselves. A set seemingly may contain itself — consider the set of all sets. Does R contain itself?

In Python, `set` is a built-in type. We can create a new set `S` with three elements as follows.

```
S = {1, 2, 3}
```

Be careful with the empty set in Python. The statement `{}` creates an empty dictionary not an empty set. You have to use `set()` instead. You can test membership using the `in` operator.

We define the union of sets S and T, denoted $S \cup T$, as the set of all the elements in S or T. Likewise, the intersection $S \cap T$ means the set of all the elements in both S and T. Finally, the difference $S \setminus T$ means the set of all the elements in S but not in T.

In Python, we use different symbols for these operators.

```
S = {1, 2, 3}
T = {3, 4, 5}

S | T # Union: {1, 2, 3, 4, 5}
S & T # Intersection: {3}
S - T # Difference: {1, 2}
T - S # Different difference: {4, 5}
```

Tuples

When order matters, we use tuples. A tuple is a finite sequence.[10] A sequence is a list of objects, usually stipulated to come from a set or sets. A tuple of length k is sometimes called a k-tuple, although a 2-tuple is usually just called a pair. In the following, t is a tuple of length 4, a 4-tuple. The first element is 1 and the last, the fourth element, is 16.

[10] Sipser, Michael, *Introduction to the Theory of Computation*, p. 6.

$$t = (1, 4, 9, 16)$$

Tuples are basically the same as lists of arrays in programming languages. The word *list* implies an order — we can talk about the first thing on a list, if it exists. Because they are ordered, it also makes sense to have more than one copy of an element in a tuple so long as they are in different positions.

In Python, `list` and `tuple` are in fact different types. They are very similar, with the main difference being that tuples are immutable and lists are mutable. Each has its advantages in different

contexts. Lists are very flexible. Tuples are hashable. In the following example, t is a tuple and l is a list. Note that lists use brackets `[]` and tuples use parentheses `()`.

```
l = [1, 4, 9, 16] # list
l[0] # 1
l[-1] # 16
l[2] = 8 # Okay: l is now [1, 4, 8, 16].

t = (1, 4, 9, 16) # tuple
t[0] # 1
t[-1] # 16
t[2] = 8 # Not okay: does not support assignment.
```

Strings

We will use the terms *string* and *tuple* interchangeably.[11] Generally, we will use the word string when we mean that all the elements in the tuple come from a single set that we will call an *alphabet*. We also generally write strings without parentheses and commas.

[11] Sipser, Michael, *Introduction to the Theory of Computation*, p. 14.

In the following, t is a tuple.

$$t = (0, 1, 1, 0, 1)$$

Whereas in the following, s is a string.

$$s = 01101$$

Clearly, with strings, it is important that each individual symbol represents an individual element. For example, the tuple $(1, 4, 9, 15)$ should not be written in string form as 15 is a single element represented by two symbols. Someone could reasonably assume 14915 contains five elements: 1, 4, 9, 1, and 5.

In Python, `str` is a built-in type. You delimit strings with either single or double quotes, the same on either side. They act more like tuples than lists, in the sense that they are immutable.

Strings are more complex than tuples, however. They have several methods associated with them, such as `lower()` and `upper()`. They also have to contend with Unicode, symbols which can take up a variable number of bytes. The alphabet for Python strings is the set of all Unicode code points.

```
s = 'Hello'
s[0] # 'H'
s[-1] # 'o'
s[2] = 'n' # Not okay: does not support assignment.
```

Languages

A language is a set of strings over an alphabet.[12] You start with any set, which you call the alphabet A. You can then form the set of all possible strings of A which is sometimes written as A^*. We call the $*$ the Kleene star after Stephen Cole Kleene.

A language is then any subset of A^*. Note that A^* is a subset of itself, so it is a language, as is the empty set $\{\}$.

Sometimes it is useful to discuss all strings of some fixed length k over an alphabet A. That is usually denoted A^k. For example, when A is the set $\{0, 1\}$ then:

$$A^2 = \{00, 01, 10, 11\}$$

$$A^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

Then we can define A^* as follows.

$$A^* = \bigcup_{i=0}^{\infty} A^i$$

Note that A^* is infinite, but every element contained in it is of finite length.

