Chapter 2

A Bunch of Set Theory and a Bit of Logic

2.1 Representing the world using sets

Set theory is the "machine language" of mathematics. If you think about it, every program that you have ever run on a computer has ultimately produced instructions that get sent to a CPU and the CPU just churns through them. Those CPU instructions are really really low-level: add this 32-bit number to that 32-bit number, grab these bits from memory and move them somewhere else, check this number to see if it's zero, and grab these other set of bits from memory and interpret them like a machine instruction. Somehow, these little primitive instructions make it possible to write programs that download pictures of kittens from the Internet and thereby make your life more fulfilling. At the end of the day, though, it's all bits: 1's and 0's. Your kittens are *represented* by a bunch of bits that your graphics hardware knows how to translate into a vision of furry joy. Your tragic poetry is written in ASCII, which is really a bunch of characters, each of which is represented by a number, which in turn is represented by bits. So your computer consistently operates on *representations* of the things that you really care about.

Set theory is like that. It's *painfully* low-level, and it doesn't understand high-level concepts like programming languages, or numbers for that matter, but it's very very powerful! Many mathematicians and logicians take set theory as *the* foundational tool for building representations of things. If you can't represent your object using the machinery of set theory, then those mathematicians would likely deem your object not so mathematical. This viewpoint is not universal, but it's safe to say that as I write this it is dominant, and the thesis guiding this class is that, regardless of its supremacy and despite its warts, it's *useful*.

2.2 Abstractions make this tractable

On computers and in mathematics, we keep this low-level stuff from exploding our brains by building up layers of *abstraction*. Much of the time we just ignore what's going on at the low-levels, but instead think in terms of higher-level concepts that we really care about. As long as your abstractions are well-designed and clearly explained, you can ignore the details. If your abstractions are "leaky", then you might write down things that make no sense to you but say something meaningful at the lower levels [**?**]. For example, I'm willing to believe that the letter `'a'` comes before `'b'`, so in a sense `'a'` $<$ `'b'`, and the C programming language tells me this, but why would I think that `')'` $<$ `'+'`??? The C language tells me that this is true too, and if I didn't know anything about the *ASCII encoding of characters as numbers*, then I wouldn't understand what's going on. In short, C's abstraction of characters is leaky. If the language didn't let you compare characters in terms of their underlying machine representations, this weird stuff wouldn't happen.[1] Later, we'll see some examples of leaky abstractions in set theory.

[1] In fact, some errors in C code arise from assuming that ASCII is always the relevant character encoding! See for example `https://stackoverflow.com/questions/16400009/why-the-char-comparison-ifc-a-c-z-is-not-portable`

Now don't get me wrong, sometimes it's useful to know what's going on under the hood, if only so you can debug problems or implement abstractions of your own. This is true on both computers and in mathematics. In essence, the ideal is to be a *full stack mathematician* who can work efficiently at a high level, but understand the underpinnings that ensure that this high-level work makes sense.

Finally, bear in mind that we sometimes use low-level concepts in our high-level reasoning. In programming, sometimes all you want is a set of bits. So bits appear in our upper-level abstraction. But be warned: just because you are talking about bits in your high-level, it doesn't mean that they are being *represented* exactly as bits at the low level! Perhaps each of your high-level bits is being represented by *an entire* 32-bit machine word, for convenience of implementation. If you don't care about the space usage, then maybe that's not a big deal. Similarly, we still use sets in our higher-level reasoning too: sometimes you just need a set of kittens to make your day. But your high-level notion of "set of kittens" is going to be interpreted somehow in set theory, and that interpretation may not be readily recognizable after you compile it down. It will surely be a set, because sets are all there is at the bottom, but a set of what? That is to say, we can really think about everything we're doing in mathematics as operations on sets.[2]

Now, to keep the navel gazing to a reasonable minimum, we are not going to build *all* of the mathematical concepts we need from the ground up. In practice, we are going to choose a few primitive notions in addition to sets and work with them. In this document, however, we will do a bit of wallowing in the set-theoretic muck, partly so you have some experience looking under the hood at the low-level parts, partly so you can appreciate the high-level abstractions we use, and partly so that you can do your own trouble-shooting and discern whether your high-level reasoning makes sense. It's good to set some ground-rules for creating and manipulating sets. You may have seen some of these concepts before, but maybe not explained in this painful amount of detail. Enjoy!

2.3 Sets and Logic

Set theory, as we will use it in class, is really an embedding of notation and axioms that are meant to codify what sets we can describe and manipulate, into a logical language, a particular variant of *first-order logic*, which gives us the tools that we need to actually say stuff about sets and prove that the things we say "make sense". The two combine to create what we technically call a *first-order set theory*. So the logical language and the language of sets are necessarily intertwined. Ultimately we want to get comfortable with being relatively informal about our discussions of sets, just like mathematicians tend to be, for the sake of economically communicating the essence of an argument, while leaving some necessary but peripheral formal details implicit. However, we want to make sure that if we really *had* to, we could get super-pedantic and formal about it, literally using first-order logic all the way down.[3]

This is quite analogous to programming practice. Experienced programmers are capable of writing precise code that a computer can execute, but they may still communicate with one another using pseudocode, knowing full well that when push comes to shove they can sit down and implement it: pseudocode is sometimes a more efficient and insightful way to communicate the essence of a program than concrete code. The same is true in mathematics, except that not all mathematicians are equipped to fill in all the low-level details, but they somehow learn to stay within (or close to) the ball park, so that their proof sketches are enough guidance to a well-trained expert to fill in the dots (and correct any typos) **?**.

For this reason, I will intersperse the casual informal set notation with the strict formal presentations of statements and proofs about sets, so you can tell precisely what we mean when we write some informal stuff down: so you can fill in the dots when you need to.

[2]One can argue about whether this is a good idea, just as one can argue whether the X86 instruction set is a good idea. Philosophers and logicians have proposed other foundations for mathematics, and with good reason! But from the early 20th century until now, set theory rules the day, much like the X86 ISA.

[3]There is an important difference between being pre-rigorous, formally rigorous, and post-rigorous, as described aptly by Fields medal winner Terrence Tao `https://terrytao.wordpress.com/career-advice/theres-more-to-mathematics-than-rigour-and-proofs/`

2.4 Sets are about Stuff

At heart, set theory is a way of talking about containers of stuff. *Everything* in set theory boils down to the *"element of"* relation $\in$. If one set A is an element of another set B, then we write $A \in B$. That's really all there is to set theory, but we will build up a lot of stuff on top of that single concept (see, we don't even have 1's and 0's just $\in$). First, we need to know how this $\in$ relationship works...what are the rules that constrain it? These are captured by the *axioms* of a set theory. I'm not going to give *all* of the axioms of set theory, just the ones that we will use regularly. In fact, there are lots of set theories out there, but many have a pretty common core. We'll be using a set of axioms for what is called *Zermelo-Frankel Set Theory* or ZF for short. Technically we'll be using a variant of the axioms called "Intuitistic ZF" a.k.a. IZF.[4] Now let's talk about some of the axioms, which tell us how sets work as well as what kinds of sets are out there, i.e. what sets *exist*.

2.5 When are two sets really the same set?

We start with a basic property of sets:

> A set's identity is entirely determined by its elements.

Think of a set as an invisible bag of stuff...all we identify are the stuff, not the material that the bag is made of. This conception of sets is captured formally by the *Axiom of Extensionality*.[5]:

$$\forall S_1.\forall S_2.S_1 = S_2 \Leftrightarrow \forall S.S \in S_1 \Leftrightarrow S \in S_2.$$

In words, this reads: "for any set expressions S_1 and S_2, they describe the same set, i.e. are *identical*, if and only if every set expression S that describes an element of S_1 also describes an element of S_2 and vice-versa." Remember: the only elements that arise in set theory are sets, so we're just worried about what sets are elements of some other set. For our purposes, this axiom defines identity (a.k.a. equality) of sets $S_1 = S_2$. Despite the use of "the equals sign", I am going to try and use the word *identity* because the idea of there being a particular set "out there" conveys a more specific sense of specifically being "the same thing, with no caveats", compared to other forms of *equivalence*, which may be "the same thing, so long as you ignore some details". First-order logics typically provide rules about identity that codify some "bare minimum" properties that one can rely on when reasoning with multiple expressions that describe "the same" object. We'll discuss those later when we learn about how to reason in our logic. The remaining fundamentals of set equality/identity can be ascribed to this axiom.

Lets briefly discuss some of the logic notation that arises in the formal statement of the axiom. The symbol $\forall$ means "for all", and $\Leftrightarrow$ which means "if and only if".

$$A \Leftrightarrow B := (A \Rightarrow B) \wedge (B \Rightarrow A).$$

The core of our logic does not have $\Leftrightarrow$ as a primitive notion. Rather, $\Leftrightarrow$ is really a syntax abbreviation, much like a *macro* written in the C preprocessor language (CPP). The $:=$ symbol is used to introduce a *notational definitions*.[6] $A := B$ can be read as "Whenever you see the syntax A, you can immediately rewrite it as B in any context." Think of it as analogous to a *macro*, like what you would write in the C Pre-processor. We use notational definitions to encode familiar concepts in terms of simpler ones (thereby keeping our logical language small and precise), often resulting in a more concise or traditional notation, but backed by a rigorous (possibly longer) logical presentation.

