ECS 20 – Fall 2021 – Phillip Rogaway Sets

(((Have a feeling that I might not have talked about negating quantified formulas in one class. Check.

PUSHING QUANTIFIERS

$\neg (\forall x\ \phi) \equiv (\exists x)\ (\neg\phi)$
$\neg (\exists x\ \phi) \equiv (\forall x)\ (\neg\phi)$

Negate this:

$(\exists x)(\ \forall\ y)\ (y>x \rightarrow \exists\ z\ (z^2 + 5z = y))$

$\neg\ (\exists x)(\ \forall y)\ (y>x \rightarrow \exists z\ (z^2 + 5z = y))$
$(\forall x)\ \neg\ (\forall y)\ (y>x \rightarrow \exists\ z\ (z^2 + 5z = y))$
$(\forall x)\ (\exists y)\ \neg\ (y>x \rightarrow \exists\ z\ (z^2 + 5z = y))$

$\neg\ (A \rightarrow B) \equiv \neg\ (\neg A \vee B) \equiv (A \wedge \neg B)$

$(\forall x)\ (\exists y)\ (y>x \wedge\ \neg\exists\ z\ (z^2 + 5z = y))$
$(\forall x)\ (\exists y)\ (y>x \wedge\ \ \forall z \neg\ (z^2 + 5z = y))$
$(\forall x)\ (\exists y)\ (y>x \wedge\ \ \forall z (z^2 + 5z \neq y))$)))

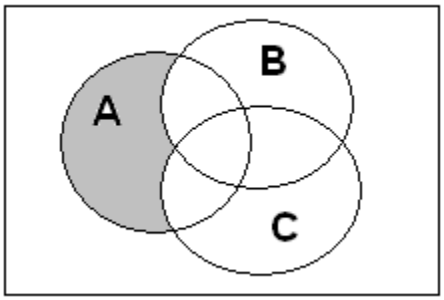
Set Theory

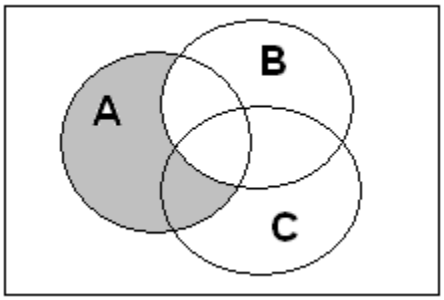
predicate symbols: 2-ary $\in$
function symbol: $\varnothing$

We write $a \in A$ instead of $\in (a, A)$.
But that doesn't change that $\in$ is a 2-ary predicate.

Seems **very** spare. What are other operators on sets, and how would we define them?

Define
union ($\cup$)
intersection ($\cap$)
complement (A^c or $\bar{A}$),
symmetric difference $\oplus$
set difference ($A \setminus B$ or $A - B$)
formally, and illustrating with **Venn Diagram:**





Eg: "For any pair of sets, x and y, there a set $x \cup y$ that contains all of the elements of x and y"

$(\forall x)(\forall y)(\exists z)(\forall u)\ (u \in z \leftrightarrow (u \in x) \vee (u \in y))$

We can do infinite unions and intersections too. Represented using the big-cup and big-cap notation.

What is $\bigcup_{n \in \mathbb{N}} \{2n\}$

What is $\bigcup_{m,b \in \mathbb{R}} \{(x,y): y = mx + b\}$

Def: $S = T$ iff $x \in S \leftrightarrow x \in T$

Def: $S \subseteq T$ if $x \in S \rightarrow x \in T$

$\{a, b\} \subseteq \{a,b,c\}$ YES
$\{a, b\} \subseteq \{a,b\}$ YES
$\{a\ b\} \subseteq \{a,d,e\}$ NO
$\varnothing \subseteq \{a,b,c\}$ YES (explain)
$\{\varnothing\} \subseteq \{a,b,c\}$ NO
$\{\varnothing\} \subseteq \{\{\varnothing\}\}$ YES
T/F: for all S, $\varnothing \subseteq S$: True

$a \notin A = \neg (a \in A)$
$A \subseteq B ::= (\forall x)(x \in A \rightarrow x \in B)$
$A \supseteq B ::= (\forall x)(x \in B \rightarrow x \in A)$

Can a set contain a set? **Yes**.
$S = \{\mathbb{N}, \{2,3\}, [0,1]\}$.

