Sets, Relations and Functions

2

Key Topics

Sets
Set Operations
Russell's Paradox
Computer Representation of sets
Relations
Composition of Relations
Reflexive, Symmetric and Transitive Relations
Relational Database Management System
Functions
Partial and Total Functions
Injective, Surjective and Bijective Functions
Functional Programming

2.1 Introduction

This chapter provides an introduction to fundamental building blocks in mathematics such as sets, relations and functions. Sets are collections of well-defined objects; relations indicate relationships between members of two sets A and B; and

G. O'Regan, *Guide to Discrete Mathematics*, Texts in Computer Science,
DOI 10.1007/978-3-319-44561-8_2

functions are a special type of relation where there is exactly (or at most)[1] one relationship for each element $a \in A$ with an element in B.

A set is a collection of well-defined objects that contain no duplicates. The term 'well defined' means that for a given value it is possible to determine whether or not it is a member of the set. There are many examples of sets such as the set of natural numbers $\mathbb{N}$, the set of integer numbers $\mathbb{Z}$, and the set of rational numbers $\mathbb{Q}$. The natural numbers $\mathbb{N}$ is an infinite set consisting of the numbers {1, 2, …}. Venn diagrams may be used to represent sets pictorially.

A binary relation R (A, B) where A and B are sets is a subset of the Cartesian product ($A \times B$) of A and B. The domain of the relation is A and the codomain of the relation is B. The notation aRb signifies that there is a relation between a and b and that $(a, b) \in R$. An n-ary relation R (A_1, A_2, … A_n) is a subset of ($A_1 \times A_2 \times \ldots \times A_n$). However, an n-ary relation may also be regarded as a binary relation $R(A, B)$ with $A = A_1 \times A_2 \times \ldots \times A_{n-1}$ and $B = A_n$.

Functions may be total or partial. A total function f: $A \rightarrow B$ is a special relation such that for each element $a \in$A there is exactly one element $b \in$B. This is written as $f(a) = b$. A partial function differs from a total function in that the function may be undefined for one or more values of A. The domain of a function (denoted by **dom** f) is the set of values in A for which the partial function is defined. The domain of the function is A provided that f is a total function. The codomain of the function is B.

2.2 Set Theory

A set is a fundamental building block in mathematics, and it is defined as a collection of well-defined objects. The elements in a set are of the same kind, and they are distinct with no repetition of the same element in the set.[2] Most sets encountered in computer science are finite, as computers can only deal with finite entities. Venn diagrams[3] are often employed to give a pictorial representation of a set, and they may be used to illustrate various set operations such as set union, intersection and set difference.

There are many well-known examples of sets including the set of natural numbers denoted by $\mathbb{N}$; the set of integers denoted by $\mathbb{Z}$; the set of rational numbers is denoted by $\mathbb{Q}$; the set of real numbers denoted by $\mathbb{R}$; and the set of complex numbers denoted by $\mathbb{C}$.

[1]We distinguish between total and partial functions. A total function f: $A \rightarrow B$ is defined for every element in A whereas a partial function may be undefined for one or more values in A.

[2]There are mathematical objects known as multi-sets or bags that allow duplication of elements. For example, a bag of marbles may contain three green marbles, two blue and one red marble.

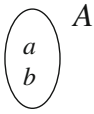
[3]The British logician, John Venn, invented the Venn diagram. It provides a visual representation of a set and the various set theoretical operations. Their use is limited to the representation of two or three sets as they become cumbersome with a larger number of sets.

Example 2.1 The following are examples of sets.

- The books on the shelves in a library
- The books that are currently overdue from the library
- The customers of a bank
- The bank accounts in a bank
- The set of Natural Numbers $\mathbb{N} = \{1, 2, 3, \ldots\}$
- The Integer Numbers $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$
- The non-negative integers $\mathbb{Z}^+ = \{0, 1, 2, 3, \ldots\}$
- The set of Prime Numbers = {2, 3, 5, 7, 11, 13, 17, …}
- The Rational Numbers is the set of quotients of integers

$$\mathbb{Q} = \{p/q : p, q \in \mathbb{Z} \text{ and } q \neq 0\}$$

A finite set may be defined by listing all of its elements. For example, the set $A = \{2, 4, 6, 8, 10\}$ is the set of all even natural numbers less than or equal to 10. The order in which the elements are listed is not relevant: i.e. the set {2, 4, 6, 8, 10} is the same as the set {8, 4, 2, 10, 6}.



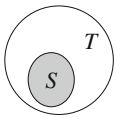
Sets may be defined by using a predicate to constrain set membership. For example, the set $S = \{n: \mathbb{N}: n \leq 10 \wedge n \bmod 2 = 0\}$ also represents the set {2, 4, 6, 8, 10}. That is, the use of a predicate allows a new set to be created from an existing set by using the predicate to restrict membership of the set. The set of even natural numbers may be defined by a predicate over the set of natural numbers that restricts membership to the even numbers. It is defined by

$$\text{Evens} = \{x | x \in \mathbb{N} \wedge even(x)\}.$$

In this example, $even(x)$ is a predicate that is true if x is even and false otherwise. In general, $A = \{x \in E \mid P(x)\}$ denotes a set A formed from a set E using the predicate P to restrict membership of A to those elements of E for which the predicate is true.

The elements of a finite set S are denoted by $\{x_1, x_2, \ldots x_n\}$. The expression $x \in S$ denotes that the element x is a member of the set S, whereas the expression $x \notin S$ indicates that x is not a member of the set S.

A set S is a subset of a set T (denoted $S \subseteq T$) if whenever $s \in S$ then $s \in T$, and in this case the set T is said to be a superset of S (denoted $T \supseteq S$). Two sets S and T are said to be equal if they contain identical elements: i.e. $S = T$ if and only if $S \subseteq T$ and $T \subseteq S$. A set S is a proper subset of a set T (denoted $S \subset T$) if $S \subseteq T$ and $S \neq T$. That is, every element of S is an element of T and there is at least one element in T that is not an element of S. In this case, T is a proper superset of S (denoted $T \supset S$).