Now whenever we see some expression with $\Leftrightarrow$ in it, we should "macro-expand" it away to its constituents. This keeps the core of our logic small but gives us some compact notation for common patterns. T I will often refer to such purely syntactic definitions as *syntactic sugar*. It does not empower you to say anything new, just to encapsulate it in a compact and meaningful notation.

In the definition of $\Leftrightarrow$ the wedge $\wedge$ is the symbol for *conjunction*, as in "and". The arrow $A \Rightarrow B$ is *implication*, as in "if A then B" or "A implies B" or "A only if B" or "B if A": all of these statements are

[4] `https://plato.stanford.edu/entries/set-theory-constructive/axioms-CZF-IZF.html`

[5] Roughly speaking, a set's *extension* is "the stuff that it is made up of."

[6] sometimes you'll see $\equiv$ instead, which I mean to replace with $:=$.

different ways of saying the exact same thing in words. To me the stranges ones are the last two, which take getting used to. But they help explain how "A if and only if B" textually breaks down into the two distinct implications "A only if B" and "A if B". For now don't worry too much about how precisely to read the above logical formula, we'll get into the details of how to understand logical propositions later.

But unpacking the axiom above a little more, we now get: "for any sets S_1 and S_2, we know that: 1) if they are equal, then any element of S_1 is an element of S_2 and any element of S_2 is an element of S_1; and 2) if any element of S_1 is an element of S_2 and vice versa, then S_1 and S_2 are equal/identical, i.e. the same set."

Now we know how to judge that "two" sets are the same set, or more precisely that two set expressions *describe* the same set. As we'll emphasize going forward, the formalism we write down is not actually a set itself, but rather an expression that (maybe) *describes* a set, and that identity is a way of indicating that two expressions describe the same set. This is similar to how "aubergine" and "eggplant" describe the same kind of vegetable, but you should not try to cut out and eat either description! Distinguishing between a set and its many possible descriptions can prevent some misconceptions, so we emphasize this throughout. Since equality/identity is a pretty key idea in logical systems, it's important that we lay this out good and early. Without a notion of equality/identity, we wouldn't be able to justify our ability to refer to a specific set, e.g., "*the* set that has no elements" as we do next.

2.6 The Empty Set

We now know how to determine if two set expressions describe the same set, *but we still don't know if there are any sets!*

Don't let the axiom of extensionality's "for all sets..." claim fool you: logic allows us to make statements like "All flying pigs prefer to drive pickup trucks." which might be a true statement, not because flying pigs don't like bicycles, but rather because we can establish that there are no flying pigs in the first place![7] This is one of the powers and challenges of logic: it empowers us to make deductions based on premises that might not be true, or might be true but we do not yet know.[8] In short, we could still talk about "all sets" even if there were none, so we are on the hook to ensure that there's *at least one* set. Let's do that.

One set that's very important to us is the set with nothing in it: *the empty set*. This leads us to state the *Axiom of the Empty Set*.:

> There is an empty set, i.e. there exists a set with nothing in it.

Using formal notation, this statement looks like the following:

$$\exists S_1.\forall S_2.S_2 \in S_1 \Rightarrow \bot.$$

Here, the logical symbol $\bot$ represents *falsehood*. In our logic, $A \Rightarrow \bot$ is our way of writing "not A." You can read it as "Well if A then pigs fly!" as in, there is no way that A holds: its truth implies *the* impossible (proposition). If you have experience with logic, then this may look a bit odd compared to other presentations of negation, but there is some actual sense to this. In fact, we often abbreviate this kind of negation using another common notation:

$$\neg A := A \Rightarrow \bot.$$

Many presentations of logic use the notation $\neg A$ for negation, but take it as primitive. We will find it very useful to treat "not A" as a non-primitive logical form, just like we treat "if and only if". Reasoning about negation reduces to reasoning about implication and reasoning about falsehood (yup, it's a thing!), which can make proofs that involve negation more mechanical and less mysterious, and can also clarify the distinction between reasoning about falsehood (which we must do) and proof by contradiction (which we will never have need for). These two distinct practices are often confused for one another unfortunately [**?**].

Also, we often abbreviate the idea that some relation doesn't hold by crossing it out. For example,

$$A \notin B := A \in B \Rightarrow \bot$$

[7] Barring recent advances in biotechnoogy of which I am unaware.

[8] For example, many results in algorithmic complexity theory are premised on assumptions like $P \neq NP$, even though we don't know, and may never know, if that is true.

and

$$X \neq Y := X = Y \Rightarrow \bot.$$

Keep these notational definitions in mind: they will help you prove theorems about non-elementhood and inequality later.

Now, we can combine the axiom of the empty set with the axiom of extensionality to prove that the empty set is unique, i.e., there is at least one and at most one set that is empty.

Proposition 6 (The Empty Set is Unique)**.**

$$(\exists S_1.\forall S_2.S_2 \in S_1 \Rightarrow \bot) \wedge \Big(\forall S_1, S_2.\,(\forall S.S \in S_1 \Rightarrow \bot) \wedge (\forall S.S \in S_2 \Rightarrow \bot) \Rightarrow S_1 = S_2\Big).$$

Reading the above formula more literally, it reads "there exists a set that has no elements; and furthermore should you find yourself with two sets that have no elements, then they are actually the same set." In other words, there's at least one empty set and at most one empty set, i.e. there's exactly one! This is why we get to call it *the* empty set, rather than *an* empty set. Thus the concept of emptiness suffices to *definitely describe* a particular set. *Definite description*—the presentation of sufficient conditions to identify a particular set—will be one of our central activities, and much of the machinery we study provides more and more sophisticated means to definitely describe certain sets. One motto of ours is: "the word *define* is short for *definitely describe* and *definition* is short for *definite description*." We introduce a formal notation for definite descriptions shortly.

Let's use notational definitions to clarify the formal statement of the Axiom of The Empty Set: the textual statement is simple but the primitive formal statement is a hot mess! We can simplify the logical formula using a couple of abbreviations. First, as described above we use $\notin$ to mean "not an element of" Second, we codify the concept of *unique existence*, a.k.a. *uniqueness*, using a notational definition. Let Φ be some property of sets. Then we define the $\exists!$ quantifier to express uniqueness:

$$\exists!S.\Phi(S) := \exists S.\Phi(S) \wedge \Big(\forall S_1, S_2.\,\Phi(S_1) \wedge \Phi(S_2) \Rightarrow S_1 = S_2\Big).$$

In the above notation, $\Phi(S)$ means that the logical formula $\Phi(S)$ may contain unquantified references to the set variable S. Then $\Phi(S_1)$ denotes the same formula as $\Phi(S)$ except replacing some unquantified references to S with S_1. This notation is often intends to replace *all* references to S in $\Phi(S)$, but in some contexts that is not necessary. Since total replacement is the default, we'll point out those cases where only some—including none!—references might be replaced.

This notational definition emphasizes the two distinct reasoning powers that uniqueness gives you: first that something with the property $\Phi(S)$ is out there (in case you need it), and second that any two set expressions that satisfy this property must actually describe the same set, so you can use logical equality rules to reason about that sameness. Reasoning with equality is an important and powerful tool.

Using these two notational definitions, we can succinctly rewrite Prop. 6 as:

$$\exists!S_1.\forall S_2.S_2 \notin S_1$$

In short: "There's a unique set that has no elements." This says *exactly the same thing* as the original proposition, since this "macro-expands" to it. Bear in mind that we have not proven this proposition yet, but we'll see that it follows from the axiom of extensionality and the axiom of the empty set.

Now that we know that there is only one empty set, we can use a definite description to name it.

$$\imath S_1.\forall S_2.S_2 \notin S_1$$

This form of expression, $\imath S.\Phi(S)$, where Φ is some proposition that (probably) refers to S is what we call a *definite description* [**?**]. This particular expression can be read "*the* unique set S_1 such that no set S_2 is an element of it." By replacing the $\exists!$ with an $\imath$, we've moved from *stating a proposition* about to a unique set to *referring to a set* by stating a condition that uniquely describes it.

A definite description is a way of...well...describing a set by stating some property $\Phi(S)$ of a set S that uniquely characterizes the set. As such, a definite description is a way of "naming" a set, kind of like "that

guy who sits in the back of class every day snapping his gum." Ideally such a description suffices to uniquely identify the set in question, in which case the description is called *proper*. If there are three guys snapping gum, then this description is *indefinite*, and if no guys snap gum then the description is *empty*. We should strive for proper definite descriptions, and avoid the latter two.

One nice thing about definite descriptions is that they not only identify a set, but do so by enunciating a property that the set in question (uniquely) satisfies. One built-in rule of our logic is that for any set S that can be identified by a proper definite description $\imath S_1.\Phi(S_1)$, we can immediately deduce that $\Phi(S)$ holds, i.e. $\Phi(\imath S_1.\Phi(S_1))$ is true. As you should expect, the unique set S that satisfies Φ...satisfies Φ! But don't forget that this only makes logical sense if wecan prove that Φ describes a unique set, thereby establishing that the definite description is proper.