We often call a set of languages a class. So a class is a set of languages, a language is a set of strings, and a string is a tuple over an alphabet, which itself is a set.

[12] Sipser, Michael, *Introduction to the Theory of Computation*, p. 14.

Languages are purely syntactical. Consider the set of prime numbers $P = \{2, 3, 5, 7, 11, 13, \ldots\}$. Here we have written the numbers using decimal notation. We can consider each number as a string written over the alphabet $A = \{0, 1, 2, 3, 4, 5, 6, 7, 9\}$. In that sense, the number thirteen is represented by the 2-tuple $(1, 3)$.

Of course, we could write the prime numbers in binary. In that case $P_B = \{10, 11, 101, 111, 1011, 1101, \ldots\}$. Then the number thirteen is the tuple $(1, 1, 0, 1)$. Sometimes it is handier to write binary numbers with the least significant bit at the start. Then thirteen is the string 1011.

Which of these is the correct way to write the number thirteen? There is no right way — which might seem frustrating. But we brought out an important distinction between syntax[13] and semantics[14]. We can now consider purely syntactical problems.

Is syntax by itself of any use or do we need semantics to say anything useful about the world? Turns out the former is true, a topic for another day.

[13] **syntax** *noun*: the arrangement of words and phrases to create well-formed sentences in a language; the structure of statements in a computer language.

[14] **semantics** *noun*: the meaning of a word, phrase, or text.

Maps

The term *function* can mean a few different things in computing. It is often used interchangeably with the term *algorithm*. However, we want to capture the idea of linking inputs to outputs with worrying about how to get there. In this case we will use the term *map*.

Informally, a map from a set X to a set Y is an assignment of the elements of X to elements in Y.[15] We need a better definition than that, so we will first define the Cartesian product[16].

The Cartesian product, $X \times Y$, of two sets X and Y is the set of all possible 2-tuples (or pairs) where the first element comes from X and the second from Y. Where X and Y are both the set of real numbers $\mathbb{R}$ we get the Cartesian plane.

A simpler example is as follows. Let S be the set $\{1, 2, 3\}$ and T the set $\{a, b\}$. Then:

$$S \times T = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$$

Now, a function f from X to Y is a subset of $S \times T$ such that every element of X is the first element of exactly one element in f. For example, in the following f_1 and f_2 are functions from S to T where S and T are as above.

$$f_1 = \{(1, a), (2, a), (3, b)\}$$
$$f_2 = \{(1, a), (2, a), (3, a)\}$$

However, the following are not functions from S to T. In the first one, the element 2 is not mapped to anything[17]. In the second, 1 is mapped to two things in T.

$$\{(1, a), (3, a)\}$$
$$\{(1, a), (1, b), (2, a), (3, a)\}$$

Note that all of X must be used, but not all of Y. In cases where all of Y is used, we say the function is *onto* Y or *surjective*, and call it a *surjection*. In cases where each element of X is mapped to a different element of Y we say the function is *one-to-one* or *injective*. In the examples above, f_1 is surjective and f_2 is not. Neither is injective.

The following function from $X = \{1, 2, 3\}$ to $Y = \{1, 4, 9\}$ is both injective and surjective. We say such functions are *bijective* and call them *bijections*.

$$f = \{(1, 1), (2, 4), (3, 9)\}$$

In Python, we define using the `def` keyword. Note that in most programming languages you must give an algorithm to compute the function.

```
def f(x):
  ans = 1
  for i in range(x):
    ans = ans * i
  return ans
```

[15] Sipser, Michael, *Introduction to the Theory of Computation*, p. 7.

[16] Named after the famous philosopher Rene Descartes.

[17] Sometimes we call this a partial function.

In the earlier definition of a function we made no reference to algorithms. To mimic those kinds of functions we can just use a dictionary.

```
def f(x):
  pairs = {1: 1, 2: 4, 3: 9}
  return pairs[x]
```

Further Topics

Now that we have defined our basic building blocks, we can discuss topics in computation and computability.