Sets 2

Today:

- ☐ Powerset
- ☐ Cross product (= Cartesian product)
- ☐ Axiomatic set theory — ZFC
- ☐ Languages (sets of strings)

Announcements:

- Quiz 2 on Friday
- In-person instruction resumes next week. Please come wearing a well-fitting N95 mask. If you don't want to come back to in-person instruction, or you can't currently come because you're sick or tested positive, there shouldn't be any big problem with remote attendance until the final. I fully expect the final to be in-person, and I won't be making accommodations for people wanting to take it remotely.
- Poll results

Review Last time I described various operations on sets, like union, intersection, complement, set difference, and symmetric difference. We looked at a few identities, like De Morgan's law (but now for sets). I told you a cool paradox, Russell's paradox, which follows if we let you "define" the set $R = \{x : x \notin x\}$. I ended by describing another operation on sets, the *powerset* (which I've decided to spell solid today!). That last part rushed, so I'm going to start there for today.

1 Powerset

You can regard powerset as a unary operator on sets, like complement. It takes in a set and it spits out a set.

For S an arbitrary set, we define $\mathcal{P}(S) = \{A : A \subseteq S\}$, the *powerset* of S. In English: the powerset of a set S is the set of all subsets of S.

Practice: What is $\mathcal{P}(\{a, b\})$? $\mathcal{P}(\{\emptyset, 4, \texttt{fig}\})$? What is $\mathcal{P}(\emptyset)$? What is $\mathcal{P}(\mathbb{N})$?

If S is a finite set, say $|S| = n$, what is $|\mathcal{P}(S)|$, the cardinality (size) of the powerset of S. It is always 2^n. In this case, you could arbitrarily order the elements of S and then each element of $\mathcal{P}(S)$ could be named by an n-bit binary number. In this way, $\mathcal{P}([1..N])$ is in one-to-one correspondence with n-bit binary strings, which is a good way of thinking of $\mathcal{P}(\cdot)$, at least when it's applied to finite sets.

2 Cross Products

You can regard the cross product as a binary operator on sets, just as union and intersection are. The cross-product operator take in a pair of sets and it spits out a set. We use a $\times$ symbol, in infix notation, to represent the cross product. It's defined by saying that

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\}.$$

Practice: What is $\{a, b\} \times \{c, d, e\}$? $\mathbb{Z} \times \mathbb{Z}$? $\mathbb{R} \times \mathbb{R}$? $\emptyset \times \mathbb{N}$? $\mathbb{B} \times \mathbb{B}$?

What about a something like $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$. We do *not* regard $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$ as a shorthand for $\mathbb{R} \times (\mathbb{R} \times \mathbb{R})$ or $(\mathbb{R} \times \mathbb{R}) \times \mathbb{R})$. If we did, then we'd see things like $(3, (2.1, 4)) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}$. Instead, $A \times B \times C$ is actually *a different operator* than $X \times Y$ done twice. It's the *3-fold cross product*, defined by

$$A \times B \times C = \{(a, b, c) : a \in A \text{ and } b \in B \text{ and } c \in C\}.$$

And you can keep going. Regard $A \times B$, $A \times B \times C$, and $A \times B \times C \times D$, as some sort of funky notation that is actually capturing different operators: CrossProduct2(A, B), CrossProduct3(A, B, C), CrossProduct4(A, B, C, D).

Another word for *cross product* is *Cartesian product*

Cross products of a set with itself come up so often that we have a shorthand for it: $A^2 = A \times A$, $A^3 = A \times A \times A$, and so on. So when we write something like $\mathbb{B}^2$ we really mean $\mathbb{B} \times \mathbb{B}$, the set of ordered pairs of bits. I've been used that notation, because when I speak of a binary operator like AND or OR, I think of its input as a point in $\mathbb{B}^2 = \mathbb{B} \times \mathbb{B}$. Similarly, points in the plane live in $\mathbb{R}^2$, those in space have coordinates in $\mathbb{R}^3$, grid points of the plane are $\mathbb{Z}^2$, and so on.

3 Axiomatic set theory

In the early 20th century problems like Russell's paradox were driving logicians to develop firm mathematical foundations for set theory. As a way to do this, logicians developed *axioms* to try to capture all of set theory. Axioms (or axiom schemas) are lists of logical statements whose consequences we investigate and whose models define a realm of study.

The following text and list is lightly edited from Wolfram MathWorld: Zermelo-Fraenkel Axiom:

> The Zermelo-Fraenkel axioms are the basis for *Zermelo-Fraenkel set theory*. In the following (Jech 1997, p. 1), $\exists$ stands for exists, $\forall$ means for all, $\in$ stands for "is an element of," $\emptyset$ for the empty set, $\rightarrow$ for implies, $\wedge$ for AND, $\vee$ for OR, and $\leftrightarrow$ for "is equivalent to."