For conciseness and tradition, we use a notational definition to ntroduce a common name for the empty set:

$$\emptyset := \imath S_1.\forall S_2.S_2 \notin S_1$$

It's no coincidence that the definite description of the empty set looks *exactly like* the statement of uniqueness, but replaces $\exists!$ with $\imath$. We've transitioned from establishing that "there exists a unique set S such that ..." to referring directly to "the unique set S such that...".

Definite descriptions are not strictly necessary for logic: we can "compile" propositions that use definite descriptions to equivalent propositions that don't. But definite descriptions more directly and precisely capture how we casually talk about mathematical objects. We are prone to make statements that directly refer to "the empty set", and reason just based on that name. Definite descriptions, and the reasoning principles they embody, mirror our informal speech. In addition to improving the mapping from prose to formality, definite descriptions clarify the relationship between definitions and reasoning principles: the definitions directly imply some principles, which we can immediately extract from the structure of the description. For example, the fundamental reasoning principle for the empty set can be deduced directly from its definite description:

$$\forall S.S \notin \emptyset.$$

We'll find that the concepts underlying definite descriptions, and the reasoning principles that they entail, scale from our lowest-levels of set theory to our high-level strategies for defining components of a programming language semantics. One could also imagine teaching a computer to recognize and manipulate them on our behalf since they are so systematic.

2.6.1 Uniqueness, Shmuniqueness!

You may be annoyed that we are still on the hook to prove that the empty set is unique. Why don't our axioms just make the uniqueness of the empty set part of the axiom of the empty set and be done with it? We could have done that, but we didn't. Our axiom of the empty set does not establish uniqueness because it does not need to: the axiom of extensionality could be used to readily establish uniqueness, so why add more complexity and redundancy to one of your axioms?

Furthermore, by relegating all reasoning about set equality/identity to the axiom of extensionality, we are left with a clear and unambiguous understanding of what constitutes set equality. If we assert uniqueness in some other axiom, then expanding the notational definition for $\exists!$ shows that such an axiom also makes claims about equality, and might accidentally introduce unintended additional properties to set equality.

These principles reflect how mathematicians tend to think in my experience: keep the *number* of axioms you assert small, keep the *strength* of each axiom as weak as possible, and try to make each axiom as *independent* from one another as possible, such that each is necessary and none implies the others. Building up a foundational theory this way keeps axioms manageable and ensures that you can get your work done with as small of a *trusted base* as possible. Mathematicians have argued for yeeeeears about various axioms: which ones are self-evident, which ones are questionable, which ones are necessary for a theory to be useful, and which ones imply really strange things but we can's seem to do without them. This is not dissimilar to an aesthetic that drives some programming language designs: the desire to provide as few mechanisms as possible (e.g. "everything's an object" or "everything's a function") and to provide only "one way to do it" whatever "it" may be.

Now this approach to choosing axioms (and constructing programming languages) *does not* make your working set of properties particularly ergonomic, but if the language provides some mechanism for building your own abstractions (like our notational definitions), then you can build up higher-level human-scale tools (i.e. propositions) on top of them with confidence and then use those in your day-to-day work. By having a firm, clear foundation, you need not worry that your mathematical castles rest on sand, or that unnecessary inconsistencies may arise accidentally. But it goes to show that these systems were built to reason *about* set theory, not to do real work in it!

2.7 Beggars Can Be Choosers

Some axioms of our set theory empower us to identify bigger sets in terms of smaller sets. We discuss them later. However, those axioms are rather blunt instruments, which give us rather coarse grained mechanisms for building sets. It's like having a dump truck full of clay as your primary means of practicing pottery: you can big clumpy piles, but not fine dinnerware. Much of the time we operate by throwing a clumpy pile down and then refining it to a precise form using precise tools. We do this by using the full power of first-order logic to carefully describe *smaller* subsets of larger ones by selecting desirable elements. The tool that we use for this selection process is called the *Axiom Schema of Specification* (also known as the *Axiom Schema of Separation*)

> Let $\Phi(X)$ be some logical predicate on sets X and let A be some set. Then there is a set S of all and only the elements $B \in A$ such that $\Phi(B)$ holds.

i.e. for every predicate $\Phi(X)$ we have the axiom:

$$\forall A.\exists S.\forall B.B \in S \Leftrightarrow B \in A \wedge \Phi(B).$$

This one is usually called an axiom *schema* because it is taken to stand for an infinite number of axioms: one for every Φ that you can come up with.

Notice how this axiom schema is phrased as an "if-and-only if", a.k.a. a *bi-implication*, so we can pick out a set whose contents *exactly* match the result of filtering. This axiom is the one that we can use to winnow things down to exact sets. With help from the axiom of extensionality, we can prove that for any A and Φ, the resulting specification describes a unique set. This uniqueness justifies a new notation for describing sets: a *set comprehension*:[9]

$$\{\, B \in A \mid \Phi(B) \,\} := \iota S.\forall B.B \in S \Leftrightarrow B \in A \wedge \Phi(B).$$

The notation $\{\, B \in A \mid \Phi(B) \,\}$ describes the unique set characterized by being all and only the B's in A for which $\Phi(B)$ holds. Its underlying definite description immediately provide the following reasoning principle:

$$C \in \{\, B \in A \mid \Phi(B) \,\} \Leftrightarrow C \in A \wedge \Phi(C).$$

Anti-Patterns in Notation Many texts that introduce set theory introduce a quite similar notation to the above, similar but different in a frought way:

$$\{\, B \mid \Phi(B) \,\}$$

We will steer clear of this notation! The main reason for doing so is that each of the set builder notations we introduce can be justified by a unique existence proof tied intimately to one of the axioms of our set theory. The above notation cannot. It corresponds to a proposition that was once proposed as an axiom of set theory called *unrestricted comprehension*:

$$\exists S.\forall B.B \in S \Leftrightarrow \Phi(B).$$

This proposition essentially says "every proposition determines a set." Unfortunately this axiom is a great way to ruin a set theory. We can use it to prove $\bot$—falsehood!—without any extra premises, which renders

[9] Set comprehensions inspired a PL feature called *list comprehensions* that appears in Python, Haskell, and other languages.

the theory useless. A logic that can prove falsehood is called *inconsistent*, and in most situations this is a death sentence. In this particular case, The logician Bertrand Russell (among others, even before him) demonstrated how to prove $\bot$, a technique now named *Russell's paradox*. It involves setting $\Phi(B) \equiv B \notin B$, yielding the set S of all sets B which are not elements of themselves. Existence of this set can be used to prove $\bot$.

With careful choice of Φ, the unrestricted notation can be used to properly define sets, but a more rigorously motivated notation can obviate the need for such care. It's far less error-prone to appeal to robust description mechanisms that are aligned with the axioms of the theory. As it happens, definitions guided by the axiom schema of separation do not suffice for all uses of set theory, but this is because the axiom schema of separation alone does not suffice. IZF includes another related axiom that fills the gap, and we can introduce another notational definition that corresponds directly to it (See Sec. 2.13). To quote Yaron Minsky, a proponent of statically-typed functional programming:[10]

> "Make illegal states unrepresentable."

2.8 Sets in Sets in Sets

So far we know what it means for two sets to be equal, we know that there's an empty set, and we know how to use predicates to curate an artisinal subset of bespoke sundries. Is that enough for us to introduce more friends for the empty set? Not so far! Why not?!? Well, all we have right now are the empty set and set comprehensions, and applying the latter to filter the former gives us back the empty set again. We're still stuck with just one set.

Our next axiom gets us some more sets, in a somewhat simple way. The *Axiom of Pairing*:

> For any sets A and B, there is a set C that contains exactly A and B as elements.

Formally,

$$\forall A.\, \forall B.\, \exists C.\, \forall D.\, D \in C \Leftrightarrow D = A \vee D = B.$$

The $\vee$ symbol represents *inclusive* "or": a formula $A \vee B$ is true if and only if A is true, B is true, *or both A and B are true*. This use of inclusive-or, common in logic, can be confusing since in informal speech we often say "or" when we mean "either A or B but not both".

This axiom is just a bit obnoxious because it tells us that there's a set with A and B as members, but it does not outright say that the set is unique. There may be tons of other sets with this property! However, just like we dealt with the axiom of the empty set, we can use the axiom of extensionality to prove that any two sets with this property are identical. Thus, this set not only exists, but is unique. Uniqueness justifies our introduction of some possibly familiar notation:

$$\{\, A, B \,\} \equiv \imath S.\, \forall Q.\, Q \in S \Leftrightarrow Q = A \vee Q = B.$$

and a reasoning principle that we can read right off of it:

$$Q \in \{\, A, B \,\} \Leftrightarrow Q = A \vee Q = B.$$

It's a little funny that our axiom, and our new notation, seem to be just for two-element sets. As we see next, that's only because this axiom, along with others, suffices to justify bigger and smaller finite sets.

