Sets and Functions in Programming Languages

Simon Oddershede Gregersen
gregersen@cs.au.dk

June, 2018

1 Sets

A set is a collection of distinct *elements*. To indicate that a is an element of a set A one writes $a \in A$, while $a \notin A$ indicates that a is not an element of A. The elements of a set can literally be anything: numbers, people, shapes, strings, other sets, and so on. A set may be defined by a membership rule or by listing its elements. For example, the set given by the rule "all natural numbers less than three" can also be given by

$$\{0, 1, 2\}.$$

In principle, any finite set can be defined by an explicit list of its elements, but specifying infinite sets requires a rule or pattern to indicate its elements. For example, the ellipsis in

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

can be read "and so on" and indicates that the list of natural numbers $\mathbb{N}$ goes on forever. Conversely,

$$\mathbb{N} = \{n \in \mathbb{Z} \mid n \geq 0\}$$

also defines the set of natural numbers, that is, the set of all integers that are greater than or equal to 0. Note that the rule can be quite complex; for example, the set **even** of all even numbers can be defined

$$\textbf{even} = \{x \in \mathbb{Z} \mid x = 2k \text{ for some } k \in \mathbb{Z}\}.$$

The set containing no elements at all is called *the empty set*, and it is denoted by an empty pair of braces $\{\}$ or the symbol $\emptyset$.

Two sets A and B are said to be *equal* if and only if they contain exactly the same elements. In this case we write

$$A = B.$$

Note two further facts about sets:

- The order in the which elements are listed does not matter.
- If an element is listed more than once, any repeated occurrences are ignored.

So, for example, $\{1, 2, 3\}$, $\{3, 2, 1\}$, and $\{1, 1, 2, 2, 3, 3\}$ all denote the same set.

1.1 Subsets

Given two sets A and B, if every element of A is also an element of B, then A is said to be a *subset* of B and we write

$$A \subseteq B.$$

Equivalently, we write $B \supseteq A$ denoting that B is a *superset* of A. *Venn diagrams* can often be a useful way of illustrating relations between sets. In such diagrams, a set is represented by a circle and all points inside the circle represent elements of that set; Figure 1 depicts the sets A and B and that $A \subseteq B$.

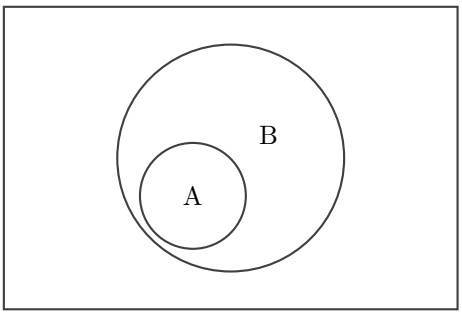

Figure 1: Venn diagram where $A \subseteq B$.

For example, the set of natural numbers makes up a subset of the set of integers ($\mathbb{N} \subseteq \mathbb{Z}$), and equivalently, the set of integers makes up a superset of the set of natural numbers ($\mathbb{Z} \supseteq \mathbb{N}$).

Notice that $A \subseteq B$ does not imply that B contains elements that A does not as the two sets could be equal. However, if B does contain at least one element that A does not, then we say A is a *proper subset* of B. In such a case we write

$$A \subset B.$$

Notice also that $\emptyset \subseteq A$ and $A \subseteq A$ is true for any set A. Finally, if $A \subseteq B$ and $B \subseteq A$, then A and B contain exactly the same elements and are therefore equal. Many propositions state that two sets A and B are equal, and often such propositions are shown by showing that any element in A is also an element in B (i.e. $A \subseteq B$) and vice versa (i.e. $B \subseteq A$).

1.2 Power sets

The *power set* of a set A is the set of all subsets of A, including the empty set $\emptyset$ and A itself. The power set of A is usually written as $\mathcal{P}(A)$. For example, if $A = \{1, 2, 3\}$, then

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

1.3 Operations

There are several fundamental operations for constructing new sets from existing sets. We have already seen the rule notation and the notion of power sets but just as we can combine two numbers to form a third with operations like addition, subtraction, multiplication, and division, so we can combine two sets to form a third with similar basic operations.

