Some basic set theory (and how it relates to Haskell)

John Longley

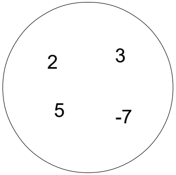
School of Informatics
University of Edinburgh
jrl@inf.ed.ac.uk

Inf1-FP guest lecture
25 September 2017

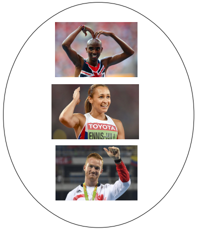
Sets

Intuitively, a set is simply a collection of things (of any kind).

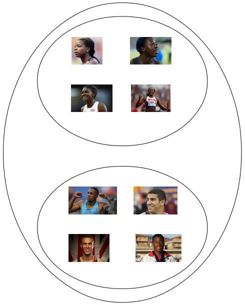
We'll allow the 'things' to be either mathematical objects or real-world entities. They could even be other sets.



A set of numbers



A set of GB athletes



A set of sets of GB athletes

Sets can be either finite, like those above, or infinite, like the mathematical set of all integers (often written as $\mathbb{Z}$), or the set of all real numbers ($\mathbb{R}$).

Membership

One way of defining sets (if they're quite small) is simply by listing all their members, like this:

$$\begin{aligned} A &= \{2, 3, 5, -7\} \\ B &= \{\mathrm{Mo}, \mathrm{Jess}, \mathrm{Greg}\} \end{aligned}$$

Curly brackets { } are standard mathematical notation for sets.

Membership

One way of defining sets (if they're quite small) is simply by listing all their members, like this:

$$\begin{aligned} A &= \{2, 3, 5, -7\} \\ B &= \{\mathrm{Mo}, \mathrm{Jess}, \mathrm{Greg}\} \end{aligned}$$

Curly brackets $\{\ \}$ are standard mathematical notation for sets.

We write $x \in A$ to mean x is a member (or element) of A. Also write $x \notin A$ to mean x is *not* a member of A. For example:

$$5 \in A \qquad 7 \notin A \qquad \mathrm{Jess} \in B \qquad \mathrm{Usain} \notin B$$

Membership

One way of defining sets (if they're quite small) is simply by listing all their members, like this:

$$\begin{aligned} A &= \{2, 3, 5, -7\} \\ B &= \{\mathrm{Mo}, \mathrm{Jess}, \mathrm{Greg}\} \end{aligned}$$

Curly brackets $\{\ \}$ are standard mathematical notation for sets.

We write $x \in A$ to mean x is a member (or element) of A. Also write $x \notin A$ to mean x is *not* a member of A. For example:

$$5 \in A \qquad 7 \notin A \qquad \mathrm{Jess} \in B \qquad \mathrm{Usain} \notin B$$

NB. We can read $x \in A$ either as x in A or as x is in A, according to the context. E.g. if P is the set of prime numbers, we can write:

For all $x \in P$, there exists $y > x$ such that $y \in P$.

Membership

One way of defining sets (if they're quite small) is simply by listing all their members, like this:

$$\begin{aligned} A &= \{2, 3, 5, -7\} \\ B &= \{\mathrm{Mo}, \mathrm{Jess}, \mathrm{Greg}\} \end{aligned}$$

Curly brackets $\{\ \}$ are standard mathematical notation for sets.

We write $x \in A$ to mean x is a member (or element) of A. Also write $x \notin A$ to mean x is *not* a member of A. For example:

$$5 \in A \qquad 7 \notin A \qquad \mathrm{Jess} \in B \qquad \mathrm{Usain} \notin B$$

NB. We can read $x \in A$ either as x in A or as x is in A, according to the context. E.g. if P is the set of prime numbers, we can write:

For all $x \in P$, there exists $y > x$ such that $y \in P$.
(x in P) *(y is in P)*

When are two sets the same?

We consider two sets to be equal if they have exactly the same members. That is, if A and B are sets, then $A = B$ if

for all $x \in A$ we have $x \in B$, *and* for all $x \in B$ we have $x \in A$.

This principle is called the extensionality rule for sets.

Consequence: The following are four different ways of writing exactly the same set.

$$\{3,5\} \qquad \{5,3\} \qquad \{3,3,5\} \qquad \{5,3,5,3,5,3,5,3,5,3\}$$

So we should think of sets as unordered collections of things without duplicates.

Test question: How many members does the following set have?

$$\{\{3\},\ \{3,3\},\ \{3,3,3\}\}$$

When are two sets the same?

We consider two sets to be equal if they have exactly the same members. That is, if A and B are sets, then $A = B$ if

for all $x \in A$ we have $x \in B$, *and* for all $x \in B$ we have $x \in A$.

This principle is called the extensionality rule for sets.

Consequence: The following are four different ways of writing exactly the same set.

$$\{3, 5\} \qquad \{5, 3\} \qquad \{3, 3, 5\} \qquad \{5, 3, 5, 3, 5, 3, 5, 3, 5, 3\}$$

So we should think of sets as unordered collections of things without duplicates.