2.8.1 From Two Elements to One Element to Many Elements

Now that we have the empty set and justification to describe sets $\{\, A, B \,\}$ given any pre-existing sets A and B, we can apply the axiom of pairing, with $\emptyset$ for A and $\emptyset$ *again* for B to prove that there exists a set S such that $Q \in S$ if and only if $Q = \emptyset$ ... or $Q = \emptyset$? If that sounds redundant to you, that's because it is. Formally, we can prove that $Q = \emptyset \vee Q = \emptyset \Leftrightarrow Q = \emptyset$. So it's safe to say that the set $\{\, \emptyset, \emptyset \,\}$ has only one element: $Q \in \{\, \emptyset, \emptyset \,\} \Leftrightarrow Q = \emptyset$ This kind of reasoning can be applied schematically to *any* pre-existing set

[10] `https://blog.janestreet.com/effective-ml-revisited/`

A, and it's a little odd to use the notation $\{ A, A \}$ to describe such *singleton* sets, so we introduce better suggestive notation:

$$\{ A \} \equiv \imath S. \forall Q. Q \in S \Leftrightarrow Q = A.$$

and reasoning principle:

$$Q \in \{ A \} \Leftrightarrow Q = A.$$

Our reasoning above implies that $\{ A, A \} = \{ A \}$: both expressions describe the same set! Technically we have to (and can) prove this, by showing that each of the two sets contain the same elements:

$$\forall R. \forall S. (\forall Q. Q \in R \Leftrightarrow Q = A \vee Q = A) \wedge (\forall Q. Q \in S \Leftrightarrow Q = A) \Rightarrow \forall Q. Q \in R \Leftrightarrow Q \in S$$

and then break out extensionality. BAM!

This is our first new set builder notation that is not *immediately* tied to a single axiom, but don't pay too much mind to that: a nice thing about logic is that we can incrementally build up a richer, more human-friendly reasoning principles and corresponding notation from a few bare parts.

At this point we we can construct a set with *zero* elements, sets with *one* element, and sets with *two* elements. What about three, four, five, and the rest? Not yet: we need more axioms! However, we can repeat our singleton and coupleton constructions to form deeply nested sets of couples of singletons of couples, like $\{ \{ \emptyset \} \}$, $\{ \{ \{ \emptyset \} \}, \emptyset \}$ and so on. That's not enough, but it's not *nothing*!

2.9 Come Together, Right Now, Over Me

Another useful and well-known approach to construct a set that combines other sets is to take the *union* of a collection of sets, i.e., a set of sets. The *Axiom of Union* supports this:

> For every set S there is a set G that contains as elements the contents of every set that S contains.

i.e.,

$$\forall S. \exists G. \forall A. A \in G \Leftrightarrow \exists B. (A \in B \wedge B \in S).$$

To make sense of the formal notation, let's consider an example. Let $S = \{ \{ 1, 2 \}, \{ 3, 4 \} \}$. Then from the axiom, we know that there is some set G such that each of the given numbers is an element of G, i.e., $1 \in G, 2 \in G, 3 \in G$, and $4 \in G$. You can look at this as "flattening" one layer of set-ness, removing inner braces in this particular case. The axiom schema of separation cannot perform this flattening because it can only build a set that has some of the same members as another set, like $\{ \{ 1, 2 \} \}$ or $\{ \{ 3, 4 \} \}$.

Following our pattern from above, we can use the axiom of extensionality to prove that sets justified by the axiom of union are unique. Uniqueness ushers in the traditional notation for set union:

$$\bigcup S := \imath G. \forall A. A \in G \Leftrightarrow \exists B. A \in B \wedge B \in S$$

We can then use general union and pairing to introduce the common notation for binary set unions:

$$A \cup B := \bigcup \{ A, B \}$$

This is our first instance of introducing a notational definition in terms of another notational definition. This layering lets us build new abstractions in terms of old abstractions, and we can understand their meaning by repeated desugaring until there is no sugar.

Why define the binary union in terms of the big one, and not the other way around? Well, because we can't form the union of an infinite number of sets by doing it two at a time: we'll get bored eventually. However, if we have a set with an infinite number of things (see below!), then we can form an infinite union whiz bang! This is why the axiom of union is not phrased in terms of two sets, like the axiom of pairing is. You might find this asymmetry between union and pairing a bit off-putting (why not make both infinitary or both binary?), but my uninformed guess is that mistrust of infinities during the early development of set theory motivated Zermelo's choice of axioms. There's nothing inevitable about this design: it's historically contingent.

As usual, the definite description immediately induces a reasoning principle for sets formed using union.

$$A \in \bigcup S \Leftrightarrow \exists G. A \in G \wedge G \in S.$$

Thus we've turned the axiom of union into a reasoning principle about *the* union of a family of sets.

2.10 Extensional Set Definitions

With the introduction of the Axiom of Union, we finally have enough principles to justify the construction of sets with an arbitrary *finite* number of elements, not just zero, one, and two! The name of the axiom of extensionality leads us to the set-builder notation that I call an *extensional set definition*. In informal practice with sets, when you want to describe a set with a finite number of elements, you can just enumerate a list of elements as a way to refer to the set. For example, any recent text that introduces some set theory or sets will introduce expressions like:

$$\{\,\text{cheese}, \text{crackers}, \text{vegemite}\,\}$$

to describe the *unique* set that contains exactly three food products: cheese, crackers, and vegemite. The notation above, in which we describe a set by enumerating a list of descriptions of elements, is so common that if you've seen it before, you were probably told to consider it a primitive syntactic construct. We are not going to take it as primitive. Instead we'll introduce it in two steps. First, we introduce a notational definition:

$$\{\,A, B, \ldots, C\,\} := \imath S_1.\, \forall S_2.\, S_2 \in S_1 \Leftrightarrow S_2 = A \vee S_2 = B \vee \ldots \vee S_2 = C.$$

This means that we expand the above expression describing food items into the complex logical expression:

$$\imath S_1.\, \forall S_2.\, S_2 \in S_1 \Leftrightarrow S_2 = \text{cheese} \vee S_2 = \text{crackers} \vee S_2 = \text{vegemite}.$$

This particular description can be read as "the (unique) set S_1 that can be described by the property that any set S_2 is an element of S_1 if and only if S_1 is identical to the set described by cheese, the set described by crackers, or the set described by vegemite."

We're still on the hook to prove that such definitions are proper. To do so, we must prove a theorem of the form:

Proposition 7 (Extensional Descriptions are Unique)**.** *Let $S_1, S_2, \ldots, S_n$ be sets. Then there is a unique set Q such that for all sets R, $R \in Q$ if and only if $R = S_1$, $R = S_2$, …, or $R = S_n$.*[11]

Once we've proven that extensional set definitions in general are proper descriptions, then we can justifiably apply the defining property. Here is the property induced immediately by our description of the set of food products.

$$S \in \{\,\text{cheese}, \text{crackers}, \text{vegemite}\,\} \Leftrightarrow S = \text{cheese} \vee S = \text{crackers} \vee S = \text{vegemite}.$$

Typical presentations of sets take this kind of element-hood property for granted. For us, this property is derived from expanding the set notation into a definite description that is written in terms of logical operations and elementhood.

2.10.1 Extensional set definitions from the ground up

So now we have enough machinery to justify extensional set definitions. Given sets a,b, and c, do how we know that the set $\{\,a, b, c\,\}$ exists? In pedantic form the statement is:

$$\exists! A. \forall S. S \in A \Leftrightarrow S = a \vee S = b \vee S = c,$$

One way to justify its unique existence is as follows:

1. use the axiom of pairing to form $\{\,a, b\,\}$;
2. use the axiom of pairing to form $\{\,c\,\}$;
3. use the axiom of pairing to form $\{\,\{\,a, b\,\}, \{\,c\,\}\,\}$;
4. use the axiom of union to form $\{\,a, b, c\,\}$.

[11] Formally $\exists! Q. \forall R. R \in Q \Leftrightarrow R = S_1 \vee R = S_2 \vee \cdots \vee R = S_n$. We'll explain the notation $\exists!$ soon.

Well that was exhausting! But we could further extend this set by constructing $\{ d \}$, pairing it with our present set, and applying union again. Singleton, pair, union; singleton, pairing, union: wash, rinse, and repeat as needed.

We will never bother performing this construction again. But note, that we have a reasoning principle for our set based simply on its description:

$$\forall S. S \in \{ a, b, c \} \Leftrightarrow S = a \vee S = b \vee S = c,$$

Recall that such reasoning also makes sense for a set name like $\{ a, a \}$, though it's mighty redundant: we like the name $\{ a \}$ better, partly because the reasoning principle that we get for the first is annoyingly redundant compared to the second. However, later we see cases where having multiple different definitions of the same set can give us *usefully* different reasoning principles, derived from different definite descriptions of the same set.

In this particular case, the redundancy of extensional definitions is a cause for care particularly when making an abstract argument. For instance, if we assume some sets a and b, and then mention "the set $\{ a, b \}$", we cannot be sure that the set $\{ a, b \}$ has two elements: what if $a = 5$ and $b = 5$? On the other hand, if we know a priori that $a \neq b$, then the set $\{ a, b \}$ definitely has two elements. Notice, though, that in both cases it's true that $a \in \{ a, b \}$ and $b \in \{ a, b \}$.

At this point, we are justified in treating extensional set definitions with *any finite number of elements* as though they were primitives of our set theory. It doesn't matter how we built them, but it *does* matter what reasoning principles we associate with them based solely on the definite description that underlies their notational definitions. This is our first substantial layer of set-theoretic abstraction!