Intersection

Given two sets A and B, the *intersection* of A and B, denoted by $A \cap B$, is the set of elements that are both in A and in B. The intersection of A and B is illustrated by the gray area of the Venn diagram in Figure 2. For example, if $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$, then $A \cap B = \{2, 3\}$. Note that if $A \cap B = \emptyset$, then A and B are said to be *disjoint*.

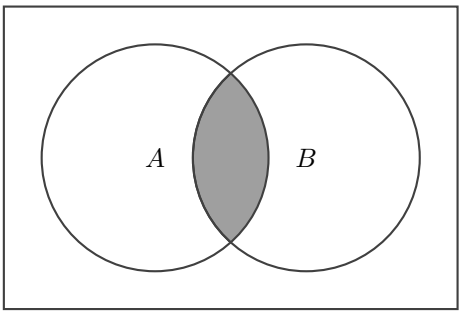

Figure 2: $A \cap B$.

Union

Given two sets A and B, the *union* of A and B, denoted by $A \cup B$, is the set of elements that are in A or in B. The union of A and B is illustrated by the gray area of the Venn diagram in Figure 3. For example, if $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$, then $A \cup B = \{1, 2, 3, 4\}$.

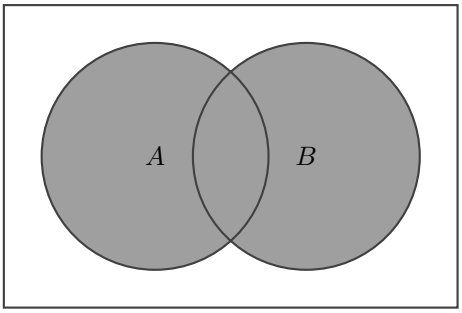

Figure 3: $A \cup B$.

Set difference

Given two sets A and B, the *set difference* of A and B, denoted by $A \setminus B$, is the set of elements that are in A but not in B. Note that B does not necessarily have to be a subset of A, and A and B might be disjoint. The set difference of A and B is illustrated by the gray area of the Venn diagram in Figure 4. For example, if $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$, then $A \setminus B = \{1\}$.

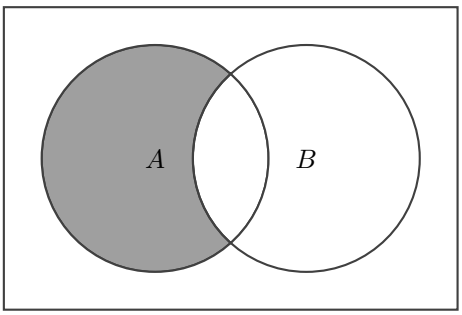

Figure 4: $A \setminus B$.

Algebraic Properties

The set operations satisfy many algebraic properties. For example, intersection and union are both *commutative*:

$$A \cap B = B \cap A$$
$$A \cup B = B \cup A,$$

associative:

$$A \cap (B \cap C) = (A \cap B) \cap C$$
$$A \cup (B \cup C) = (A \cup B) \cup C,$$

and *distribute* over each other:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

Cartesian Product

An *ordered pair* (a, b) is a pair of elements where the order is significant; the ordered pair (a, b) is different from (b, a) unless $a = b$. In contrast, the *unordered pair* (c, d) is equal to the unordered pair (d, c).

A new set can be constructed by associating every element of one set with every element in another set. The *Cartesian product* of two sets A and B, denoted by $A \times B$, is the set of all ordered pairs (a, b) such that $a \in A$ and $b \in B$:

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}.$$

For example, if $A = \{1, 2, 3\}$ and $B = \{a, b\}$, then

$$A \times B = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}.$$

The Cartesian product can be generalized to the *n-ary Cartesian product* over n sets $A_1, \ldots, A_n$ as the set of n-tuples

$$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) \mid a_i \in A_i\}.$$

Notice that the example above corresponds exactly to the case where $n = 2$.