The empty set (denoted by $\varnothing$ or $\{\}$) represents the set that has no elements. Clearly $\varnothing$ is a subset of every set. The singleton set containing just one element x is denoted by $\{x\}$, and clearly $x \in \{x\}$ and $x \neq \{x\}$. Clearly, $y \in \{x\}$ if and only if $x = y$.

Example 2.2

(i) $\{1, 2\} \subseteq \{1, 2, 3\}$
(ii) $\varnothing \subset \mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$

The cardinality (or size) of a finite set S defines the number of elements present in the set. It is denoted by $|S|$. The cardinality of an infinite[4] set S is written as $|S| = \infty$.

Example 2.3

(i) Given $A = \{2, 4, 5, 8, 10\}$ then $|A| = 5$.
(ii) Given $A = \{x \in \mathbb{Z}: x^2 = 9\}$ then $|A| = 2$
(iii) Given $A = \{x \in \mathbb{Z}: x^2 = -9\}$ then $|A| = 0$.

2.2.1 Set Theoretical Operations

Several set theoretical operations are considered in this section. These include the Cartesian product operation; the power set of a set; the set union operation; the set intersection operation; the set difference operation; and the symmetric difference operation.

Cartesian Product
The Cartesian product allows a new set to be created from existing sets. The Cartesian[5] product of two sets S and T (denoted $S \times T$) is the set of ordered pairs $\{(s, t) \mid s \in S, t \in T\}$. Clearly, $S \times T \neq T \times S$ and so the Cartesian product of two sets is not commutative. Two ordered pairs (s_1, t_1) and (s_2, t_2) are considered equal if and only if $s_1 = s_2$ and $t_1 = t_2$.

The Cartesian product may be extended to that of n sets $S_1, S_2, \ldots, S_n$. The Cartesian product $S_1 \times S_2 \times \ldots \times S_n$ is the set of ordered tuples $\{(s_1, s_2, \ldots, s_n) \mid s_1$

[4]The natural numbers, integers and rational numbers are countable sets whereas the real and complex numbers are uncountable sets.

[5]Cartesian product is named after René Descartes who was a famous 17th French mathematician and philosopher. He invented the Cartesian coordinates system that links geometry and algebra, and allows geometric shapes to be defined by algebraic equations.

$\in S_1, s_2 \in S_2, \ldots, s_n \in S_n\}$. Two ordered n-tuples $(s_1, s_2, \ldots, s_n)$ and $(s_1', s_2', \ldots, s_n')$ are considered equal if and only if $s_1 = s_1'$, $s_2, = s_2'$, $\ldots$, $s_n = s_n'$.

The Cartesian product may also be applied to a single set S to create ordered n-tuples of S: i.e. $S^n = S \times S \times \ldots \times S$ (n-times).

Power Set

The power set of a set A (denoted $\mathbb{P}A$) denotes the set of subsets of A. For example, the power set of the set $A = \{1, 2, 3\}$ has 8 elements and is given by

$$\mathbb{P}A = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

There are $2^3 = 8$ elements in the power set of $A = \{1, 2, 3\}$ and the cardinality of A is 3. In general, there are $2^{|A|}$ elements in the power set of A.

Theorem 2.1 (Cardinality of Power Set of A) *There are* $2^{|A|}$ *elements in the power set of A*

Proof Let $|A| = n$ then the cardinality of the subsets of A are subsets of size 0, 1, $\ldots$, n. There are $\binom{n}{k}$ subsets of A of size k.[6] Therefore, the total number of subsets of A is the total number of subsets of size 0, 1, 2, $\ldots$ up to n. That is

$$|\mathbb{P}A| = \sum_{k=0}^{n} \binom{n}{k}$$

The Binomial Theorem (we prove it in Example 4.2 in Chap. 4) states that

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

Therefore, putting $x = 1$ we get that

$$2^n = (1+1)^n = \sum_{k=0}^{n} \binom{n}{k} 1^k = |\mathbb{P}A|$$

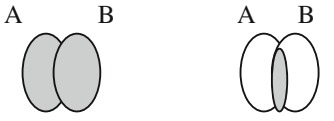
Union and Intersection Operations

The union of two sets A and B is denoted by $A \cup B$. It results in a set that contains all of the members of A and of B and is defined by

$$A \cup B = \{r | r \in A \text{ or } r \in B\}.$$

For example, suppose $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$ then $A \cup B = \{1, 2, 3, 4\}$. Set union is a commutative operation: i.e. $A \cup B = B \cup A$. Venn Diagrams are used to illustrate these operations pictorially.

[6] We discuss permutations and combinations in Chap. 5.



$A \cup B$ $A \cap B$

The intersection of two sets A and B is denoted by $A \cap B$. It results in a set containing the elements that A and B have in common and is defined by

$$A \cap B = \{r | r \in A \text{ and } r \in B\}.$$

Suppose $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$ then $A \cap B = \{2, 3\}$. Set intersection is a commutative operation: i.e. $A \cap B = B \cap A$.

Union and intersection are binary operations but may be extended to more generalized union and intersection operations. For example

$\cup_{i=1}^{n} A_i$ denotes the union of n sets.
$\cap_{i=1}^{n} A_i$ denotes the intersection of n sets

Set Difference Operations

The set difference operation $A \backslash B$ yields the elements in A that are not in B. It is defined by

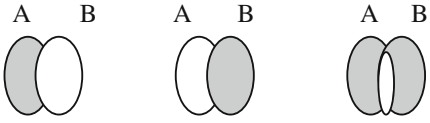
$$A \backslash B = \{a | a \in A \text{ and } a \notin B\}.$$

For A and B defined as A = {1, 2} and B = {2, 3} we have $A \backslash B = \{1\}$ and $B \backslash A = \{3\}$. Clearly, set difference is not commutative: i.e. $A \backslash B \neq B \backslash A$. Clearly, $A \backslash A = \varnothing$ and $A \backslash \varnothing = A$.