Test question: How many members does the following set have?

$$\{\{3\},\ \{3, 3\},\ \{3, 3, 3\}\}$$

Answer: Just one — namely the set $\{3\}$.

Some particularly simple sets

For any entity x, we can form the singleton set $\{x\}$, whose only member is x.
Notice that

$$\text{Mo}$$

(who is an athlete) is *not* the same thing as

$$\{\ \text{Mo}\ \}$$

(a set of athletes), and this is itself not the same as

$$\{\ \{\ \text{Mo}\ \}\ \}$$

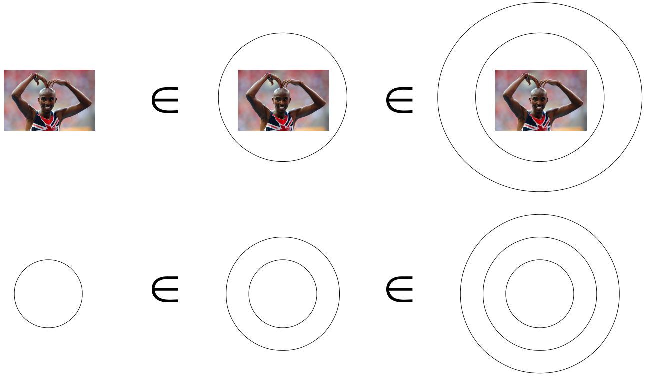
(a set of sets of athletes). However, we do have:

$$\text{Mo}\ \in\ \{\ \text{Mo}\ \}\ \in\ \{\ \{\ \text{Mo}\ \}\ \}\ \in\ \cdots$$

Another special set is the empty set $\{\ \}$, often written as $\emptyset$.
Again, $\emptyset$ (the empty set) is not the same thing as $\{\emptyset\}$ (a set with one element) ... though of course we have

$$\emptyset\ \in\ \{\emptyset\} \in \{\,\{\emptyset\}\,\}\cdots$$

In pictures . . .



Subsets

Suppose A and B are sets. We say A is a subset of B, and write $A \subseteq B$, if every member of A is also a member of B. That is,

$$A \subseteq B \text{ if for all } x \in A \text{ we have } x \in B.$$

Some people write just $\subset$, but $\subseteq$ seems nicer since we always have $A \subseteq A$. (Analogy: we have $5 \leq 5$, but not $5 < 5$.)

Examples:
set of Inf1-FP students $\subseteq$ set of UoE students:
set of prime numbers $\subseteq$ set of integers: $P \subseteq \mathbb{Z}$.

Notice that if $A \subseteq B$ and $B \subseteq A$ then $A = B$, by extensionality.

Exercise: Prove that if $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$.

Defining subsets

There's a useful notation for defining subsets of an existing set. Examples:

$$\{s \in \mathrm{UoE} \mid s \text{ is taking Inf1-FP}\} \qquad \{n \in \mathbb{Z} \mid n \text{ is prime}\}$$

The general form here is $\{x \in A \mid \cdots\}$, which can be pronounced 'the set of $x \in A$ such that $\cdots$'. (We'll meet other variations later.)

This carves out a subset of A consisting of all elements having a certain property, represented by the '$\cdots$'.

For now, we'll allow the 'property' to be expressed in English, as in the examples above. As you learn more, this will be increasingly replaced by formal logical notation.

This way of defining sets is known as set comprehension. It is the inspiration for a notation in Haskell known as list comprehension.

Ordered pairs

Example: Suppose we label the rows of a chessboard by $0, \ldots, 7$, and the columns also by $0, \ldots, 7$.
Then a square on the board can be identified by an ordered pair such as $(3, 5)$, sometimes written as $\langle 3, 5\rangle$. (Row 3, Column 5).

Notice that $(3, 5)$ and $(5, 3)$ refer to different squares! So ordered pairs are conceptually different from sets: order matters!
We also have ordered pairs like $(3, 3)$, referring to diagonal squares.

In general, for any entities x and y, we can form the ordered pair (x, y), also written $\langle x, y\rangle$. We say this pair has first component x and second component y.

If A and B are sets, we write $A \times B$ for the set of all ordered pairs (x, y) where $x \in A$ and $y \in B$. Example: If

$$A = \{\text{Jason, Laura}\}, \qquad B = \{2, 3, 5\}$$

then $A \times B$ is the set

$\{(\text{Jason}, 2), (\text{Jason}, 3), (\text{Jason}, 5), (\text{Laura}, 2), (\text{Laura}, 3) (\text{Laura}, 5)\}$

Combining pairing and comprehension

The combination of pairing and set comprehension is very powerful.

E.g. Suppose S is the set of UoE students, C the set of UoE courses. We can now form many sets such as . . .

