Sets

What are sets?

- In many programming languages, sets are unordered containers of unique items
  - {3, 1, 2} and {1, 2, 3} are the same set
  - {1, 1, 2, 3} is *not* a set
- In Lean, to create a set, you need to first specify what type is contained in the set (in Lean, a set cannot contain elements of different types)
- A set of naturals (`set ℕ`) is a map from the naturals to a proposition
  - The proposition is whether a natural number is an element of the set

Set notation

- In Lean, the empty set is denoted with either `{}` or ∅.
  - For a set of natural numbers, the empty set is equivalent to the function: `λ(a: ℕ), false`
- We can also define the set of even numbers:
  - def evs : set ℕ := { n | ∃ m, m + m = n }
- Or a *singleton* set, containing just the number 1: `{1}`
  - What other ways could we define this set?
    - def one: set ℕ := {n: ℕ | n = 1}
    - def one': set ℕ := {n: ℕ | n > 0 ∧ n < 2}

Membership

- $1 \in \{1\}$
  - “One is a member (or element) of the set containing just the number 1”
- This reduces to:

  `1 = 1 ∨ false`
- Is $1 \in \{1, 2, 3\}$?
- Is $\{1, 2\} \in \{1, 2, 3\}$?

Union

- Consider the sets {1, 2, 3} (y) and {2, 3, 4} (z)
- What is the union of these two sets?
- I.e., what set contains all elements in either y and z? and no other elements?
  - Is this set uniquely defined?
- How is the union defined in Lean?

```
λ(a: ℕ), (a = 3 ∨ a = 2 ∨ a = 1 ∨ false) ∨ a = 4 ∨ a = 3 ∨ a = 2 ∨ false
```

Intersection

- Consider the sets {1, 2, 3} (y) and {2, 3, 4} (z)
- What is the intersection of these two sets?
- I.e., what set contains all elements in that are both in y and z and no other elements?
- How is this intersection defined in Lean?

```
λ(a: ℕ),(a = 3 ∨ a = 2 ∨ a = 1 ∨ false) ∧ (a = 4 ∨ a = 3 ∨ a = 2 ∨ false)
```

- Why are there parentheses around “z” here?

Set Difference

- Consider the sets {1, 2, 3} (y) and {2, 3, 4} (z)
- What is the difference from y to z?
- I.e., what set contains all elements in that are in y and that are not in z and no other elements?
- How is this defined in Lean?

```
λ(a : ℕ), (a = 3 ∨ a = 2 ∨ a = 1 ∨ false) ∧
          (a = 4 ∨ a = 3 ∨ a = 2 ∨ false → false)
```

Subset

- Is 1 a subset of {1, 2, 3}?
- Is {1} a subset of {1, 2, 3}?
- In Lean, the proposition {1} ⊆ {1, 2, 3} is expressed as
  `∀(a: ℕ), a = 1 ∨ false → a = 3 ∨ a = 2 ∨ a = 1 ∨ false`
- Note that there is a difference between x ⊆ y (subset) and x ⊂ y (*proper* subset)

Set Equality

- By the principle of extensionality, two sets are equal if for all elements of the set type, the element exists in the first set if and only if the element exists in the second set
- I.e., for A and B of type set T,
  `∀(e: T), (e ∈ A ↔ e ∈ B) → (A = B)`
- Are the sets {1, 2} and {2, 1} equal?

Powersets

- The powersets of set A is the set of all sets that are subsets of A
- If A is the set {1, 2}, $\mathcal{P}$ A is the set {∅, {1}, {2}, {1, 2}}
- Exercise: What is $\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathcal{P}\ \emptyset)))$?
- $\mathcal{P}\ \emptyset$ is {∅} (2^0 = 1 element)
- $\mathcal{P}(\mathcal{P}\ \emptyset)$ is {∅, {∅}} (2^1 = 2 elements)
- $\mathcal{P}(\mathcal{P}(\mathcal{P}\ \emptyset))$ is {∅, {∅}, {{∅}}, {∅, {∅}}} (2^2 = 4 elements)
- $\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathcal{P}\ \emptyset)))$ is {∅, {∅}, {{∅}}, {{{∅}}}, {{∅, {∅}}}, {∅, {∅}}, {∅, {{∅}}},
  {∅, {∅, {∅}}}, {{∅}, {{∅}}}, {{∅}, {∅, {∅}}},
  {{{∅}}, {∅, {∅}}}, … etc.} (2^4 = 16 elements)
- Yet more proofs

Tuples

- If S and T are types, then the product type of S and T, written out as (prod S T) and in shorthand as S × T, has as its values all of 2-tuples, or ordered pairs, (s, t), where s : S, and t : T.
- Does (1, 2) = (2, 1)?
- S and T can be different types!