The symmetric difference of two sets A and B is denoted by $A \Delta B$ and is given by

$$A \Delta B = A \backslash B \cup B \backslash A$$

The symmetric difference operation is commutative: i.e. $A \Delta B = B \Delta A$. Venn diagrams are used to illustrate these operations pictorially.

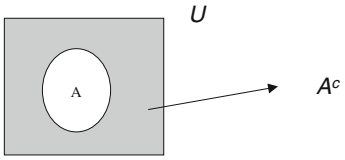


$A \setminus B$ $B \setminus A$ $A \Delta B$

The complement of a set A (with respect to the universal set U) is the elements in the universal set that are not in A. It is denoted by A^c (or A') and is defined as

$$A^c = \{u | u \in U \text{ and } u \notin A\} = U \backslash A$$

The complement of the set A is illustrated by the shaded area below



2.2.2 Properties of Set Theoretical Operations

The set union and set intersection properties are commutative and associative. Their properties are listed in Table 2.1.

These properties may be seen to be true with Venn diagrams, and we give a proof of the distributive property (this proof uses logic which is discussed in Chaps. 14–16).

Proof of Properties (Distributive Property)
To show $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Suppose $x \in A \cap (B \cup C)$ then

$$x \in A \wedge x \in (B \cup C)$$

$$\Rightarrow x \in A \wedge (x \in B \vee x \in C)$$

Table 2.1 Properties of set operations

Property	Description
Commutative	Union and intersection operations are commutative: i.e. $S \cup T = T \cup S$ $S \cap T = T \cap S$
Associative	Union and intersection operations are associative: i.e. $R \cup (S \cup T) = (R \cup S) \cup T$ $R \cap (S \cap T) = (R \cap S) \cap T$
Identity	The identity under set union is the empty set $\varnothing$, and the identity under intersection is the universal set U. $S \cup \varnothing = \varnothing \cup S = S$ $S \cap U = U \cap S = S$
Distributive	The union operator distributes over the intersection operator and vice versa. $R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$ $R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$.
DeMorgan's[a] Law	The complement of $S \cup T$ is given by $(S \cup T)^c = S^c \cap T^c$ The complement of $S \cap T$ is given by $(S \cap T)^c = S^c \cup T^c$

[a]De Morgan's law is named after Augustus De Morgan, a nineteenth century English mathematician who was a contemporary of George Boole

$$\Rightarrow (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C)$$
$$\Rightarrow x \in (A \cap B) \vee x \in (A \cap C)$$
$$\Rightarrow x \in (A \cap B) \cup (A \cap C)$$

Therefore, $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$
Similarly $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$
Therefore, $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

2.2.3 Russell's Paradox

Bertrand Russell (Fig. 2.1) was a famous British logician, mathematician and philosopher. He was the co-author with Alfred Whitehead of *Principia Mathematica*, which aimed to derive all of the truths of mathematics from logic. Russell's Paradox was discovered by Bertrand Russell in 1901, and showed that the system of logicism being proposed by Frege (discussed in Chap. 14) contained a contradiction.

Question (Posed by Russell to Frege)
Is the set of all sets that do not contain themselves as members a set?

Russell's Paradox
Let $A = \{S$ *a set and* $S \notin S\}$. *Is* $A \in A$? *Then* $A \in A \Rightarrow A \notin A$ *and* vice versa. *Therefore, a contradiction arises in either case and there is no such set A.*

Two ways of avoiding the paradox were developed in 1908, and these were Russell's theory of types and Zermelo set theory. Russell's theory of types was a response to the paradox by arguing that the set of all sets is ill formed. Russell developed a hierarchy with individual elements the lowest level; sets of elements at the next level; sets of sets of elements at the next level; and so on. It is then prohibited for a set to contain members of different types.

A set of elements has a different type from its elements, and one cannot speak of the set of all sets that do not contain themselves as members as these are of different

Fig. 2.1 Bertrand russell

types. The other way of avoiding the paradox was Zermelo's axiomatization of set theory.

Remark Russell's paradox may also be illustrated by the story of a town that has exactly one barber who is male. *The barber shaves all and only those men in town who do not shave themselves*. The question is who shaves the barber.

If the barber does not shave himself then according to the rule he is shaved by the barber (i.e. himself). If he shaves himself then according to the rule he is not shaved by the barber (i.e. himself).

The paradox occurs due to self-reference in the statement and a logical examination shows that the statement is a contradiction.

2.2.4 Computer Representation of Sets

Sets are fundamental building blocks in mathematics, and so the question arises as to how a set is stored and manipulated in a computer. The representation of a set M on a computer requires a change from the normal view that the order of the elements of the set is irrelevant, and we will need to assume a definite order in the underlying universal set $\mathscr{U}$ from which the set M is defined.

That is, a set is always defined in a computer program with respect to an underlying universal set, and the elements in the universal set are listed in a definite order. Any set M arising in the program that is defined with respect to this universal set $\mathscr{U}$ is a subset of $\mathscr{U}$. Next, we show how the set M is stored internally on the computer.

The set M is represented in a computer as a string of binary digits $b_1 b_2 \ldots b_n$ where n is the cardinality of the universal set $\mathscr{U}$. The bits b_i (where i ranges over the values 1, 2, … n) are determined according to the rule

$b_i = 1$ if ith element of $\mathscr{U}$ is in M
$b_i = 0$ if ith element of $\mathscr{U}$ is not in M

For example, if $\mathscr{U} = \{1, 2, \ldots 10\}$ then the representation of $M = \{1, 2, 5, 8\}$ is given by the bit string 1100100100 where this is given by looking at each element of $\mathscr{U}$ in turn and writing down 1 if it is in M and 0 otherwise.

Similarly, the bit string 0100101100 represents the set $M = \{2, 5, 7, 8\}$, and this is determined by writing down the corresponding element in $\mathscr{U}$ that corresponds to a 1 in the bit string.

Clearly, there is a one-to-one correspondence between the subsets of $\mathscr{U}$ and all possible n-bit strings. Further, the set theoretical operations of set union, intersection and complement can be carried out directly with the bit strings (provided that the sets involved are defined with respect to the same universal set). This involves a bitwise 'or' operation for set union; a bitwise 'and' operation for set intersection; and a bitwise 'not' operation for the set complement operation.