Can a set contain the empty set? **Yes**
In fact, we even use this for defining natural numbers!

$0 ::= \emptyset$
$1 ::= \{0\} = \{\emptyset\}$
$2 ::= \{0,1\} = \{\emptyset, \{\emptyset\}\}$
$3 ::= \{0,1,2\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$
…

Algebra of sets

$A \cup A = A$	$A \cap A = A$
$A \cup (B \cup C) = (A \cup B) \cup C$	$A \cap (B \cap C) = (A \cap B) \cap C$
$A \cup B = B \cup A$	$A \cap B = B \cap A$
$A \cup (B \cap C) = (A \cup C) \cap (B \cup C)$	$A \cap (B \cup C) = A \cap B \cup A \cap C$
$A \cup \emptyset = A$	$A \cap \emptyset = \emptyset$
$A \cup U = U$	$A \cap U = A$
$(A^c)^c = A$	
$A \cup A^c = U$	$A \cap A^c = \emptyset$
$U^c = \emptyset$	$\emptyset^c = U$

$(A \cup B)^c = A^c \cap B^c$ $\quad$ $(A \cap B)^c = A^c \cup B^c$ <-- De Morgan's laws

Let's prove one of these—say the first of the two De Morgan's laws. Often one proves that two sets are equal by showing that each is a subset of the other. But if we're careful we can save some work by chaining every thing together if if-and-only-ifs:

To show $(A \cup B)^c = A^c \cap B^c$
it's enough to show that
$x \in (A \cup B)^c$ iff $x \in A^c \cap B^c$.
Well

$x \in (A \cup B)^c$ iff
$\neg\ (x \in A \cup B)$ iff // Definition of the complement of a set
$\neg\ (x \in A \ \vee \ x \in B)$ iff // Definition of union of two sets
$\neg\ (x \in A) \ \wedge \ \neg\ (x \in B)$ iff // De Morgan's law in the *logical* setting
$x \in A^c \ \wedge \ x \in B^c$ iff // definition of the complement of a set
$x \in A^c \cap B^c$ // definition of intersection

Try to do the other De Morgan's law analogously
$(A \cap B)^c = A^c \cup B^c$

Ways to specify sets

$A = \{2i+1: i \in \mathbb{Z}\}$
$= \{...,-5,-3,-1, 1, 3, 5,...\}$ // But do we really all agree on the meaning of the ...
$= \{x: x \text{ is an odd integer}\}$
$= \{n: n \in \mathbb{Z} \text{ and } \neg\ (\exists j \in \mathbb{Z})(2j=n)\}$
Or

Let P be the set of prime numbers.
$P = \{n: n \textit{ is } \text{a prime number}\}$
$P = \{n \in \mathbb{N}: \ i \mid n \rightarrow i = 1 \ \vee n = -1 \ \vee i = n \ \vee i = -n\}$
$P = \{2,3,5,7,11,...\}$

Some important sets for math and computer science
$\mathbb{N} = \{1, 2, 3, ...\}$ // some books include 0, some don't
$\mathbb{R} = \{x: x \text{ is a real number}\}$
$\mathbb{Z} = \{...,-2,-1, 0, 1, 2, ,...\}$
$\mathbb{Q} = \{m/n: \quad m, n \in \mathbb{Z}, n \neq 0\}$

$[a .. b]$ integers between a and b, inclusive.
$[a, b]$ reals between a and b, inclusive

$[1..N] = \{1, 2, ..., N\}$
$[N] = \mathbb{Z}_N = \{0, 1, ..., N-1\}$

Naïve set theory, where we describe sets with natural language, where we write things like {x: …} with it being *implicit* the universe *U* from which x is drawn, can sometime run into trouble. Examples:

(1) Sets can contains sets. Give examples. But can a set contain itself? If we casually allow stuff like that, we encounter **Russell's paradox:** Let $S = \{ x \mid x \notin x\}$
Problem: is $S \in S$ iff $S \notin S$. Carefully go through the reasoning for this contradiction.