2.10.2 Tool Building: A General Principle of Cases

The reasoning principle induced by extensional set definitions is quite useful, in that it suffices for reasoning about the elements of sets that are defined using it. But rather than using it directly all of the time, it can be nice to develop a reasoning principle that is phrased in terms of *propositions* $\Phi(S)$ that describe potential properties of the set's elements rather than in terms of the *elements* S themselves. For instance, we will often want to prove that every element of a set satisfies some complex property. Having a principle that explains how to do this generally can streamline proofs, though we must put in some work ahead of time to prove the new principle in terms of the induced one.

In particular, we can use the definite description of extensional sets to state and prove the following principle.

Proposition 8 (Principle of Cases on $\{ S_1, S_2, \ldots, S_n \}$)**.** *Let Φ be a property of sets and $S_1, S_2, \ldots, S_n$ be sets. Then $\Phi(S)$ holds for all $S \in \{ S_1, S_2, \ldots, S_n \}$ if $\Phi(S_1)$, $\Phi(S_2)$, …, and $\Phi(S_n)$.*[12]

We will be equipped to prove this proposition once we study the proof techniques that apply to our logical operations. This property may seem quite natural and intuitive, so one may be surprised how much work it takes to prove this rigorously. But think about it: when programming, how many easy-to-understand procedures do you know of where what you want seems straightforward, and it's obvious that it can be implemented, but the finicky details you had to go through to get it done right inspire you to search the internet for the code or (gasp!) use the standard library implementation? Our proof of this is like that. For now let's take it as given.

This is an example of *using our tools to build better tools*, much like one can use the primitives of a programming language to design powerful and productive libraries of routines.

Right from the start, we see an incident of a formal definition (extensional sets) affecting our reasoning principles: an extensional definition of a set licenses you to reason about its elements by cases on its (finite) members, and lets us build a general principle for proving properties of a sets elements. When we get to defining sets that contain an infinitude of elements, we'll need new tools. Luckily our definitions of those sets will help with that.

[12] Formally, $\Phi(S_1) \wedge \Phi(S_2) \wedge \cdots \wedge \Phi(S_n) \Rightarrow \forall S \in \{ S_1, S_2, \ldots, S_n \}. \Phi(S)$. One seeming mismatch of textual vs. formal statements of propositions is that in text the "for all" statement often comes *after* the proposition to which it applies, I suspect to draw the reader's attention to the heart of the statement rather than the quantifiers.

For example, consider the set $X := \{ 7, 9, 11 \}$. We might want to prove that doubling each element yields an even number:

$$\forall n. n \in X \Rightarrow even(n + n).$$

How to prove it? Well, set $\Phi(n) := even(n + n)$, and apply our principle of cases on X, which involves proving $\Phi(7)$, $\Phi(9)$, and $\Phi(11)$.

2.11 Infinite stuff!

Given the tools we have so far, we can only build finite sets! That's because all of our logical arguments to establish the existence of sets can only involve a finite number of steps, and each of those steps so far has used a finite number of finite sets to describe some other finite set. This insistence on finite-length arguments can really be a pain sometimes, but that's how (most) logicians roll. This is also how computer programmers roll: ever seen an infinite-length program?

So how do we get any infinite sets? Well, like we did with the empty set, we *assume one*, via the *Axiom of Infinity*:

> There's a set that (1) contains the empty set and (2) contains the set $a \cup \{ a \}$ whenever it contains a.

i.e.,

$$\exists S. (\emptyset \in S) \wedge (\forall A. A \in S \Rightarrow A \cup \{ A \} \in S).$$

Notice how I used $A \cup \{ A \}$ in the above formula, rather than spelling the whole thing out in terms of $\in$, $\exists$, uniqueness, etc. That would be way too painful! Luckily we can already exploit our paucity of abstractions to make life more pleasant. You may as an exercise try to boil this all the way down to just statements about elementhood.

Set theorists like to call the set that has *only* the elements above ω, and they use it to represent the set of natural numbers:[13]

$$\begin{aligned} 0 &\equiv \emptyset \\ 1 &\equiv \{ 0 \} \cup 0 = \{ \emptyset \} \\ 2 &\equiv \{ 1 \} \cup 1 = \{ \{ \emptyset \}, \emptyset \} \\ &\vdots \end{aligned}$$

First, notice the leaky abstraction: when was the last time you took the union of a set with a number??? Let's not do that any more! instead, we will just consider the nice abstract set of *natural numbers* as the set,

$$\mathbb{N} = \{ 0, 1, 2, \dots \},$$

and say that questions like $1 \in 2$ are off limits...that would be exploiting the *representation* of numbers as sets, rather than the properties of numbers *as an abstraction* that we care about. Notice that 0 is a natural number. All right-thinking computer scientists take this as a given.[14]

2.12 All the Subsets

Now for another axiom that lets us build bigger sets from existing sets: the *Axiom of Power Set*:

> Given any set A, there exists a set that contains all subsets of A.

[13] This encoding is due to the great mathematician John von Neumann.

[14] I once went to a wedding of two computer scientists, and during dinner they sat at "Table 0" because, as the table manifest said: "computer scientists start counting from zero."

i.e.,

$$\forall A.\exists S.\forall B.B \in S \Leftrightarrow (\forall C.C \in B \Rightarrow C \in A).$$

Good grapes, just looking at that makes my eyes bleed! In the above formalism, B is a subset of A because every element C of B is also an element of A. Because we use subsets so often, there's a convenient notation for them:

$$X \subseteq Y := \forall C.C \in X \Rightarrow C \in Y.$$

With this we could have written:

$$\forall A.\exists S.\,\forall B.\,B \in S \Leftrightarrow B \subseteq A.$$

Once again, this axiom doesn't say that the set with this property is unique, but we can whack it with the axiom of extensionality to establish uniqueness. We call this set the *powerset* of A, and give it a notation:

$$\mathcal{P}(A) := \imath S.\,B \in S \Leftrightarrow \forall B.\,B \subseteq A.$$

Some books use the notation 2^A, which sort of explains the name "powerset" (2 to the power of A): the reason for that notation is that if some set A has n elements, then $\mathcal{P}(A)$ has 2^n elements.

The uniqueness and description of the powerset give us another reasoning principle:

$$X \in \mathcal{P}(A) \Leftrightarrow X \subseteq A.$$

2.13 Bait and Switch

With the axioms and principles introduced so far, we can get very far, using powerset and union to build up sets, and specification to hone them down. But sometimes, surprisingly rarely, we need to describe a set by relating it directly to the properties of another set. It was only after some years of working with Zermelo's axioms that logicians realized that the axioms outlined above were not sufficient for this task. As a result, Abraham Frankael introduced a new defining principle, which Thoralf Skolem refined into a new formal axiom schema. This contribution is the reason that our set theory is referred to as Zermelo-*Frankael*.[15] It is called the *Axiom Schema of Replacement*:

Suppose $\Phi(A, B)$ is a proposition, and S is a set, and for every set $A \in S$, there is a unique set B such that $\Phi(A, B)$. Then there is a set Q whose elements are exactly those B such that $\Phi(A, B)$ for some $A \in S$. i.e.,

$$\forall S.\,(\forall A \in S.\,\exists! B.\,\Phi(A, B)) \Rightarrow \exists Q.\,\forall B.\,B \in Q \Leftrightarrow \exists A \in S.\,\Phi(A, B).$$

Let's unpack this a bit. The set S is one that we constructed previously...somehow. The goal is to describe a new set Q by relating each element A of S to exactly one element B of Q, allowing for the possibility that multiple elements of S may correspond to the same element of Q. The proposition $\Phi(A, B)$ captures the relationship between elements, and the axiom allows any proposition Φ that quite strictly acts in a "function-like" manner.

With a little more reasoning, incorporating the axiom of extensionality, we can prove that a specific S and appropriate Φ together induce a unique Q, which gives us license, *under assumption*, to introduce a new definite description and corresponding notation:

$$\{\, B \leftarrow \Phi(A, B) \textbf{ for } A \in S \,\} := \imath Q.\,\forall B.\,B \in Q \Leftrightarrow \exists A \in S.\,\Phi(A, B).$$

This particular notation and its definite description are *contingent* on being able to prove that $\Phi(A, B)$ is indeed "function-like". If Φ does not satisfy that property, then the axiom cannot be applied to prove that Q exists, let alone being unique. So unlike our previous set builder notations, this *set replacement* notation is not even meaningful without establishing a property of Φ. This is somewhat unfortunate, because of the extra care required, but it is useful, and we will run into this issue repeatedly when describing domain-specific sets that come with conditions.

[15] Poor Skolem!

An example of set replacement from arithmetic might be the set of all multiples of 7:

$$\{\, m \leftarrow m = 7 * n \textbf{ for } n \in \mathbb{Z} \,\}$$

Here, S is the set of integers, $\mathbb{Z}$, and $\Phi(n, m)$ is $m = 7 * n$. This particular case, where the proposition $\Phi(A, B)$ says that B is equal to some expression involving A, is sufficiently common that we introduce a more compact and direct notation for it:

$$\{\, F(A) \textbf{ for } A \in S \,\} := \{\, B \leftarrow B = F(A) \textbf{ for } A \in S \,\},$$

So the multiples of 7 can be written

$$\{\, 7 * n \textbf{ for } n \in \mathbb{Z} \,\}.$$

The general replacement notation is then reserved for those descriptions that benefit from its full expressive power. But remember that in contrast to our previous set description notations, this one is only proper if the prerequisite $\forall S. \forall A \in S. \exists! B. \Phi(A, B)$ holds.