2.3 Relations

A binary relation $R(A, B)$ where A and B are sets is a subset of $A \times B$: i.e. $R \subseteq A \times B$. The domain of the relation is A and the codomain of the relation is B. The notation aRb signifies that $(a, b) \in R$.

A binary relation $R(A, A)$ is a relation between A and A. This type of relation may always be composed with itself, and its inverse is also a binary relation on A. The identity relation on A is defined by $a\ i_A a$ for all $a \in A$.

Example 2.4 There are many examples of relations

(i) The relation on a set of students in a class where $(a, b) \in R$ if the height of a is greater than the height of b.

(ii) The relation between A and B where A = {0, 1, 2} and B = {3, 4, 5} with R given by

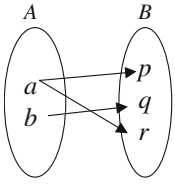
$$R = \{(0, 3), (0, 4), (1, 4)\}$$

(iii) The relation less than (<) between and $\mathbb{R}$ and $\mathbb{R}$ is given by

$$\{(x, y) \in \mathbb{R}^2 : x < y\}$$

(iv) A bank may represent the relationship between the set of accounts and the set of customers by a relation. The implementation of a bank account will often be a positive integer with at most eight decimal digits.
The relationship between accounts and customers may be done with a relation $R \subseteq A \times B$, with the set A chosen to be the set of natural numbers, and the set B chosen to be the set of all human beings alive or dead. The set A could also be chosen to be $A = \{n \in \mathbb{N}: n < 10^8\}$

A relation $R(A, B)$ may be represented pictorially. This is referred to as the graph of the relation, and it is illustrated in the diagram below. An arrow from x to y is drawn if (x, y) is in the relation. Thus for the height relation R given by $\{(a, p), (a, r), (b, q)\}$ an arrow is drawn from a to p, from a to r and from b to q to indicate that (a, p), (a, r) and (b, q) are in the relation R.



The pictorial representation of the relation makes it easy to see that the height of a is greater than the height of p and r; and that the height of b is greater than the height of q.

An n-ary relation R $(A_1, A_2, \ldots A_n)$ is a subset of $(A_1 \times A_2 \times \ldots \times A_n)$. However, an n-ary relation may also be regarded as a binary relation $R(A, B)$ with A = $A_1 \times A_2 \times \ldots \times A_{n-1}$ and $B = A_n$.

2.3.1 Reflexive, Symmetric and Transitive Relations

There are various types of relations including reflexive, symmetric and transitive relations.

(i) A relation on a set A is *reflexive* if $(a, a) \in R$ for all $a \in A$.
(ii) A relation R is *symmetric* if whenever $(a, b) \in$ R then $(b, a) \in R$.
(iii) A relation is *transitive* if whenever $(a, b) \in R$ and $(b, c) \in$ R then $(a, c) \in R$.

A relation that is reflexive, symmetric and transitive is termed an *equivalence relation*.

Example 2.5 (**Reflexive Relation**) A relation is reflexive if each element possesses an edge looping around on itself. The relation in Fig. 2.2 is reflexive.

Example 2.6 (**Symmetric Relation**) The graph of a symmetric relation will show for every arrow from a to b an opposite arrow from b to a. The relation in Fig. 2.3 is symmetric: i.e. whenever $(a, b) \in R$ then $(b, a) \in R$.

Example 2.7 (**Transitive relation**) The graph of a transitive relation will show that whenever there is an arrow from a to b and an arrow from b to c that there is an arrow from a to c. The relation in Fig. 2.4 is transitive: i.e. whenever $(a, b) \in R$ and $(b, c) \in R$ then $(a, c) \in R$.

Example 2.8 (**Equivalence relation**) The relation on the set of integers $\mathbb{Z}$ defined by $(a, b) \in R$ if $a - b = 2k$ for some $k \in \mathbb{Z}$ is an equivalence relation, and it partitions the set of integers into two equivalence classes: i.e. the even and odd integers.

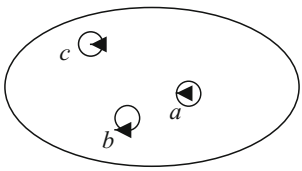


Fig. 2.2 Reflexive relation

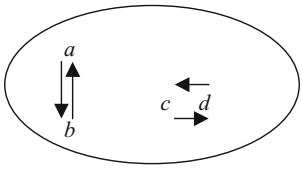


Fig. 2.3 Symmetric relation

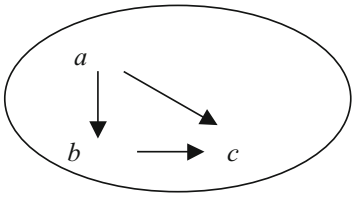


Fig. 2.4 Transitive relation

Domain and Range of Relation

The domain of a relation R (A, B) is given by $\{a \in A \mid \exists b \in B \text{ and } (a, b) \in R\}$. It is denoted by **dom** R. The domain of the relation R = $\{(a, p), (a, r), (b, q)\}$ is $\{a, b\}$.

The range of a relation R (A, B) is given by $\{b \in B \mid \exists a \in A \text{ and } (a, b) \in R\}$. It is denoted by **rng** R. The range of the relation $R = \{(a, p), (a, r), (b, q)\}$ is $\{p, q, r\}$.

Inverse of a Relation

Suppose $R \subseteq A \times B$ is a relation between A and B then the inverse relation $R^{-1} \subseteq B \times A$ is defined as the relation between B and A and is given by

$b\ R^{-1}\ a$ if and only if $a\ R\ b$

That is

$$R^{-1} = \{(b, a) \in B \times A: (a, b) \in R\}$$

Example 2.9 Let R be the relation between $\mathbb{Z}$ and $\mathbb{Z}^+$ defined by *mRn* if and only if $m^2 = n$. Then $R = \{(m, n) \in \mathbb{Z} \times \mathbb{Z}^+: m^2 = n\}$ and $R^{-1} = \{(n, m) \in \mathbb{Z}^+ \times \mathbb{Z}: m^2 = n\}$.