1. **Axiom of Extensionality**: If X and Y have the same elements, then $X = Y$.
$$\forall u(u \in X \leftrightarrow u \in Y) \rightarrow X = Y.$$
2. **Axiom of Pairing**: For any a and b there exists a set $\{a, b\}$ that contains exactly a and b.
$$\forall a \forall b \exists c \forall x(x \in c \leftrightarrow (x = a \vee x = b)).$$
3. **Axiom of Subsets**: If ϕ is a property (with parameter p), then for any X and p there exists a set $Y = \{u \in X : \ \phi(u, p)\}$ that contains all those u in X that have property ϕ.
$$\forall X \forall p \exists Y \forall u(u \in Y \equiv (u \in X \wedge \phi(u, p))).$$
4. **Axiom of Union**: For any X there exists a set $Y = \bigcup X$, the union of all elements of X.
$$\forall X \exists Y \forall u(u \in Y \leftrightarrow \exists z(z \in X \wedge u \in z)).$$
5. **Axiom of the Power Set**: For any X there exists a set $Y = \mathcal{P}(X)$, the set of all subsets of X.
$$\forall X \exists Y \forall u(u \in Y \leftrightarrow u \subseteq X).$$
6. **Axiom of Infinity**: There exists an infinite set.
$$\exists S[\emptyset \in S \wedge (\wedge \forall x \in S)(x \cup \{x\} \in S)).$$
7. **Axiom of Replacement**: If F is a function, then for any X there exists a set $Y = F[X] = \{F(x) : \ x \in X\}$.
$$\forall x \forall y \forall z(\phi(x, y, p) \wedge \phi(x, z, p) \rightarrow y = z) \longrightarrow \forall X \exists Y \forall y(y \in Y = (\exists x \in X)\phi(x, y, p)).$$
8. **Axiom of Foundation**: Every nonempty set has an $\in$-minimal element.
$$\forall S(S \neq \emptyset \rightarrow (\exists x \in S)S \cap x = \emptyset).$$
9. **Axiom of Choice**: Every family of nonempty sets has a choice function.
$$\forall x \in a \exists A(x, y) \rightarrow \exists y \forall x \in a A(x, y(x)).$$

The axiom of choice has been the most "controversial" of these axioms, and sometimes Zermelo-Fraenkel set theory is regarded as excluding it (writing ZFC or ZF when it is or is not included). Informally, the axiom of choice says that given any collection of bins, each containing at least one object, it is possible to make a selection of exactly one object from each bin—even if the collection of bins is infinite.

For more on ZFC, see the Wikipedia article on Zermelo-Fraenkel set theory.

I don't expect you go to through the ZFC list; I am presenting it more for your "cultural" understanding of contemporary mathematics. Axiomatized set theory has shaped our basic understanding of the limits of what can be proven in mathematics. In about two weeks I will mention an interesting question in set theory whose answer is known to be independent of ZFC. And I want you to have at least a vague understanding of what such a statement means.

If you'd like to learn more about the history of the foundations of set theory, it could certainly support a good extra-credit report.

4 Languages

Languages (also called *formal* languages) are sets of central importance in computer science, so it is hard to explain why they are often ignored in discrete math books. I want you to be familiar with the basic notions and vocabulary from formal language theory.

An **alphabet** is a finite, nonempty set. We call its elements **characters.** Examples would include $\{0,1\}$ (the *binary* alphabet) $\{1\}$ (a *unary* alphabet), $\{0,1,2,3,4,5,6,7,8,9\}$, $\{a,b\}$, $\{a,b,\ldots,z\}$, and ASCII (a list of 128 named characters), and printable ASCII (a 95-element subset of these). It is common to use the letter Σ to represent an alphabet.

A **string** is a finite sequence of characters. Examples: `hello`, `this big dog`, or `10110011`. Every string x has a nonnegative *length*, denoted $|x|$, which is the number of characters in it. The length of a string is always finite—conventional strings are not infinite. Strings include the *empty string*, denoted ε, the unique string of length 0.

There's a basic operation on strings, a binary operator called *concatenation*. You just stick the two strings together. So: `Hello` $\circ$ `There` = `HelloThere`. It is routinely written with an implicit operator (just like multiplication in the integers or reals), so $xy = x \circ y$ when x and y are strings. So if $x = y =$ `2` are strings, $xy =$ `22`, not `4`. Naturally $|xy| = |x| + |y|$.

A crucial kind of set in computer science is a set of strings. A set of strings is called a **language**. All of the strings in a language should be composed of characters from a single alphabet Σ.

Here are examples of languages: Numbers written in decimal. Numbers written in binary. 8-bit bytes. 32-bit words. 64-bit words. A list of all prime number, written in decimal. All valid Python programs. All valid Python programs that take in no input and eventually stop when you run them. And all grammatical English sentences. Well, maybe not. That last example is really too vague to permit, as there is no way that people would always agree what is or isn't an English sentence. For natural languages, the property of being grammatical can sort of "fade off," getting less and less reasonable. Just the same, linguistics has motivated a good deal of study in formal language theory.

