Sets 3

Today:

- ☐ Review; regular languages
- ☐ Sets that computers like
- ☐ Sets with operations
- ☐ The size of sets

Announcements:

- Quiz 2 on Friday, 25 mins, 7am – 7pm.
- I'm excited to see you in person next week. I request you come to class wearing a well-fitting N95 mask. But if you don't want to come back to in-person instruction—or you can't currently come back because you're sick or tested positive—there shouldn't be a problem.

Today's notes are divided into two halves, the first part on sets ("Sets 3") and the second part on relations ("Relations and Functions 1").

1 Review

Last time we talked about the **powerset** of a set and the **cross product** of two or more sets. Then I described **strings** and **languages**. We ended with a couple of operators on languages, **concatenation** and **star**, that are unique to languages. I defined regular languages, although we haven't *done* anything with them.

So let's do something with the idea. Suppose we fix an alphabet, say $\Sigma = \{0, 1\}$, and I "give you" the singleton sets $\{0\}$ and $\{1\}$. Throw them in a pot. Then I tell you: *you may enrich this pot by removing any two sets A and B and inserting into the pot $A \circ B$.* Now what strings will be in the pot?

The answer: all singleton sets of nonempty binary strings, $\{w : w \in \Sigma^* - \{\varepsilon\}\}$. Hmm. Makes you think we probably should have put the empty string into the pot at the beginning.

Now suppose I tell you: *you may also use union: given sets A and B in the pot, $A \cup B$ may also be in the pot.* Now what sets can you make?

Answer: all finite sets except $\emptyset$. For example $\{10, 101\} = \{1\} \circ \{0\} \cup \{1\}\{0\}\{1\}$. Kind of makes you think we should have put the empty set into the pot, too, at the very beginning. So let's throw it in.

Alright, so now we have a pot that contains all finite sets of strings—all finite languages. That's already pretty rich. For example, it contains all decimal representation of prime numbers of length less than 1000. It contains all legal Python programs less than 1 Gbyte. And so one. Any finite language $\{x_1, \ldots, x_n\}$.

Now, suppose I tell you: you can use the star operator, $(*)$, too. Given a language L in the pot, throw L^* into the pot.

Now the pot gets fuller. For example, are "all even strings" in the pot? Yes; it's $\{00, 01, 10, 11\}^*$. How about: all strings that start with 101? Sure; it's $\{101\}\ \{0,1\}^*$. Strings that start and end with the same character? Yes; that's $\{0\}\ \{0,1\}^*\{0\} \cup \{1\}\ \{0,1\}^*\{1\}$.

The languages that are in the pot now are called the *regular* languages, which I mentioned last time.

2 Sets that computers like

There are a bunch of finite sets that computers are especially fond of. Fond of because their designers directly build in support for these sets, allowing users to manipulate them with the computer's CPU-supported instruction set. Here are some of those key sets:

1. BYTES $= \{0,1\}^8$. The language of 8-bit strings. Traditionally, the most basic important grouping of bits. Note our use of formal-language notation, suggesting that we are thinking of bytes only as strings. They don't have a numeric value or something.

2. WORD32 $= \{0,1\}^{32}$. The language of all 32-bit strings. While we often think of these as nothing but strings, computer have operations that manipulate points in this set as though they represented numbers in $[0..2^{32} - 1]$. But not quite: sometimes the computer's sum, for example, will differ from the integer sum, due to overflow or underflow. So computer addition is actually quite different from integer addition: it is more like the world we will soon denote $\mathbb{Z}_{2^{32}}$ where everything is computed "modulo" 2^{32}.

   Similarly, computers support operations that manipulate points of WORD32 as though they were an integers in $[-2^{-31}..2^{31} - 1]$. But again, not quite: the computer's sum, may differ from the integer sum.

   It's an interesting question as to *what* string should be used to represent *what* number. It's not obvious! And there are different potential ways to answer the questions, especially when negative numbers are involved. The fact that I wrote $[-2^{-31}..2^{31} - 1]$ instead of $[-2^{-31} - 1..2^{31} - 1]$ instead of already implies that I had one representation in mind—the one that is usually employed.

3. WORD64 $= \{0,1\}^{64}$. The language of all 64-bit strings. Same thing about supporting different operations on this set, with different interpretations about what each string represents.