For example, $-3\ R\ 9$, $-4\ R\ 16$, $0\ R\ 0$, $16\ R^{-1} - 4$, $9\ R^{-1} - 3$, etc.

Partitions and Equivalence Relations

An equivalence relation on A leads to a partition of A, and vice versa for every partition of A there is a corresponding equivalence relation.

Let A be a finite set and let $A_1, A_2, \ldots, A_n$ be subsets of A such $A_i \neq \varnothing$ for all i, $A_i \cap A_j = \varnothing$ if $i \neq j$ and $A = \cup_i^n A_i = A_1 \cup A_2 \cup \ldots \cup A_n$. The sets A_i partition the set A, and these sets are called the classes of the partition (Fig. 2.5).

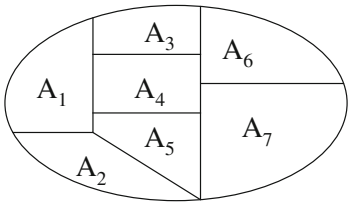


Fig. 2.5 Partitions of A

Theorem 2.2 (Equivalence Relation and Partitions) *An equivalence relation on A gives rise to a partition of A where the equivalence classes are given by Class (a) = {x | x ∈ A and (a, x) ∈ R}. Similarly, a partition gives rise to an equivalence relation R, where (a, b) ∈ R if and only if a and b are in the same partition.*

Proof Clearly, $a \in \text{Class}(a)$ since R is reflexive and clearly the union of the equivalence classes is A. Next, we show that two equivalence classes are either equal or disjoint.

Suppose $\text{Class}(a) \cap \text{Class}(b) \neq \varnothing$. Let $x \in \text{Class}(a) \cap \text{Class}(b)$ and so (a, x) and $(b, x) \in \text{R}$. By the symmetric property $(x, b) \in \text{R}$ and since R is transitive from (a, x) and (x, b) in R we deduce that $(a, b) \in \text{R}$. Therefore $b \in \text{Class}(a)$. Suppose y is an arbitrary member of Class (b) then $(b, y) \in \text{R}$ therefore from (a, b) and (b, y) in R we deduce that (a, y) is in R. Therefore since y was an arbitrary member of $\text{Class}(a)$ we deduce that $\text{Class}(b) \subseteq \text{Class}(a)$. Similarly, $\text{Class}(a) \subseteq \text{Class}(b)$ and so $\text{Class}(a) = \text{Class}(b)$.

This proves the first part of the theorem and for the second part we define a relation R such that $(a, b) \in \text{R}$ if a and b are in the same partition. It is clear that this is an equivalence relation.

2.3.2 Composition of Relations

The composition of two relations $R_1(A, B)$ and $R_2(B, C)$ is given by $R_2 \text{ o } R_1$ where $(a, c) \in R_2 \text{ o } R_1$ if and only there exists $b \in \text{B}$ such that $(a, b) \in R_1$ and $(b, c) \in R_2$. The composition of relations is associative: i.e.

$$(R_3 \text{ o } R_2) \text{ o } R_1 = R_3 \text{ o } (R_2 \text{ o } R_1)$$

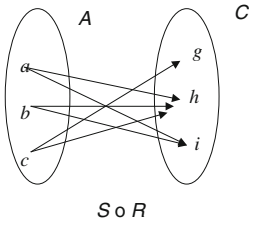
Example 2.10 (**Composition of Relations**) Consider a library that maintains two files. The first file maintains the serial number s of each book as well as the details of the author a of the book. This may be represented by the relation $R_1 = sR_1a$. The second file maintains the library card number c of its borrowers and the serial

number s of any books that they have borrowed. This may be represented by the relation $R_2 = c\ R_2 s$.

The library wishes to issue a reminder to its borrowers of the authors of all books currently on loan to them. This may be determined by the composition of R_1 o R_2: i.e. $c\ R_1$ o $R_2\ a$ if there is book with serial number s such that $c\ R_2\ s$ and $s\ R_1\ a$.

Example 2.11 (**Composition of Relations**) Consider sets $A = \{a, b, c\}$, $B = \{d, e, f\}$, $C = \{g, h, i\}$ and relations $R(A, B) = \{(a, d), (a, f), (b, d), (c, e)\}$ and $S(B, C) = \{(d, h), (d, i), (e, g), (e, h)\}$. Then we graph these relations and show how to determine the composition pictorially.

S o R is determined by choosing $x \in A$ and $y \in C$ and checking if there is a route from x to y in the graph (Fig. 2.6). If so, we join x to y in S o R. For example, if we consider a and h we see that there is a path from a to d and from d to h and therefore (a, h) is in the composition of S and R.



The union of two relations $R_1(A, B)$ and $R_2(A, B)$ is meaningful (as these are both subsets of $A \times B$). The union $R_1 \cup R_2$ is defined as $(a, b) \in R_1 \cup R_2$ if and only if $(a, b) \in R_1$ or $(a, b) \in R_2$.

Similarly, the intersection of R_1 and R_2 ($R_1 \cap R_2$) is meaningful and is defined as $(a, b) \in R_1 \cap R_2$ if and only if $(a, b) \in R_1$ and $(a, b) \in R_2$. The relation R_1 is a subset of R_2 ($R_1 \subseteq R_2$) if whenever $(a, b) \in R_1$ then $(a, b) \in R_2$.

The inverse of the relation R was discussed earlier and is given by the relation R^{-1} where $R^{-1} = \{(b, a) \mid (a, b) \in R\}$.

The composition of R and R^{-1} yields: R^{-1} o $R = \{(a, a) \mid a \in \text{dom } R\} = i_A$ and R o $R^{-1} = \{(b, b) \mid b \in \textbf{dom}\ R^{-1}\} = i_B$.