The concatenation operator on strings can be extended to languages. If $A = \{0,1\}$ and $B = \{00,101\}$ then $A \circ B = \{000,0101,100,1101\} = \{000,100,0101,1101\}$. This is not the same as $A \times B = \{(0,00),(0,101),(1,00),(1,101)\}$. In general, if A and B are languages then $A \circ B = \{x \circ y\colon\ x \in A,\ y \in B\}$.

In formal language theory, $L^2 = L \circ L$, $L^3 = L \circ L \circ L$, and so on. Same notation as in set theory, but with a different meaning! There, L^2 would mean $L \times L$, and $L^3 = L \times L \times L$, and so on. Because languages *are* sets, if A is a language, it's potentially ambiguous what A^n should mean. That said, when L is a language L^n usually refers to n-fold concatenation and not the n-fold cross product.

A few more examples:

$\{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(11)\}$

$\{0,1\} \circ \{0,1\} = \{00,01,10,11\}$

$\{0,00\} \times \{0,00\} = \{(0,0),(0,00),(00,0),(00,00)\}$

$\{0, 00\} \circ \{0, 00\} = \{00, 000, 0000\}$. Note from this example that it is entirely possible that $|A \circ B| < |A| \cdot |B|$; all we can say is that $|A \circ B| \leq |A| \cdot |B|$ (why?).

$\{0,1\}^8 = \{00000000, 00000001, \ldots, 11111111\}$. This set has 256 points, each an 8-bit string.

$\{0,1\}^{32} = \{w\colon\ w \text{ is a 32-bit string}\}$. This set has 2^{32} points, each a 32-bit string.

There are other nice operators on strings besides concatenation. For example, you can extract the i-th character from an n-bit string x, an operation denoted $x[i]$. Sometimes indexing is understood to start at 0, other times, indexing starts at 1. Or you can grab a chunk, a *substring*, of characters from x, an operation that could be denoted $x[i:j]$ or $x[i..j]$. Again, conventions vary as to how indexing is done. (Mathematicians would normally have indexing start at 1, and the two endpoints would be inclusive. But this is not the convention adhered to in common programming languages, or their libraries.)

All of the operations that apply to sets, like union, complement, set difference, and symmetric difference, apply *ipso facto* to languages, because languages *are* sets. But $\circ$ was something unique to languages—a binary operator just for languages. And there's an important unary operator on languages, too: the **star**, or **Kleene closure** of a language L.

Given a language L, we let L^* be the set of all strings $x_1 x_2 \cdots x_n$, for any $n \geq 0$, where each $x_i \in L$. When $n = 0$, we understand this to be the empty string.

Another way of saying the same thing is that $L^* = \bigcup_{n \geq 0} L^n$, where $L^0 = \{\varepsilon\}$.

Examples: $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. $\{1\}^* = \{\varepsilon, 1, 11, 111, \ldots\}$. Or $\{\texttt{dog}\}^* = \{\varepsilon, \texttt{dog}, \texttt{dogdog}, \texttt{dogdogdog}, \ldots\}$. Or $\{a, bb\}^* = \{\varepsilon, a, aa, bb, aaa, abb, bba, aaaa, \ldots\}$. The last language I have listed in **lexicographic order**: list all the strings of length 0; then all the strings of length 1; then all the strings of length 2; and so on. Within a given length, list the strings alphabetically. Where you have arbitrarily ordered the letters of the alphabet so as to make "alphabetical order" a meaningful term.

Note that the way we have defined things, $\varepsilon \in L^*$ for any L (even the empty set).

Warning: Don't confuse the empty set $\emptyset$, the empty string ε, and the singleton language $\{\varepsilon\}$. (A singleton set is a set that has exactly one element.) These are all different things. The empty string is a string. It is not a language. The empty set is a set. It is not a string. The set that contains just the empty string is a perfectly good language. But it isn't a string. The set that contains just the empty set is a perfectly good set. But it isn't a language.

One reason that languages are central to computer science is that they are how we think about problems. You pose a problem by writing a string. For decision questions (yes/no problems), you can understand the task as trying to decide if a string is or isn't in some particular language. Even for optimization problems, a string is coming in and an answer is coming out, so strings seem inevitably in the picture. Our programs are strings. Our books are strings (if you have such poor taste as not to care about their physicality). To a computer scientists, pretty much *everything* is a string. Except, perhaps, the physical artifact that attaches to a kite—which has nothing to do with a string.

Languages that you can build with only the operations union, concatenation, and star are called *regular languages*. You get to start with the singletons that are in your alphabet, as well as

the empty set. If you take ECS120, you'll spend a maybe three weeks or so studying these languages.