Since our set replacement notation requires some pre-existing set S and some "function-like" proposition Φ, it does not allow us to express Russell's paradox.

2.14 Sets May Not Eat Themselves!

Based on our reasoning principles, we may find ourselves constructing some set Q, and then subsequently constructing a set R such that $Q \in R$. Can we ever construct such a set like this where $R \in Q$ also holds? Nothing we have said so far precludes this possibility. However, our set theory does have an axiom, called the *Axiom of $\in$-Induction* whose express purpose is to forbid sets from containing themselves.[16] So ultimately Q and R cannot be the same set. Since we do not need to explicitly use the axiom of foundation, we discuss it no further. But recognizing that we have to explicitly disallow cyclic sets if we do not want them can give some more insight about logic.[17]

2.14.1 Coda

This concludes the axioms of set theory that we focus on. IZF has more axioms, but they are mostly geared toward details we will barely touch on, if at all, so we can ignore them.

You may want to mess around with the axioms that we've covered so far to try and build various sets and see what a pain it is to hack in machine code.

Note that each mechanism that we have for defining/naming sets comes with its own reasoning principle. This is a recurring theme in mathematics and in this course in particular: *the structure of your definitions determines the structure of your reasoning.* Often we can describe reasoning principles directly in terms of some definition for picking out a set of interest.

Next, we'll consider some of the pragmatics of using set theory.

2.15 The Practice of Working with Sets

In this section, I'll describe how we actually informally work with sets. Some of this rehashes concepts that we covered above, but ideally this version is written in a more high-level, actionable way, corresponding to how we use set theory in real practice.

[16] In Zermelo's original set theory, this property is introduced using the Axiom of Foundation. The Axiom of $\in$-Induction yields the same effect without compromising constructivity [**?**]. In a non-constructive set theory, the two are equivalent.

[17] This design is a decision! There exist set theories that replace the axiom of foundation with an *axiom of anti-foundation* for the express purpose of allowing sets to contain themselves. Wild!

2.15.1 Assumed Sets

From time to time we will simply *assume* the existence of a set.

It will look like:

> Suppose $a \in \textsc{Atom}$...

What we mean by this is that there must be some set with an infinite number of things in it that we can tell apart, and distinguish from the elements of other sets. So I can always pull out some a_i and then grab some other $a_j \neq a_i$. When we want to talk about specific members, we usually use the same letter as the metavariable but typeset it in blue, as in $\mathtt{a}_1$. In the case of program variables, like $x \in \textsc{Var}$, we may be more creative with our choice of elements.

We can justify this because we already know that there is a set with an infinite number of things, and one of the nice/crazy things about such a set is that you can take an infinite number of things out of that set and still have *another* infinite amount of things left over. For example, if I take the odd numbers out of the set of natural numbers, how many numbers are left over? An infinite amount: the even numbers! My head explodes when I think about this kind of thing.[18]

In practice, we often need to use a finite number of elements of such a set. To do so we appeal to what is sometimes called the *cofinite* principle of the set, where cofinite means the complement of a finite set:

Proposition 9. *If $X \subseteq \textsc{Atom}$ is finite, then there is some $a \in \mathit{Atom}$ such that $a \notin X$.*

this property lets us assert the existence of a new "fresh" atom a, distinct from the others we have considered so far. If we want another fresh atom later, we can apply the same principle, now using $X \cup \{\, a \,\}$ instead of X, which is also a finite set.

2.16 Other Common Set formations

In addition to the above means of forming sets, backed by axioms and further reasoning, we can explain a number of other common set descriptions, typically in terms of the earlier ones. We're not trying to be super-primitive, or make sure that there's only one way to build a set. Rather, we just want to connect common set construction notations to the ones we have already introduced.

2.16.1 Intersection

In addition to forming a set by grabbing element that appear in *any* of a given set of sets, we can also consider the set of elements that appear in *every* set in a given set. That's the intersection. If A and B are sets, then there is a set we call $A \cap B$ with the property that $c \in A \cap B$ if and only if $c \in A$ and $c \in B$.

Example 1. If $A = \{\, \mathsf{paper}, \mathsf{cut} \,\}$ and $B = \{\, \mathsf{news}, \mathsf{paper} \,\}$ then $A \cap B = \{\, \mathsf{paper} \,\}$.

This operation can be defined in terms of set comprehension:

$$A \cap B := \{\, a \in A \mid a \in B \,\}.$$

In this case, the predicate $\Phi(a) \equiv a \in B$. Notice that you cannot use set comprehension to describe the *union* of two sets without already having something that contains the union. That's why we have an axiom for union, but not one for comprehension.

Generalizing this idea, if F is a *non-empty* set of sets, then we can form the set $\bigcap F$, which is the intersection of all the sets in F: $c \in \bigcap F$ if and only if $C \in F$, and $c \in C$ for *every* set $C \in F$. Analogous to unions, $A \cap B = \bigcap \{\, A, B \,\}$, but in stark contrast, there is no such set $\bigcap \emptyset$! This means that when you are reasoning about sets, you must establish that F is not empty before you take its intersection. In the case of both intersection and union, you must establish that F contains only sets before you take their union, otherwise it's an "abstraction error".[19]

[18] In the 19th century, the mathematician Richard Dedekind proposed roughly the above property as the defining characteristic of an infinite set.

[19] By "abstraction error" I mean that technically it's fine to take the union, because at the lowest level all we have are sets, but you would surely be violating an abstraction principle. For example, $\bigcup 1$ is nonsensical for numbers, but not for their von Neumann representation where $\bigcup 1 \equiv \bigcup \{\, \emptyset \,\} = \emptyset \equiv 0$. Yuk!

2.16.2 Set Difference

Another example, which we are introducing for the first time here, is the notion of *set difference*:

$$A \setminus B := \{\, a \in A \mid a \notin B \,\}.$$

Notice that set difference is always well defined for any two sets A and B.

2.16.3 Sequences and Products

In addition to sets, unordered collections of unique elements, we will often want to consider *sequences* of elements that have a specific order and are not necessarily unique. Some examples of this are *pairs* like $\langle 1, 2\rangle$ and $\langle 1, 1\rangle$, triples like $\langle 4, 5, 3\rangle$, and more generally *tuples*, meaning some ordered sequence of n elements. Note that for our purpose, sequences are *not* sets, so we don't talk about $1 \in \langle 1, 2\rangle$: that's an "abstraction error."[20]

Now that we have a notion of sequences/tuples, we can form sets of them. If A is a set, and B is a set, then we can refer to the *product* of A and B, which we name $A \times B$, and it has the property that $c \in A \times B$ if and only if $c = \langle a, b\rangle$ for some $a \in A$ and $b \in B$.

Example 2. If $A = \{\, \mathsf{eat}, \mathsf{throw} \,\}$ and $B = \{\, \mathsf{chicken}, \mathsf{darts} \,\}$ then
$A \times B = \{\, \langle \mathsf{eat}, \mathsf{chicken}\rangle, \langle \mathsf{eat}, \mathsf{darts}\rangle, \langle \mathsf{throw}, \mathsf{chicken}\rangle, \langle \mathsf{throw}, \mathsf{darts}\rangle \,\}$.

There are several generalizations of products. You can write $A \times B \times C$ for the set of triples from these three sets: if you *really* want to write the product of $A \times B$ with C, you should parenthesize as $(A \times B) \times C$. This is just a convention since we are far more likely to want triples than pairs that include pairs in the first position.

Also, in analogy to arithmetic, the set $A^2 = A \times A$ and $A^3 = A \times A \times A$ and so on. Note that $A^0 = \{\, \langle\rangle \,\}$, the set containing only the empty sequence (since it's the only "0-long" tuple).

Given a set A, we use A^* to refer to the set of all finite sequences of elements of A. We sometimes use the name ε to denote the empty sequence $\langle\rangle$.

$$A^* = \bigcup_{n \in \mathbb{N}} A^n = A^0 \cup A^1 \cup A^2 \cup \ldots.$$

Technically, I haven't told you how to construct this set (we need one more axiom of set theory (Replacement) to do it, but we're not going to use it often, so I'm eliding it)

2.16.4 Trees

[RG: Under Construction!] [RG: Begin Stolen from Inductive Definitions Chapter] The key idea here is that we start with some basic, easily defined pre-existing set and filter it down to the subset that we want. Most mathematical definitions are based around this general idea.

We do not want to concern ourselves with as low level a representation as strings of bytes, so we start more abstractly. First, we assume some infinite set of *atomic* elements:

$$a \in \textsc{Atom}$$

Our base assumption is that there are an infinite number of Atoms in the set (not just 256), so we can always find another one if we need one.[21] Our only requirement of Atoms is that we can *tell them apart*: we don't care what they really look like, i.e. their internal structure, just whether we can tell if two of them are the same Atom or not. This is why they we call them atoms.[22] We can state this formally as a (seemingly obvious) property:

[20] Bear in mind that since all we *really* have to work with are sets, we end up encoding pairs and other sequences using, well, sets. See `https://en.wikipedia.org/wiki/Ordered_pair#Defining_the_ordered_pair_using_set_theory` for a variety of pair encodings in set theory that have been developed by a variety of mathematicians and logicians.