4. FLOAT32, FLOAT64. As a set, these are exactly the same as WORD32 and WORD64. But the computer now supports a bunch of different operations on them. Operations that

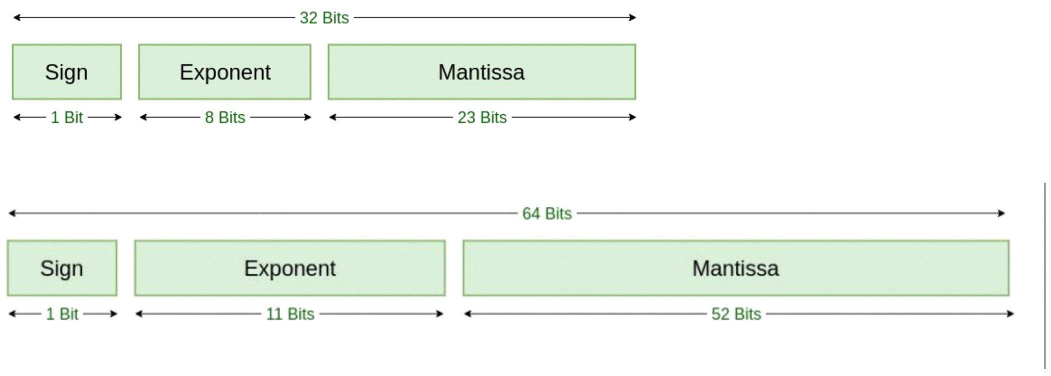

Figure 1: Some conventions of the IEEE 754 standard for floating point numbers.

are *like* addition, subtraction, multiplication, and division in the real numbers $\mathbb{R}$—yet operations that, routinely, don't actually coincide with these things.

Beyond the fact that FLOAT32 and FLOAT64 are finite, in radical contrast to the reals, the FLOAT32 and FLOAT64 types include elements totally unlike anything in the reals: they have a point we regard as a positive infinity $+\infty$; a point that we regard as negative infinity $-\infty$; and a point that we call NaN, for "not-a-number." The reals have nothing like these things! And it's not even true that distinct point in FLOAT32 or FLOAT64 always represent distinct real numbers: there are two points that represent zero, for example, a "positive zero" $+0$ and a "negative zero" -0. To learn about this very cool set and the operations on them, read about the IEEE 754 standard. Figure 1 sketches some conventions of the standard.

3 Sets with Operations

The discussion above makes apparent that we often want to think of our sets as having associated operations. The idea is a basic one in math and in computer science both. Mathematicians name these composite *things*, a set with some operations, with words like a *group*, a *ring*, or a *field*. The operation is required to have particular properties to deserve the label. In contrast, computer scientists use the word *abstract data type* for sets that have associated operations. Things like FLOAT64 is an abstract data types. But a wonderful thing about computer science is our willingness (eagerness?) to define new abstract data types. It turns out to be really useful to be able to conceptualize sets with operations on them as a single semantic *thing*. A *dictionary*. A *stack*. A *queue*. That sort of thing.

Here is an important example. A **group** is a nonempty set G together with a binary operation $\cdot : G \times G \to G$. To be called a group, three axioms must be satisfied:

1. For all $a, b, c \in G$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
2. There is some $e \in G$ such that for all $a \in G$, $e \cdot a = a \cdot e = a$.
3. For every element $a \in G$ there exists an element $b \in G$ such that $a \cdot b = b \cdot a = e$.

Can you rewrite these three requirements in first-order logic?

We think of the group as one *thing*: the set together with the operation on it. That's the group.

The first property above is called the *associative* property. The second property speaks to the existence of an *identity*. The third property talks about the existence of an *inverses* for every group element. We often use $+$ and 0 to represent the group operation and its inverse; other times we use $\cdot$ and 1.

In rather the same way, we can think of an abstract data type (ADT) as a set with operators, those operators having axioms that must be satisfied.

4 The size of sets

It is useful to distinguish sets that are *finite* from those that are *infinite*. It's common in CS to have sets that are finite but huge. Cryptographers like me live there. Among infinite sets, we distinguish between those that are *countably infinite* and those that are not, which we call *uncountable*. An infinite set S is set to be *countably infinite* if we can enumerate its elements $a_1, a_2, a_3, \ldots$, in some order: $S = \{a_1, a_2, a_3, \ldots\} = \{a_i\colon\ i \in \mathbb{N}\}$. A set is countable if it's finite or countably infinite. We will prove soon that uncountable sets exist.

A nice example of a countable set outside of the naturals is $\{0,1\}^*$. To list its elements, use the lexicographic ordering: $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$. Can you figure out a simple way to compute the ith string in the list? Or, going in the other direction, to convert take a string x and find the i such that the string is the ith string in our enumeration?