1. $\{(s, c) \in S \times C \mid$ student s is taking course $c\}$
2. $\{(s, t) \in S \times S \mid$ s and t are taking the same courses$\}$
3. $\{(s, t) \in S \times S \mid$ s and t have a course in common$\}$
4. $\{(c, d) \in C \times C \mid$ c and d have a student in common$\}$
5. $\{(s, t) \in S \times S \mid$ s is a Facebook friend of $t\}$
6. $\{(s, n) \in S \times \mathbb{Z} \mid$ s is taking exactly n courses$\}$
7. $\{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x < y\}$
8. $\{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid$ y is divisible by $x\}$

Other notations are also used: e.g. example 7 might be written

$$\{(x, y) \mid x \in \mathbb{Z},\ y \in \mathbb{Z},\ x < y\}$$

Functions

Suppose A and B are sets. Intuitively, a function from A to B is a way of associating a member of B with each member of A.

We write $f : A \to B$ for 'f is a function from A to B'.
If $x \in A$, we write $f(x)$ for the associated member of B.

Examples:

1. Define $f : \mathbb{Z} \to \mathbb{Z}$ by $f(x) = x^2 + 5$.
2. Define m : Athletes $\to \mathbb{Z}$ by 'number of Olympic medals'.
3. Define e : Times $\to \mathbb{R}$ by 'exchange rate (dollars to pounds)'.
4. Define h : Points-on-Earth $\to$ Distances by 'height above sea level'.
5. Define g : Students $\times$ Times $\to$ Courses by:

$$g(s,t) \;=\; \text{favourite course of student } s \text{ at time } t\ .$$

Functions are often depicted by graphs, e.g. with x and y axes.

More on functions

Strictly speaking, a function $f : A \to B$ must associate exactly one member of B with each member of A.

[E.g. we can't define a function $f : \mathbb{R} \to \mathbb{R}$ by $f(x) = 1/x$, since '$1/0$' is undefined. But we may call f is a partial function $\mathbb{R} \rightharpoonup \mathbb{R}$.]

Of course, different members $x, y \in A$ may well get the same associated values $f(x), f(y) \in B$.

Suppose both f, g are functions $A \to B$. We consider f, g to be equal (as functions) if they associate the same member of B to each member of A:

$$f = g \quad \text{if for every } x \in A \text{ we have } f(x) = g(x) \quad .$$

This is called the principle of extensionality for functions.

Extending set comprehension notation

We can now relax our notation for set comprehension to allow things like . . .

1. $\{x^2 \mid x \in \mathbb{Z}\} \qquad = \{0, 1, 4, 9, 16, \ldots\}$
2. More generally, $\{f(x) \mid x \in A\}$, for any $f : A \to B$
3. $\{(a, m(a)) \mid a \in \text{Athletes}\}$. Has members such as $(\text{Mo}, 4)$.
4. $\{x \in \text{Points-on-Earth} \mid h(x) > 1000 \text{ metres}\}$
5. $\{g\,(\text{Jo}, t) \mid t \in \text{Times}\}$
   (All courses that have ever been Jo's favourite.)

Lists in Haskell

So far, we've been discussing general 'mathematical' concepts. Let's now look at some similar gadgetry within the Haskell programming language.

In Haskell, the most common way of representing 'collections of things' is via lists rather than sets. The notation [ ] is used for lists:

$$[2, 3, 5, -7] \qquad [\texttt{"Mo"}, \texttt{"Jess"}, \texttt{"Greg"}]$$

Lists are a bit different from sets: the order of elements matters, and duplicates are allowed. So the following are four different lists:

$$[3, 5] \qquad [5, 3] \qquad [3, 3, 5] \qquad [5, 3, 5, 3, 5, 3, 5, 3, 5, 3]$$

More on lists in Haskell

- In Haskell, all the elements in a list have to be of the same type. So we're not allowed 'mixed' lists like ["Mo", 5].
- We can write e.g. [1..100] as an abbreviation for the list of integers from 1 to 100, i.e. $[1, 2, 3, \cdots, 99, 100]$.
- In Haskell, a string such as "Mo" is actually just an abbreviation for a list of characters: `['M','o']`.
  So ["Mo","Jess","Greg"] is technically a list of lists of characters.
- For this lecture we'll stick with finite lists, though Haskell actually supports infinite lists too (perhaps surprisingly . . .)

List comprehension

List comprehension is a powerful means of constructing lists in Haskell. It is inspired by the set comprehension notation.

E.g. by analogy with the set

$$\{x^2 \mid x \in \mathbb{Z},\ -3 \leq x \leq 3\}$$

we have the list

```
[ x*x | x <- [-3..3] ]
```

Whereas the first of these defines the set $\{0, 1, 4, 9\}$, the second produces the list `[9,4,1,0,1,4,9]`.

The symbol `<-` may be pronounced ‘drawn from’. Notice how elements of the resulting list are generated from elements of the list `[-3,..3]`, taken in order.

Further examples of list comprehension

- `[ c | c <- "gobbledegook", c < 'm' ]`
  Produces `"gbbledegk"`
- `[ (x,x) | x <- [0..5] ]`
  Produces `[(0,0),(1,1),(2,2),(3,3),(4,4),(5,5)])`
- `[ (x,y) | x <- [0..3], y <- [0..3], x<y ]`
  Produces `[(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)]`

Using tricks like this, we can accomplish a lot of interesting things with very few lines of code!