[21] To make this concrete, it's as though we had a computer that could work with arbitrary natural numbers instead of bytes.

[22] It helps to know that early physicists thought that atoms were the most primitive non-decomposable elements in the universe. Then along came neutrons and electrons and protons...and then quarks!

Proposition 10. $\forall a_1, a_2 \in \textsc{Atom}.\, a_1 = a_2 \vee a_1 \neq a_2$.

For our purposes, this should be interpreted as saying "given atoms a_1 and a_2, we can determine whether they are identical or not." That reading may be a bit different than you are used to, in that it might seem like a content-less tautology. In classical logic it is, but for us it's not! These Atoms will be used to represent the primitive constructs of our languages. If you've ever studied parsing, this is analogous to your tokens. For purposes of reasoning, we'll give each atom an abstract name, written in blue, like car, or avocado-toast, and whenever we use two different abstract names, we are referring to two different atoms, e.g. car $\neq$ avocado-toast, by convention. Note that this is *very* different than referring to atoms using *metavariables* like a_1 or a_2. We cannot up-front assume that $a_1 \neq a_2$: to do so we must have that fact as an immediate assumption, or be able to prove it from other assumptions.

Then, building on top of the Atoms, we assume a set of all possible trees of Atoms.

$$r \in \textsc{Tree}[\textsc{Atom}]$$

In set theory, we can create pairs of elements of sets, subsets of sets, and so forth. Using the basic tools of set theory, we can design a representation for trees, just like we can build tree data structures in the programming language of your choice. We're not going to build up a particular set-based representation of trees in any detail, but here's a sketch of a representation that gets the job done:

1. Pick some set $\mathcal{X}$;
2. Use sets to define some representation for sequences of $\mathcal{X}$, i.e. $\mathcal{X}^*$.
3. Define a tree of $\mathcal{X}$, i.e. $\textsc{Tree}[\mathcal{X}]$ as a set of non-empty sequences of $\mathcal{X}$, where each sequence describes a path from the root node of the tree to some subtree. This implies that:
   (a) The empty tree is represented by the empty set;
   (b) Every sequence starts with the same element of $\mathcal{X}$ (the root node of the tree);
   (c) If the sequence $\langle x_1, \ldots, x_n \rangle$ is an element of the set, then so is every non-empty prefix $\langle x_1, \ldots, x_k \rangle$ for $k \geq 1$: those are the paths to the ancestors of the node x_n;
   (d) If subtrees need to be ordered, or if multiple immediate subtrees can have the same payload, then that can be represented by ornamenting the "payload" $\mathcal{X}$ with indices.

In this case $\textsc{Tree}[\textsc{Atom}]$ is the name we use for the set of all trees that have atoms at their nodes. In general, we write $\textsc{Tree}[X]$ for "trees of elements of X." Note, that this name is just a convention: we define each set of trees manually, though we could use notational definitions to make the process schematic.

For now, since we are only concerned with trees of atoms, we will just write Tree as an abbreviation.

These trees in $\textsc{Tree}[\textsc{Atom}]$ have Atoms for nodes. For convenience, we can write them in parenthesized notation, where a node is juxtaposed with a parenthesized list of subtrees. For example, $a_1(a_2(), a_3())$ is the tree with parent node a_1 and two subtrees with root nodes a_2 and a_3 respectively. We can draw this as a diagrammatic tree:

For succinctness, we elide the empty parentheses after root nodes, writing instead $a_1(a_2, a_3)$. We will disambiguate where needed between the Atom a_1 and the $\textsc{Tree}[\textsc{Atom}]$ a_1, and similar for other kinds of trees.

[RG: Reasoning Principles]

The above construction is very concrete representation of Trees as sets of sequences. But we do not care about that particular representation, just that we *could* represent Trees somehow. But what really *counts* as a representation of Trees? Here we must accept that Trees are a structural concept, i.e. an API, and not an actual concrete thing. Any representation of Trees should do, so long as we never dive under the interface to take advantage of implementation bits. So we're doing some API design here, having created an implementation. We represent our API using abstract reasoning principles:

Proposition 11 (TREE[X] reasoning principles). *1. Let $R \subseteq$ TREE[X] and $x \in X$. Then $x(R) \in$ TREE[X];*

2. Let $r \in$ TREE[X]. Then there exists a unique $R \subseteq$ TREE[X] and $x \in X$ such that $x(R) = r$.

From these two principles, we can prove that If $r_1 = r_2$ then they have the same root and same set of children.

[RG: Mention Pierce's representation as a function from paths to values, which will also satisfy this relation]
[RG: End Stolen from Inductive Definitions Chapter]

2.16.5 Total Functions, Partial Functions, and Relations

From your days taking math classes you have a rough idea of what a function is: a mapping that takes one value to another. We can formalize that idea in set theory: if A is a set, and B is a set, then we have a set $A \to B$ of *total functions from A to B*. It is characterized as follows: $f \in A \to B$ if and only if

1. $f \subseteq A \times B$.
2. If $\langle a, b_1\rangle \in f$ and $\langle a, b_2\rangle \in f$ then $b_1 = b_2$. This says that f maps elements of A uniquely to elements of b. Notice that what this is saying is that $\langle a, b_1\rangle = \langle a, b_2\rangle$, so more informally, there is at most one pair in f with a in its first position.
3. If $a \in A$ then $\langle a, b\rangle \in f$ for some $b \in B$. This means informally that there is *at least* one pair in f with a in its first position.

We can relax the notion of total functions to get the notion of *partial functions*. Intuitively, a partial function is a mapping that doesn't necessarily map *every* element. From A and B, we can form the set of partial functions $A \rightharpoonup B$, where elements $f \in A \rightharpoonup B$ need only satisfy the first two properties of total functions: they don't have to map every element of A to some element in B, but any element that they do map, they map uniquely. It should be clear from this description that every total function is a partial function. Note as well, when we just say "function", we mean "total function."

A little bit of terminology about functions. Given a partial function $f : A \rightharpoonup B$, where this is just a common notation for $f \in A \rightharpoonup B$, we call A the function's *domain*; we call B the function's *codomain* (others call it the function's *range*). Sometimes when we need to talk about it, we may write $dom(f)$ and $cod(f)$ for the function's domain and codomain respectively.

Even more general than partial functions is the idea of *relations*. A relation captures many-to-many correspondences between elements of sets. The most direct analogue to a partial function is a *binary relation*, which is technically any set of pairs: a binary relation R on A and B is some subset $R \subseteq A \times B$. Relations need not be only binary: you can have a relation between three sets, i.e. $R \subseteq A \times B \times C$, and even a unary relation $R \subseteq A$, which is basically a way of representing some property of $A's$ by simply giving the subset of A that satisfies that property.

Notation We treat functions, partial functions, and relations with special notation sometimes. In particular, we introduce *function application* notation:

$$F[D \rightharpoonup C](A) \equiv \imath S.\, F \in D \rightharpoonup C \wedge \langle A, S\rangle \in F.$$

In words, $F[D \rightharpoonup C](A)$ denotes the unique set S such that F is a partial function from D's to C's, and F maps A to S. $F[D \rightharpoonup C](A)$ is only a proper description if F is indeed a partial function *and* F is defined for A. This in turn implies that A is in F's domain.

Unlike standard function application notation, this notation is annotated with a particular partial function domain and codomain. In principle we could abstract away from the particular domain and codomain, using existential quantification but in practice it makes reasoning unbelievably painful. Instead, we will typically omit the specific enunciation of the domain and codomain, as in the more standard notation $F(A)$, and depend on context clues to deduce what they should be. This may feel like cheating, but it corresponds to mathematical practice, and often a computer implementation of such notation in a proof assistant will use some form of *elaboration* to automate this process for the sake of user experience.

Thus, in the case of a partial function f, we write $f(a) = b$ to mean that $\langle a, b\rangle \in f$. We also write $f(a)$ to refer to the element b such that $\langle a, b\rangle \in f$. In the case of total functions, the expression $f(a)$ always makes sense, because there must be some b that fits the bill, but for partial functions, that may be nonsense, because the partial function may be undefined at a. If a partial function f is undefined at a, we write $(f(a)\uparrow$ which literally means $\forall b \in cod(f).\, \langle a, b\rangle \notin f$. The opposite, that f is defined for a, is written $f(a)\downarrow$.