1.4 Sets in Scala

Most programming languages provide an implementation of sets as part of the standard library. In Scala, the class `scala.collection.immutable.Set` (imported automatically) implements sets that behave as the mathematical definition, including most operations.

```
val A = Set(1, 2, 3) // {1, 2, 3}
```

We can test for containment of one or more elements in a set. The `contains` method asks whether a set contains a given element (i.e. whether $a \in A$). The more compact syntax `set(elem)` equivalent to `set contains elem` is available. The `subsetOf` method tests for the $\subseteq$ inclusion, that is, whether all elements of a set is also an element of another set.

```
val A = Set(1, 2, 3)
val B = Set(1, 2)
A contains 1 // true
A(1)         // true
A(4)         // false
A subsetOf B // false
B subsetOf A // true
```

Set operations for union, intersection and set difference are also available. Each operation exists in both an alphabetic and symbolic form. The alphabetic versions are `intersect`, `union`, and `diff`, whereas the symbolic versions are `&`, `|`, and `&~`, respectively.

```
val A = Set(1, 2, 3)
val B = Set(2, 3, 4)
A intersect B // {2, 3}
A union B     // {1, 2, 3, 4}
A diff B      // {1}
A & B         // {2, 3}
A | B         // {1, 2, 3, 4}
A &~ B        // {1}
```

Notice that `Set`s are *immutable*, that is, they cannot be modified after they have been created. Instead, we can create new sets. `Set` provides useful methods `+`, `++`, `-`, and `--` for constructing new sets from existing sets. Note that these operations can be completely defined in terms of the basic operations intersection, union and set difference.

```
val A = Set(1, 2, 3)
val B = Set(2, 3, 4)
A + 42        // {1, 2, 3, 42}
A + (42, 43) // {1, 2, 3, 42, 43}
A - 3         // {1, 2}
A ++ B        // {1, 2, 3, 4}
A -- B        // {1}
```

In Scala, all such operations are in fact method calls. For example, `A intersect B` and `A + 42` could just as well have been written `A.intersect(B)` and `A.+(42)`.

2 Functions

A *function* $f : X \to Y$ from a set X to a set Y is, informally speaking, a way to relate each element $x \in X$ to exactly one element $f(x) \in Y$. The element $f(x)$ is called the *image* of x under f and we write

$$x \mapsto f(x).$$

More formally, the notion of functions can be defined in terms of sets. A function f from X to Y is a subset of the Cartesian product $X \times Y$ subject to the following condition: every element of X is the first component of one and only one ordered pair in the subset. In other words, for every $x \in X$ there is exactly one element y such that the ordered pair (x, y) is contained in the subset defining the function f. For example, if $A = \{1, 2, 3\}$ and $B = \{a, b\}$, then the function $f : A \to B$ where $1 \mapsto a$, $2 \mapsto b$, and $3 \mapsto a$ can be represented by the set

$$\{(1, a), (2, b), (3, a)\} \subseteq A \times B.$$

Given two functions $f : X_1 \to Y_1$ and $g : X_2 \to Y_2$, then $f = g$, informally, if and only if $X_1 = X_2$ and $f(x) = g(x)$ for every $x \in X_1 = X_2$. Consider, for example, the absolute value on real values $x \in \mathbb{R}$ where we can define both $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ such that

$$f(x) = |x|$$
$$g(x) = \sqrt{x^2}.$$

These two function forms yield the same values for all $x \in \mathbb{R}$. As such, f and g are equal functions.

A *recursive definition* is used to define the elements of a set in terms of other elements in the set. A recursive definition of a function defines values of the function for some input in terms of the values of the same function for other inputs. For example, the n'th Fibonacci number $\text{fib}(n)$ can be computed by the recursively defined function

$$\text{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{otherwise} \end{cases}.$$

Note that not all recursive definitions are meaningful as, informally, the recursion must terminate after a bounded number of steps. For example, a function definition of $f : X \to Y$ such that $f(x) = f(x) + f(x)$ is not meaningful. A more formal treatment of this matter is out of scope of these notes.