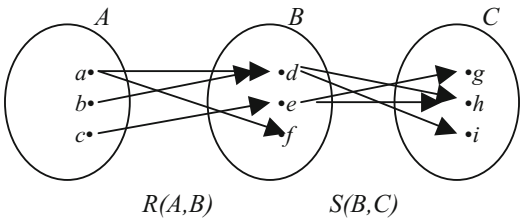


Fig. 2.6 Composition of relations

2.3.3 Binary Relations

A binary relation R on A is a relation between A and A, and a binary relation can always be composed with itself. Its inverse is a binary relation on the same set. The following are all relations on A:

$$R^2 = R \circ R$$
$$R^3 = (R \circ R) \circ R$$
$$R^0 = i_{\mathrm{A}}(\text{identity relation})$$
$$R^{-2} = R^{-1} \circ R^{-1}$$

Example 2.12 Let R be the binary relation on the set of all people P such that $(a, b) \in R$ if a is a parent of b. Then the relation R^n is interpreted as

R is the parent relationship
R^2 is the grandparent relationship
R^3 is the great grandparent relationship.
R^{-1} is the child relationship.
R^{-2} is the grandchild relationship.
R^{-3} is the great grandchild relationship

This can be generalized to a relation R^n on A where $R^n = R \circ R \circ \ldots \circ R$ (n-times). The transitive closure of the relation R on A is given by

$$R^* = \cup_{\mathrm{i}=0}^{\infty} R^i = R^0 \cup R^1 \cup R^2 \cup \ldots R^n \cup \ldots$$

where R^0 is the reflexive relation containing only each element in the domain of R: i.e. $R^0 = i_{\mathrm{A}} = \{(a, a) \mid a \in \mathbf{dom}\ R\}$.

The positive transitive closure is similar to the transitive closure except that it does not contain R^0. It is given by

$$R^+ = \cup_{\mathrm{i}=1}^{\infty} R^i = R^1 \cup R^2 \cup \ldots \cup R^n \cup \ldots$$

$a\ R^+\ b$ if and only if $a\ R^n\ b$ for some $n > 0$: i.e. there exists $c_1, c_2 \ldots c_n \in A$ such that

$$aRc_1, c_1Rc_2, \ldots, c_nRb$$

Parnas[7] introduced the concept of the limited domain relation (LD-relation), and a LD relation L consists of an ordered pair (R_L, C_L) where R_L is a relation and C_L is a subset of Dom R_L. The relation $\mathrm{R_L}$ is on a set U and C_L is termed the competence

[7]Parnas made important contributions to software engineering in the 1970s. He invented information hiding which is used in object-oriented design.

set of the LD relation *L*. A description of LD relations and a discussion of their properties are in Chap. 2 of [1].

The importance of LD relations is that they may be used to describe program execution. The relation component of the LD relation *L* describes a set of states such that if execution starts in state *x* it may terminate in state *y*. The set U is the set of states. The competence set of *L* is such that if execution starts in a state that is in the competence set then it is guaranteed to terminate.

2.3.4 Applications of Relations

A relational database management system (RDBMS) is a system that manages data using the relational model, and examples of such systems include RDMS developed at MIT in the 1970s; Ingres developed at the University of California, Berkeley in the mid-1970s; Oracle developed in the late 1970s; DB2; Informix; and Microsoft SQL Server.

A relation is defined as a set of tuples and is usually represented by a table. A table is data organized in rows and columns, with the data in each column of the table of the same data type. Constraints may be employed to provide restrictions on the kinds of data that may be stored in the relations. Constraints are Boolean expressions which indicate whether the constraint holds or not, and are a way of implementing business rules in the database.

Relations have one or more keys associated with them, and the *key uniquely identifies the row of the table*. An index is a way of providing fast access to the data in a relational database, as it allows the tuple in a relation to be looked up directly (using the index) rather than checking all of the tuples in the relation.

The Structured Query Language (SQL) is a computer language that tells the relational database what to retrieve and how to display it. A stored procedure is executable code that is associated with the database, and it is used to perform common operations on the database.

The concept of a relational database was first described in a paper '*A Relational Model of Data for Large Shared Data Banks*' by Codd [2]. A relational database is a database that conforms to the relational model, and it may be defined as a set of relations (or tables).

Codd (Fig. 2.7) developed the *relational data base model* in the late 1960s, and today, this is the standard way that information is organized and retrieved from computers. Relational databases are at the heart of systems from hospitals' patient records to airline flight and schedule information.

A binary relation R(A, B) where A and B are sets is a subset of the Cartesian product (A × B) of A and B. The domain of the relation is A, and the codomain of the relation is B. The notation *a*R*b* signifies that there is a relation between *a* and *b* and that $(a, b) \in R$. An *n*-ary relation R $(A_1, A_2, \ldots A_n)$ is a subset of the Cartesian product of the *n* sets: i.e. a subset of $(A_1 \times A_2 \times \ldots \times A_n)$. However, an

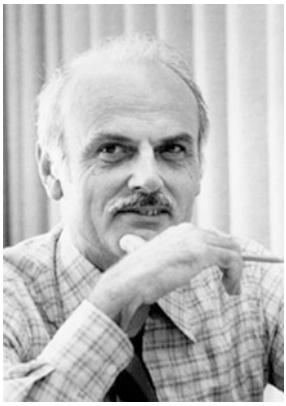


Fig. 2.7 Edgar Codd

n-ary relation may also be regarded as a binary relation R(A, B) with $A = A_1 \times A_2 \times \ldots \times A_{n-1}$ and $B = A_n$.

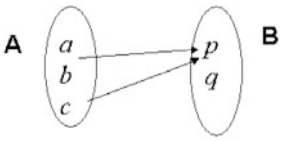
The data in the relational model are represented as a mathematical n-ary relation. In other words, a relation is defined as a set of n-tuples, and is usually represented by a table. A table is a visual representation of the relation, and the data are organized in rows and columns. The data stored in each column of the table are of the same data type.

The basic relational building block is the domain or data type (often called just type). Each row of the table represents one n-tuple (one tuple) of the relation, and the number of tuples in the relation is the cardinality of the relation. Consider the PART relation taken from [3], where this relation consists of a heading and the body. There are five data types representing part numbers, part names, part colours, part weights, and locations in which the parts are stored. The body consists of a set of n-tuples, and the PART relation given in Fig. 2.8 is of cardinality six.