We often write binary relations using *infix* notation:

$$a\ R\ b := \langle a, b\rangle \in R.$$

For a familiar example, $1 \leq 2$ just means $\langle 1, 2\rangle \in \leq$ where $\leq \subseteq \mathbb{N}\times\mathbb{N}$. For relations among more than two sets, we will often write them in "mixed-fix" notation. For example, when we discuss type systems, we will write the *typing judgment* $\Gamma \vdash t : T$ to mean that $\langle \Gamma, t, T\rangle \in (\bullet \vdash \bullet : \bullet)$ where $(\bullet \vdash \bullet : \bullet) \subseteq \textsc{TEnv} \times \textsc{Term} \times \textsc{Type}$. As with this example, we often use bullets $\bullet$ in the names of functions and relations to indicate the positions where arguments go. So we will refer to the typing judgment either by a usage, $\Gamma \vdash t : T$, or by the dotted name $\bullet \vdash \bullet : \bullet$. The nice thing about the usage version is that you can deduce the sets that make up the relation from its metavariables. The bullet notation, though, is clearer about which elements are really parameters.

Mathematical Functions Don't Run! One thing that it might take time to get used to in set theory is that functions are not like functions in programming languages...they don't take an input, churn on it, and then spit out a result after a bunch of computation. They're NOT *procedures*! Rather, they are (possibly infinite) lookup tables. Often when we write down a function definition, using a bunch of *equations*, it looks almost exactly like a program in a functional programming language like Haskell,[23] but really that is just a way of *naming* a particular function, just like we treat $\{ a, b, c \}$ as the *name* of a set. Such names, or descriptions, give us certain reasoning principles, and it's that process of reasoning that corresponds to the kind of computation that a function definition in a computer does. So two mottos that arise are:

- *Functions are like database tables or spreadsheets, not like procedures.*
- *Computation arises from deduction, not from definition.*

We'll return to this discussion later because it's a common sticking point for programmers when it comes to math. At least it was a sticking point for *this* programmer!

2.16.6 Set Operations

Most of the set constructions that we have been describing so far make "bigger" sets from smaller sets. We are very careful about the ways that we can build sets so that we don't write down a description of a set that can't possibly exist.[24]

However, once you have a set, you can always build a smaller set by choosing some of the elements of the set. We call this *set comprehension*: Given some set A, and some *predicate* $P(a)$ that describes properties of elements a of A, we can form the set $\{ a \in A \mid P(a) \}$ which is the set of elements of A for which the predicate P holds.

Partial Functions

$$A \rightharpoonup B = \{ f \in \mathcal{P}(A \times B) \mid \forall a \in A.\, \forall b_1, b_2 \in B.\, \langle a, b_1\rangle \in f \wedge \langle a, b_2\rangle \in f \Rightarrow b_1 = b_2 \}$$

Total Functions

$$\begin{aligned} A \to B &= \{ f \in A \rightharpoonup B \mid \forall a \in A.\, \exists b \in B.\, \langle a, b\rangle \in f \} \\ &= \left\{ f \in \mathcal{P}(A \times B) \;\middle|\; \begin{array}{l} (\forall a \in A.\, \exists b \in B.\, \langle a, b\rangle \in f) \wedge \\ (\forall a \in A.\, \forall b_1, b_2 \in B.\, \langle a, b_1\rangle \in f \wedge \langle a, b_2\rangle \in f \Rightarrow b_1 = b_2) \end{array} \right\} \end{aligned}$$

[23] That's no coincidence: they purposely stole the notation!

[24] For more on this, look at the Wikipedia page on "Paradoxes of set theory."

2.17 The Syntax of First-Order Set Theory

So far we have been simply using logical notation without explanation, in the hope that seeing it in context would provide *some* understanding, much like an immersive natural language course. However logical notation is precise, and a precise understanding is necessary for mastery. Let me try for moment to be a bit more precise about at least the low-level parts of first-order set theory. We will not be formal about all of it, but let me introduce some of it.

A word of warning though. In the following I use formal notation like Backus-Naur Form (BNF) simply to suggest the structure of the language, assuming that you have seen BNF notation before and can intuitively interpret it. Later I will give a very formal interpretation to BNF for programming languages. One can apply the same analysis to the following development as well, but doing so would amount to "defining set theory using set theory", kind of like writing a Python interpreter in Python.[25]

Representing a language inside itself does not amount to a fully-formal definition, but thanks to our human intuition and prior learning, we can still learn a lot from it. That's the goal of this section.

The core of the syntax of logic can be described as follows:

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\Xi \in \textsc{PropIdentifier}, \quad \Psi \in \textsc{AtomicProp}, \quad \Phi \in \textsc{Prop}, \quad S \in \textsc{SetIdentifier}, \quad \mathcal{E} \in \textsc{SetExp}} \\
\Psi & ::= & \Xi \mid \mathcal{E} = \mathcal{E} \mid \mathcal{E} \in \mathcal{E} \\
\Phi & ::= & \Psi \mid \top \mid \bot \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \Rightarrow \Phi \mid \forall S.\, \Phi \mid \exists S.\, \Phi \\
\mathcal{E} & ::= & S \mid \imath S.\, \Phi
\end{array}
$$

Here the metavariables S denotes a set identifier, while $\mathcal{E}$ denotes a set expression. Both are means of giving a *name* for a set. Some example names that we use are $\textsc{Term}$, or $\{\,1, 2, 3\,\}$. The latter is not a set in and of itself, but is a *descriptive* name: in the latter case we deduce properties of the set being named based on the structure of that name (whereas $\textsc{Term}$ is not immediately telling without knowing that the name $\textsc{Term}$ is equivalent to a more contentful set description). The propositional identifiers Ξ can be viewed as formal placeholders for real propositions. We use them to help explain the propositional connectives of our logic without having to worry about the particular content or meaning of an atomic proposition. This comes into play especially when we discuss our rules for logical deduction.

Now for a few common abbreviations. First, since we are not set-theorists—exploring the properties of *arbitrary sets*—but rather programming language theorists—exploiting set theory to study sets of programs and related structures—we will mostly be considering elements of *particular sets*. This leads to a desire to qualify our quantifiers. We do that by using the following notations that expand as described:

$$
\begin{aligned}
\forall S_1 \in S_2.\, \Phi &:= \forall S_1.\, S_1 \in S_2 \Rightarrow \Phi; \\
\exists S_1 \in S_2.\, \Phi &:= \exists S_1.\, S_1 \in S_2 \wedge \Phi.
\end{aligned}
$$

Take a moment to convince yourself that the expansions will have the intended meaning. It's indeed a bit curious that the expansion is not the same for universal and existential quantification, but this is necessary to induce the desired interpretation.

Also, since we often introduce many elements of the same set, we abbreviate this in terms of the above abbreviations:

$$
\begin{aligned}
\forall S_1, S_2, \ldots, S_n \in S.\, \Phi &:= \forall S_1 \in S.\, \forall S_2 \in S.\, \ldots\, \forall S_n \in S.\, \Phi; \\
\exists S_1 \in S_2.\, \Phi. &:= \exists S_1 \in S.\, \exists S_2 \in S.\, \ldots\, \exists S_n \in S.\, \Phi.
\end{aligned}
$$

Furthermore, we can abbreviate using the same kind of quantifier for different kinds of elements:

$$
\begin{aligned}
\forall S_1, S_2, \ldots, S_n \in S, T_1, T_2, \ldots, T_n \in T, .\, \Phi &:= \forall S_1, S_2, \ldots, S_n \in S. \forall T_1, T_2, \ldots, T_n \in T, .\, \Phi; \\
\exists S_1, S_2, \ldots, S_n \in S, T_1, T_2, \ldots, T_n \in T, .\, \Phi &:= \exists S_1, S_2, \ldots, S_n \in S. \exists T_1, T_2, \ldots, T_n \in T, .\, \Phi;
\end{aligned}
$$

Sometimes we wish to quantify elements of some n-ary relation. For example, recall that when given some binary relation $\odot \subseteq S \times T$, we write write $S_1 \odot T_1 := \langle S_1, T_2 \rangle \in (\bullet \odot \bullet)$ (occasionally using the $\bullet$ placeholders

[25] Nothing wrong with that! The PyPy project (`http://pypy.org`) has amply demonstrated benefits to doing this, and the LISP family of programming languages has a strong tradition of defining *metacircular interpreters* [?].

to indicate that we use *infix*, or more generally *mixed fix* notation) to indicate membership. Given such a relation, we use the following abbreviation to quantify elements of the relation in terms of elements of the subcomponents:

$$\forall(S_1 \odot T_2).\Phi := \forall S_1 \in S, T_2 \in T.\, S_1 \odot T_2 \Rightarrow \Phi;$$
$$\exists(S_1 \odot T_2).\Phi := \exists S_1 \in S, T_2 \in T.\, S_1 \odot T_2 \wedge \Phi.$$

Finally, we introduce some richer logical forms that are interpreted using the basic ones above:

$$\neg\Phi := \Phi \Rightarrow \bot;$$
$$\Phi_1 \Leftrightarrow \Phi_2 := (\Phi_1 \Rightarrow \Phi_2) \wedge (\Phi_2 \Rightarrow \Phi_1);$$
$$\exists! S.\, \Phi(S) := (\exists S.\Phi(S)) \wedge (\forall S_1, S_2.\, \Phi(S_1) \wedge \Phi(S_2) \Rightarrow S_1 = S_2).$$

The first one represents "not Φ, the second represents "if and only if"; and the third represents "there exists a unique set S such that Φ." The notation $\Phi(S)$ Says that Φ is a formula that *may* refer to S freely in it. Once this notation is introduced, then $\Phi(T)$ Refers to the same formula but with T substituted for S.