(2) Let BIG be the largest natural number that can be described with fewer than 200 characters of English text. *//92 chars*

What's wrong with this?
Let BIGGER be one more than BIG, where BIG is the largest natural number that can be described with fewer than 200 characters of English text. *//142 chars*

What's wrong is very subtle—really has to do with English being too imprecise to do this. If you're more careful about your descriptive language, you *can* define huge number with this approach. It becomes the "busy beaver function" of computability theory.

More operators on sets

Cartesian Product (= Cross product)

$A \times B = \{(a,b): a \in A, b \in B\}$
Use A^n for the *n*-fold cross product. $A \times A \times \cdots \times A$. How many times will *A* appear? *n*-1, not *n*.

$\mathbb{R}^2$ points in the plane

$A \times B \times C$: Thought of as ordered triples $\{(a,b,c): a \in A, b \in B, c \in C\}$ as opposed to pairs the first element of which is a pair.

A^n for the n-fold cross product of A with itself to pairs the first element of which is a pair.

How would you name an infinite collection of grid points in the plane? $\mathbb{Z} \times \mathbb{Z}$

Practice: Name all lines on the plane in set notation.
$S = \{L: L \text{ is a line in the plane}\}$

Or how about:
$L_{m,b} = \{(x,y) \in \mathbb{R}^2: \ y = mx+b\}$
$L_a = \{(x,y) \in \mathbb{R}^2: x=a\}$
$S = \{ L_{m,b} : m,b \in \mathbb{R}\} \cup \{L_a: a \in \mathbb{R}\}$

Sets of Strings (=Languages)

For computers, important sets correspond to those things that our architectures natively manipulate:

BYTES = $\{0,1\}^8$
WORDS32 = $\{0,1\}^{32}$
WORDS64 = $\{0,1\}^{64}$

These are sets of **strings**, a fundamental thing we consider sets of. What are strings?

An **alphabet** is a finite, nonempty set. We call its elements **characters**.
Eg: {0,1}, {0,1,2,3,4,5,6,7,8,9}, {a,b}, {a,b,…,z}, ASCII

A **string** is a finite sequence of characters.
Eg: hello, "this big dog", 10110011
Includes the **empty string**, ε , the unique string of length 0.
There's a basic operation on strings, **concatenation**. You stick them together. Hello ○ There = HelloThere Routinely written with a suppressed operator: x y = x ○ y.

An important kind of set in computer science is a **set of strings**. A set of strings is called a **language**.

In formal language theory, when we write L^n we aren't taking an n-wise cross product but, instead, applying the concatenation operator n-1 times.

So we don't end up with tuples; we end up with strings. You could regard strings as tuples, of course. Yet there is a bit of difference in how we understand these products:

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

$\{0,00\} \times \{0,00\} = \{(0,0), (0,00), (00,0), (00,00)\}$
$\{0,00\} \circ \{0,00\} = \{00,000,0000\}$

Not the same—even if we regard strings as tuples. Because 0 00 and 00 0 are treated as the same.

Representing Integers

You're all familiar by now with representing unsigned numbers between 0 and 2^n-1 in an n-bit value. How do we represent negative numbers, too? Something that better approximates our abstraction of an integer?

The most natural answer, perhaps, is

1-bit	*n-1 bits*
Sign	Unsigned-Value

If we establish the convention of using a Sign bit of 0 for a positive value and a Sign bit of 1 for a negative value, then we end up with bytes, for example, representing [-127, …, 127]. But we also, rather strangely, end up with one representation for a "positive zero" and one representation for a "negative zero". Illustrating it for nibbles instead:

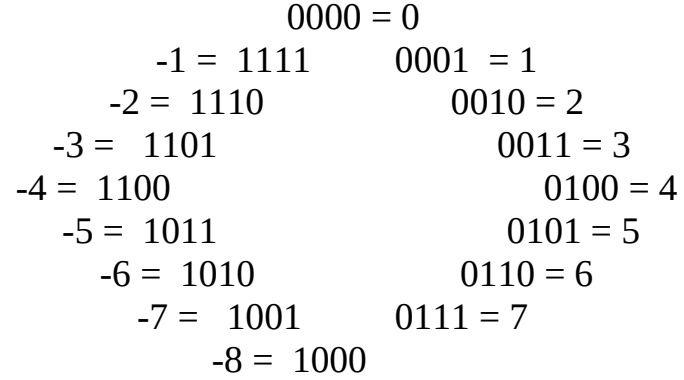
0000 0
0001 1
0010 2
0011 3
0100 4
0101 5
0110 6
0111 7