Product Sets

- The Cartesian product set of two sets, A and B, denoted as A × B in everyday math, is the set of all ordered pairs, (a, b) (values of type prod A B), where a ∈ A and b ∈ B.
- Note that in Lean, there is a difference between `A × B` and `set.prod A B` — we want the latter not the former

Exercises

- How many elements are in the powerset of the vowels {a, b, c, d, e}?
- How many sets in this set have 1 element in them?
- How many sets have 2 elements in them?
- How many sets have 3 elements in them?
- How many sets have 4 elements in them?
- How many sets have 5 elements in them?
- How many elements are in the product set of the vowels with itself?

Set Complements

- The complement of the set A is written as -A and contains all of the elements in the type associated with set A that is not in set A.
- The complement of the set of even numbers is the set of odd numbers.

Insertion

- Lean supports syntax for adding (inserting) elements into a set to create another set (the original set is unmodified)

```
def myInsert
   {T : Type} (a: T) (s: set T): set T :=
     {b | b = a ∨ b ∈ s}
```

- `insert 5 {1, 2, 3, 4}` yields the set `{1, 2, 3, 4, 5}`

Sets as Functions

Review: What are functions?

- Functions take one or more inputs and provide one or more outputs
  - Technically, functions *can* take zero inputs (constant functions)
  - In some contexts, functions can have no outputs
  - Multiple outputs can be combined together as a single output that is a tuple
- Functions have domains (valid input values) and ranges (valid output values) and map values in the domain into values in the range
- In Lean, a function that takes inputs of type α and β and returns an outputs of type γ and δ would have the type: $\alpha \to \beta \to \gamma \times \delta$

Review: What are predicates

- Predicates are functions that return a proposition
  - E.g., $\alpha \rightarrow \beta \rightarrow$ Prop
- Alternatively, predicates are propositions with parameters
  - E.g., fromCharlottesville(p: Person): Prop := …
  - E.g., onSegment(s: Segment)(p: Point): Prop := …
- Note that in Lean, predicates have a return type of Prop and not bool
  - In other languages (e.g., PVS), this distinction is not meaningful

How can sets represent predicates

- There are two ways that sets can represent predicates
  1. The set could have the same domain as the predicate
     - Membership in the set would indicate a true proposition
     - Non-membership in the set represents a false proposition
     - E.g., for a predicate `my_pred(a: α)(b: β)`, we could define the set `my_set` to be of type `set (α × β)`
       - ```
         def my_set: set (α × β) :=
           {tuple | my_pred tuple.1 tuple.2}
         ```
  2. The set could follow the same convention as for other functions…
         ```

How can sets represent functions

- Consider a function taking a single natural number that is only valid over the inputs 1, 2, and 3 (its domain) and that returns a value that is one greater than its input (so its range is 2, 3, and 4)
- We can represent this as this set of tuples: {(1, 2), (2, 3), (3, 4)}

Exercise

- Represent the following functions as sets:
- `def fn1(n: ℕ) := n + 1`
  - `def set1: set (ℕ × ℕ) :=`
    `{tuple | tuple.2 = tuple.1 + 1}`
- `def fn2(n: ℕ)(m: ℕ) :=`
  `n / m`
  - `def set2: set (ℕ × ℕ × ℕ) :=`
    `{tuple | tuple.2.2 = tuple.1 / tuple.2.1}`
- `def fn3(n: ℕ)(m: {a: ℕ // a ≠ 0}) :=`
  `n / m`
  - `def set3: set (ℕ × {a: ℕ // a ≠ 0} × ℕ) :=`
    `{tuple | tuple.2.2 = tuple.1 / tuple.2.1}`

Exercise (2)

- Represent the following function as a set:
- `variable fn1: ℕ → ℕ`
  - `def set1: set (ℕ × ℕ) :=`
    `{tuple | tuple.2 = fn1 tuple.1}`
- `variable fn2: ℕ → ℕ → ℕ`
  - `def set2: set (ℕ × ℕ × ℕ) :=`
    `{tuple | tuple.2.2 = fn2 tuple.1 tuple.2.1}`
- `variable fn3: ℕ → ℕ → (ℕ × ℕ)`
  - `def set3: set (ℕ × ℕ × (ℕ × ℕ)) :=`
    `{tuple | (tuple.2.2.1, tuple.2.2.2) =`
    `fn3 tuple.1 tuple.2.1}`

Fin