For more information on the relational model and databases see [4]

2.4 Functions

A function f: $A \rightarrow B$ is a special relation such that for each element $a \in A$ there is exactly (or at most)[8] one element $b \in B$. This is written as $f(a) = b$.

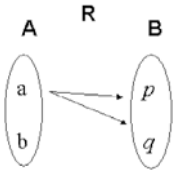


[8]We distinguish between total and partial functions. A total function is defined for all elements in the domain whereas a partial function may be undefined for one or more elements in the domain.

P#	PName	Colour	Weight	City
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Fig. 2.8 PART relation

A function is a relation but not every relation is a function. For example, the relation in the diagram below is not a function since there are two arrows from the element $a \in A$.



The domain of the function (denoted by **dom** *f*) is the set of values in A for which the function is defined. The domain of the function is A provided that *f* is a total function. The codomain of the function is B. The range of the function (denoted **rng** *f*) is a subset of the codomain and consists of

$$\mathbf{rng} f = \{r | r \in B \text{ such that } f(a) = r \text{ for some } a \in A\}.$$

Functions may be partial or total. A *partial function* (or partial mapping) may be undefined for some values of *A*, and partial functions arise regularly in the computing field (Fig. 2.9). *Total functions* are defined for every value in *A* and many functions encountered in mathematics are total.

Example 2.13 (**Functions**) Functions are an essential part of mathematics and computer science, and there are many well-known functions such as the trigonometric functions $\sin(x)$, $\cos(x)$, and $\tan(x)$; the logarithmic function $\ln(x)$; the exponential functions e^x; and polynomial functions.

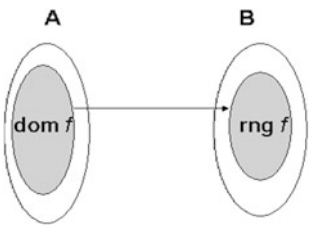


Fig. 2.9 Domain and range of a partial function

(i) Consider the partial function f: $\mathbb{R} \rightarrow \mathbb{R}$ where

$$f(x) = 1/x \quad (\text{where } x \neq 0).$$

This partial function is defined everywhere except for $x = 0$

(ii) Consider the function f: $\mathbb{R} \rightarrow \mathbb{R}$ where

$$f(x) = x^2$$

Then this function is defined for all $x \in \mathbb{R}$

Partial functions often arise in computing as a program may be undefined or fail to terminate for several values of its arguments (e.g. infinite loops). Care is required to ensure that the partial function is defined for the argument to which it is to be applied.

Consider a program P that has one natural number as its input and which for some input values will never terminate. Suppose that if it terminates it prints a single real result and halts. Then P can be regarded as a partial mapping from $\mathbb{N}$ to $\mathbb{R}$.

$$P : \mathbb{N} \rightarrow \mathbb{R}$$

Example 2.14 How many total functions f: $A \rightarrow B$ are there from A to B (where A and B are finite sets)?

Each element of A maps to any element of B, i.e. there are $|\mathrm{B}|$ choices for each element $a \in A$. Since there are $|\mathrm{A}|$ elements in A the number of total functions is given by

$$\begin{aligned} &|\mathrm{B}|\,|\mathrm{B}|\ldots|\mathrm{B}| \quad (|\mathrm{A}| \text{ times}) \\ &= |\mathrm{B}|^{|\mathrm{A}|} \qquad \text{total functions between A and B.} \end{aligned}$$

Example 2.15 How many partial functions f: $A \rightarrow B$ are there from A to B (where A and B are finite sets) ?

Each element of A may map to any element of B or to no element of B (as it may be undefined for that element of A). In other words, there are $|\mathrm{B}| + 1$ choices for each element of A. As there are $|\mathrm{A}|$ elements in A, the number of distinct partial functions between A and B is given by

$$\begin{aligned} &(|\mathrm{B}|+1)(|\mathrm{B}|+1)\ldots(|\mathrm{B}|+1) \quad (|\mathrm{A}| \text{ times}) \\ &= (|\mathrm{B}|+1)^{|\mathrm{A}|} \end{aligned}$$

Two partial functions f and g are equal if

1. dom f = dom g
2. $f(a) = g(a)$ for all $a \in$ dom f.

A function f is less defined than a function g ($f \subseteq$ g) if the domain of f is a subset of the domain of g, and the functions agree for every value on the domain of f

1. dom $f \subseteq$ dom g
2. $f(a) = g(a)$ for all $a \in$ dom f.

The composition of functions is similar to the composition of relations. Suppose f: A $\rightarrow$ B and g: B $\rightarrow$ C then $g \; o \; f$: A $\rightarrow$ C is a function, and this is written as $g \; o \; f$ (x) or $g(f(x))$ for $x \in$ A.

The composition of functions is not commutative and this can be seen by an example. Consider the function f: $\mathbb{R} \rightarrow \mathbb{R}$ such that $f(x) = x^2$ and the function g: $\mathbb{R} \rightarrow \mathbb{R}$ such that $g(x) = x + 2$. Then

$$g \, o f(x) = g\left(x^2\right) = x^2 + 2.$$
$$f \, o \, g(x) = f(x+2) = (x+2)^2 = x^2 + 4x + 4.$$

Clearly, $g \; o \; f(x) \neq f \; o \; g(x)$ and so composition of functions is not commutative. The composition of functions is associative, as the composition of relations is associative and every function is a relation. For f: A $\rightarrow$ B, g: B $\rightarrow$ C, and h: C $\rightarrow$ D we have

$$h \, o \, (g \, o f) = (h \, o \, g) \, o f$$

A function f: A $\rightarrow$ B is *injective* (*one to one*) if

$$f(a_1) = f(a_2) \Rightarrow a_1 = a_2.$$

For example, consider the function f: $\mathbb{R} \rightarrow \mathbb{R}$ with $f\ (x) = x^2$. Then $f(3) = f\ (-3) = 9$ and so this function is not one to one.