1000 -0
1001 -1
1010 -2
1011 -3
1100 -4
1101 -5
1110 -6
1111 -7

The -0 is weird, but maybe we wouldn't really care. On the other hand, if we try adding 5 + (-3), for example, we get

```
   0101  = 5
 + 1011  =-3
-------
    0000   which isn't the right answer
```

We can solve both problems at the same time by thinking of 0..15 as being arranged in a circle, and using this to suggest what the "right" representation is for -1, -2, …



Moving clockwise is adding 1; moving counterclockwise is subtracting 1. In some sense, -1=15; they are just different names for the same point. For that matter, binary 1000 corresponds to +8 and to -8 both. But if we are going to map the 16 points to a subset of the positive and negative integers, it makes sense to make the switch-over where the leading bit of 1. Hence the asymmetry of MININT=-8 while MAXINT=7 (for an 8-bit representation). For 32-bits, we'd have MAXINT = 2^{31}-1 = 2147483647 and MININT = -2^{31} = 2147483648. Not that big—2+ billion. 64-bits is more reasonable, with 2^{63} being about 10^{19}.

This is just suggestive, though; does it work?

```
   0101  = 5
 + 1101  =-3
-------
   0010  = 2   which is the right answer.  And so are the rest?
```

Here's the recipe:

To compute –A:
write the unsigned A, take the bitwise complement, the add 1

Why does this work? In a nutshell..

$A + \overline{A} = 11\ldots11 = -1$, so

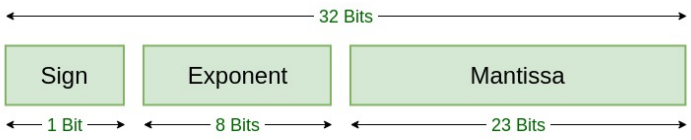
$A + (\overline{A} + 1) = 0$, whence the parenthesized quantity is -A

We will revisit this when we consider the world of integers mod N.

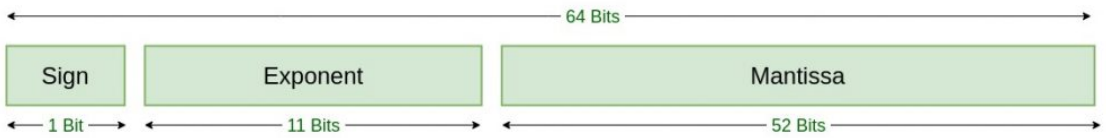
Representing Floating Point Numbers **IEEE 754**

IEEEFLOAT32 = $\{0,1\}^{32}$
IEEEFLOAT64 = $\{0,1\}^{64}$ = representing exponents -1022 .. 1023 (about 16 digits of accuracy)
Weirder than you may think

- sign, significand (=coefficient), exponent $(-1)^{sign} \cdot signficand \cdot 2^{exponent}$EEF
- + ∞ and −∞
- NaN (of various kinds)
- Zero can be +0 or –0



Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

TYPES	SIGN	BIASED EXPONENT	NORMALISED MANTISA	BIAS
Single precision	1(31st bit)	8(30-23)	23(22-0)	127
Double precision	1(63rd bit)	11(62-52)	52(51-0)	1023

Example

85.125

85 = 1010101

0.125 = 001

85.125 = 1010101.001

=1.010101001 x 2^6

sign = 0

1. Single precision:

biased exponent **127+6=133**

133 = 10000101

Normalized mantissa = **010101001**

We will add 0's to the right complete the 23 bits

The IEEE 754 Single precision is 0 10000101 01010100100000000000000

This can be written in hexadecimal form **42AA4000**

2. Double precision:

biased exponent 1023+6=1029

1029 = 10000000101

Normalised mantisa = 010101001

we will add 0's to complete the 52 bits

The IEEE 754 Double precision is:

= 0 10000000101 0101010010000000000000000000000000000000000000000000

This can be written in hexadecimal form **4055480000000000**

Special Values: IEEE has reserved some values that can ambiguity.