Given $f : X \to Y$ and $g : Y \to Z$ we can define a new function $g \circ f : X \to Z$ called the *composition* of g and f. The composition of g and f is defined by

$$(g \circ f)(x) = g(f(x))$$

for all $x \in X$. For example, given $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ where $f(x) = 2x$ and $g(x) = x^2$, then

$$(g \circ f)(x) = g(f(x)) = (2x)^2 = 4x^2.$$

Note that function composition is associative; given $f : X \to Y$, $g : Y \to Z$, and a third function $h : Z \to V$, then

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

2.1 Partial Functions

A *partial* function $f : X \hookrightarrow Y$ from X to Y is a function $f : X' \to Y$ for some $X' \subseteq X$. The notion of a partial function generalizes the concept of function $f : X \to Y$ by not forcing f to map every element of X to an element of Y. That is, we will say that for any $x \in X$, either

- $f(x) \in Y$ or
- $f(x)$ is undefined.

The set of elements X' for which f is defined is called the *domain* of f and we write $\text{dom}(f) = X'$. If $X' = X$, then f is called a *total* function. For example, consider the function $f : \mathbb{Z} \hookrightarrow \mathbb{Z}$ defined as $f(x) = \sqrt{x}$, that is, the square root function restricted to integers. In that case, $f(x)$ is only defined if x is a perfect square (i.e. $0, 1, 4, 9, \ldots$) and is hence partial whereas a function like $g : \mathbb{Z} \to \mathbb{Z}$ where $g(x) = x^2$ is total.

As a convenience, we have notation for *extending* partial function definitions. Given a partial function $f : X \hookrightarrow Y$, the extended partial function $g : X \hookrightarrow Y$ that pairs $a \in X$ with $b \in Y$ but otherwise behaves as f is denoted by

$$g = f[a \mapsto b].$$

Equivalently,

$$g(x) = f[a \mapsto b](x) = \begin{cases} b & \text{if } x = a \\ f(x) & \text{otherwise} \end{cases}.$$

If we want to extend a partial function with multiple values at the same time we usually collapse the brackets. For example, the extended map $f[a \mapsto b][c \mapsto d]$ means the same as $f[a \mapsto b, c \mapsto d]$.

2.2 Function Spaces

A *function space* is a set of functions between two fixed sets. In set theory, the set of functions from a set X to a set Y may be denoted $X \to Y$; when writing $f : X \to Y$ we might just as well write $f \in X \to Y$. Similarly, $X \hookrightarrow Y$ denotes the set of partial functions from X to Y. Note that the specification of a function space can be arbitrarily complex. For example, if A, B, C, and D are sets, then $f : A \times (B \hookrightarrow C) \hookrightarrow D$ is a partial function that takes as input an ordered pair (a, g) where $a \in A$ and $g \in B \hookrightarrow C$, that is, g is also a partial function.

2.3 Maps

As described above, a function is generally defined by a formula that specifies how to compute output values from input values. In programming, a *map* is a (partial) function that is represented by a table of input/output values. For example, if $X = \{1, 2, 3\}$ and $Y = \{a, b\}$ then the map

$$[1 \mapsto a, 2 \mapsto b]$$

represents the partial function $f : X \to Y$ where

$$f(x) = \begin{cases} a & \text{if } x = 1 \\ b & \text{if } x = 2 \\ \text{undefined} & \text{otherwise} \end{cases}.$$

The empty map, written [], corresponds to a partial function that yields undefined for all inputs. Note that we often use the notation for extending partial function definitions in conjunction with maps.

2.4 Functions and Maps in Scala

In Scala, functions can be defined using `def`. As an example, the Fibonacci function can be defined as follows.

```
def fib(n : Int) : Int =
    if (n <= 1) n
    else fib(n - 1) + fib(n - 2)
```

Like sets, most programming languages also provide an implementation of maps as part of the standard library. In Scala, the class `scala.collection.immutable.Map` implements maps that behave more or less as the ones discussed in Section 2.3.

```
val A = Map(1 -> "a", 2 -> "b") // [1 -> "a", 2 -> "b"]
A(2) // "b"
```

Also like sets, `Map`s, as defined by `scala.collection.immutable.Map`, are immutable, but we can create new maps from existing ones.

```
val A = Map(1 -> "a", 2 -> "b")
A + (3 -> "c")              // [1 -> "a", 2 -> "b", 3 -> "c"]
A + (3 -> "c", 4 -> "d")    // [1 -> "a", 2 -> "b", 3 -> "c", 4 -> "d"]
A - 2                       // [1 -> "a"]
A -- Set(1, 2)              // []
```