- **Zero –**

  Zero is a special value denoted with an exponent and mantissa of 0. -0 and +0 are distinct values, though they both are equal.

- **Denormalized –**

  If the exponent is all zeros, but the mantissa is not then the value is a denormalized number. This means this number does not have an assumed leading one before the binary point.

- **Infinity –**

  The values +infinity and -infinity are denoted with an exponent of all ones and a mantissa of all zeros. The sign bit distinguishes between negative infinity and positive infinity. Operations with infinite values are well defined in IEEE.

- **Not A Number (NAN) –**

  The value NAN is used to represent a value that is an error. This is represented when exponent field is all ones with a zero sign bit or a mantissa that it not 1 followed by zeros. This is a special value that might be used to denote a variable that doesn’t yet hold a value.

William Kahan. Primary architect of the IEEE 754 floating-point standard

Or particular language:

The set of all valid C programs

The set of valid URLs

The set of valid http programs

Sets with operations

What makes many sets interesting is the complement of operations that they support. You can add integers. You can multiply them. You can add and multiply WORD64 values. The operations are related but not the same. You can do logical operations on Boolean values. Etc. When we think about sets, we often want to think about the operations that go with them.

Sometimes these things are so tightly coupled that we think of the set along with the operations as the thing, rather than the set itself. We identify the key properties that the operation has, or is required to have. We do this both in pure mathematics and in computer science. A couple examples:

Group A group is a set G together with an operation $\cdot$ where:

1) $(x \cdot y) \cdot z = x \cdot (y \cdot z)$;

2) there exists an element 1 in G such that $x \cdot 1 = 1 \cdot x = x$;

3) for every element x there is an element y such that $x \cdot y = 1 = y \cdot x$

Sometimes we write the operation as + and the unit as 0. Whether we write it one way or the other is irrelevant; the properties demanded are the same. (But I think when we write it + there might be an implication that it's communitive.)

Example: Booleans with xor.
Example: Equal-length binary strings with bitwise xor
Example: Integers with customary addition
Example: WORD32 with an operation of addition that throws away the carry (that is, $\mathbb{Z}_{2\text{^}32}$)
~~Reals with multiply~~ *Explain why not*
Example: $\mathbb{R}$ - {0} under customary multiplication
~~Strings with concatenation~~ *Explain why not*

But let me emphasize that a set, all by itself, does **not** have operations defined on its elements

Dictionary ADT

and its realization with a list and with a hash table

Want to be able to **Insert** items into a dictionary and to **Lookup** if an item is already in the dictionary. (Sometimes want to be able to **Delete** an item, too.) For concreteness, think of the items we are inserting as strings.

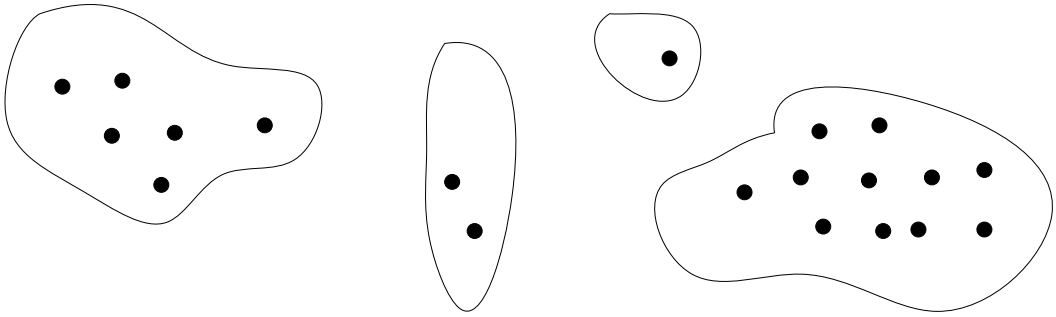
Example: discover how many distinct words are in a book.

Implementation
1) A **list** of words, each one appearing at most once.
2) A **hash table**.

Explain how each works.
Show how to modify the hash table to do a frequency count.

Representing a collection of sets in a computers

A different game – we are going to maintain a collection of **disjoint sets**. We want to be able to figure out if two things are in the same set, or in different sets. For example, each point in the set might represent a person and when we learn that person one and person two know one another – maybe one calls or emails the other – then we combine them. Each set then represents people that know one another through **some path** of knowing.



More interesting applications will come later, when we do graph theory.
You want to realize

- **find**(x) return a *canonical name* for the unique set containing *x*.
  x and y are in the same set iff find(x)=find(y)
- **union(x,y) merge the sets containing x and y.**
- **makeset** (x) create a set containing the element x. Return a canonical name for it

Naïve implementation: list of elements

Smarter – "union/find data structure"
Union by rank
Collapsing find.
Any sequence of *n* operations takes *n* □(*n*) time, for an incredibly slows growing function□□(*n*). [Omit big-O because not yet introduced]

Tarjan (1975)

```
function MakeSet(x)
     x.parent := x
     x.rank   := 0
 function Union(x, y)
     xRoot := Find(x)
     yRoot := Find(y)
     if xRoot == yRoot
         return

     // x and y are not already in same set. Merge them.
     if xRoot.rank < yRoot.rank
         xRoot.parent := yRoot
```

```
    else if xRoot.rank > yRoot.rank
        yRoot.parent := xRoot
    else
        yRoot.parent := xRoot
        xRoot.rank := xRoot.rank + 1
```

The second improvement, called *path compression*, is a way of flattening the structure of the tree whenever *Find* is used on it. The idea is that each

```
function Find(x)
    if x.parent != x
       x.parent := Find(x.parent)
    return x.parent
```

Power Set

$\mathcal{P}$ – Power set operator, unary operator (takes 1 input). $\mathcal{P}$(x) is the "set of all subsets of x"

$\mathcal{P}(X) = \{A: A \subseteq X\}$

Example: X = {a, b, c}

Example: $\mathcal{P}(\mathbb{N})$

Variant notation: $\mathcal{P}(X) = 2^X$

Notation is suggestive of size –
For X finite, $|\mathrm{P}(X)| = 2^{|X|}$

Zermello-Frankel Set Theory
See https://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory
for a nice description of the axioms.

1. The **axiom of extension** says that two sets are equal if and only if they have the same elements. For example, the set $\{1, 3\}$ and the set $\{3, 1\}$ are equal.
2. The **axiom of foundation** says that every set S (other than the empty set) contains an element that is disjoint (shares no members) with S.
3. The **axiom of specification** says that given a set S, and a predicate F (a function that is either true or false), that a set exists that contains exactly those elements of S where F is true. For example, if $S = \{1, 2, 3, 5, 6\}$, and F is "this is an even number", then the axiom says that the set $\{2, 6\}$ exists.
4. The **axiom of pairing** says that given two sets, there is a set whose members are exactly the two given sets. So, given the two sets $\{0, 3\}$ and $\{2, 5\}$, this axiom says that the set $\{\{0, 3\}, \{2, 5\}\}$ exists.
5. The **axiom of union** says that for any set, there exists a set that consists of just the elements of the elements of that set. For example, given the set $\{\{0, 3\}, \{2, 5\}\}$, this axiom says that the set $\{0, 3, 2, 5\}$ exists.
6. The **axiom of replacement** says that for any set S and a function F, that the set consisting of the results of calling F on all the members of S exists. For example, if $S = \{1, 2, 3, 5, 6\}$ and F is "add ten to this number", then the axiom says that the set $\{11, 12, 13, 15, 16\}$ exists.
7. The **axiom of infinity** says that the set of all integers (as defined by the Von Neumann construction) exists. This is the set $\{0, 1, 2, 3, 4, \dots\}$
8. The **axiom of power set** says that the power set (the set of all subsets) of any set exists. For example, the power set of $\{2, 5\}$ is $\{\{\}, \{2\}, \{5\}, \{2, 5\}\}$

9. The **axiom of choice** says that it is possible to take one object out of each of the elements of a set and make a new set. For example, given the set $\{\{0, 3\}, \{2, 5\}\}$, the axiom of choice would show that a set such as $\{3, 5\}$ exists. For finite sets, this axiom can be proved from the other axioms, but not for infinite sets.