A function f: A $\rightarrow$ CB is *surjective* (*onto*) if given any $b \in$ B there exists an $a \in$ A such that $f(a) = b$ (Fig. 2.10). Consider the function f: $\mathbb{R} \rightarrow \mathbb{R}$ with $f(x) = x + 1$. Clearly, given any $r \in \mathbb{R}$ then $f\ (r - 1) = r$ and so f is onto.

A function is *bijective* if it is one to one and onto (Fig. 2.11). That is, there is a one-to-one correspondence between the elements in A and B for each $b \in B$ there is a unique $a \in A$ such that $f(a) = b$.

The inverse of a relation was discussed earlier and the relational inverse of a function f: A $\rightarrow$ B clearly exists. The relational inverse of the function may or may not be a function.

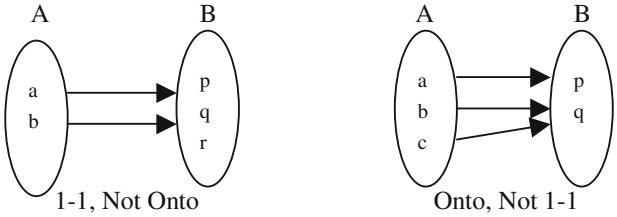


Fig. 2.10 Injective and surjective functions

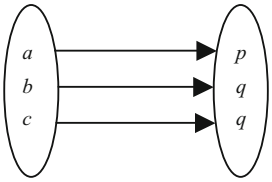


Fig. 2.11 Bijective function (One to one and Onto)

However, if the relational inverse is a function it is denoted by f^{-1}: B $\rightarrow$ A. A total function has an inverse if and only if it is bijective whereas a partial function has an inverse if and only if it is injective.

The identity function 1_A: A $\rightarrow$ A is a function such that $1_A(a) = a$ for all $a \in$ A. Clearly, when the inverse of the function exists then we have that $f^{-1}\ o\ f = 1_A$ and $f^{-}\ o\ f^{-1} = 1_B$.

Theorem 2.3 (**Inverse of Function**) *A total function has an inverse if and only if it is bijective.*

Proof Suppose f: $A \rightarrow B$ has an inverse f^{-1}. Then we show that f is bijective.

We first show that f is one to one.
Suppose $f(x_1) = f(x_2)$ then

$$\begin{aligned} f^{-1}(f(x_1)) &= f^{-1}(f(x_2)) \\ &\Rightarrow f^{-1}of(x_1) = f^{-1}of(x_2) \\ &\Rightarrow 1_A(x_1) = 1_A(x_2) \\ &\Rightarrow x_1 = x_2 \end{aligned}$$

Next we first show that f is onto. Let $b \in B$ and let $a = f^{-1}$ (b) then

$$f(a) = f(f^{-1}(b)) = b \text{ and so} f \text{ is surjective}$$

The second part of the proof is concerned with showing that if f: $A \rightarrow B$ is bijective then it has an inverse f^{-1}. Clearly, since f is bijective we have that for each $a \in A$ there exists a unique $b \in B$ such that $f(a) = b$.

Define g: $B \rightarrow A$ by letting $g(b)$ be the unique a in A such that $f(a) = b$. Then we have

$$g\,o\,f(a) = g(b) = a \,\text{and}\, f\,o\,g(b) = f(a) = b.$$

Therefore, g is the inverse of f.

2.5 Application of Functions

In this section, we discuss the applications of functions to functional programming, which is quite distinct from the imperative programming languages used in computing. Functional programming differs from imperative programming in that it involves the evaluation of mathematical functions, whereas imperative programming involves the execution of sequential (or iterative) commands that change the state. For example, the assignment statement alters the value of a variable, and the value of a given variable x may change during program execution.

There are no changes of state for functional programs, and the fact that the value of x will always be the same makes it easier to reason about functional programs than imperative programs. Functional programming languages provide *referential transparency*: i.e. equals may be substituted for equals, and if two expressions have equal values, then one can be substituted for the other in any larger expression without affecting the result of the computation.

Functional programming languages use higher order functions,[9] recursion, lazy and eager evaluation, monads,[10] and Hindley–Milner type inference systems.[11] These languages are mainly been used in academia, but there has been some industrial use, including the use of Erlang for concurrent applications in industry. Alonzo Church developed Lambda calculus in the 1930s, and it provides an abstract framework for describing mathematical functions and their evaluation. It provides the foundation for functional programming languages. Church employed lambda calculus to prove that there is no solution to the decision problem for first-order arithmetic in 1936 (discussed in Chap. 13).

[9]Higher order functions are functions take functions as arguments or return a function as a result. They are known as operators (or functionals) in mathematics, and one example is the derivative function dy/dx that takes a function as an argument and returns a function as a result.

[10]Monads are used in functional programming to express input and output operations without introducing side effects. The Haskell functional programming language makes use of uses this feature.

[11]This is the most common algorithm used to perform type inference. Type inference is concerned with determining the type of the value derived from the eventual evaluation of an expression.

Lambda calculus uses transformation rules, and one of these rules is variable substitution. The original calculus developed by Church was untyped, but typed lambda calculi have since been developed. Any computable function can be expressed and evaluated using lambda calculus, but there is no general algorithm to determine whether two arbitrary lambda calculus expressions are equivalent. Lambda calculus influenced functional programming languages such as Lisp, ML and Haskell.

Functional programming uses the notion of higher order functions. Higher order takes other functions as arguments, and may return functions as results. The derivative function $\mathrm{d}/\mathrm{d}x\ f(x) = f'(x)$ is a higher order function. It takes a function as an argument and returns a function as a result. For example, the derivative of the function $\mathrm{Sin}(x)$ is given by $\mathrm{Cos}(x)$. Higher order functions allow currying which is a technique developed by Schönfinkel. It allows a function with several arguments to be applied to each of its arguments one at a time, with each application returning a new (higher order) function that accepts the next argument. This allows a function of n-arguments to be treated as n applications of a function with 1-argument.

John McCarthy developed LISP at MIT in the late 1950s, and this language includes many of the features found in modern functional programming languages.[12] Scheme built upon the ideas in LISP. Kenneth Iverson developed APL[13] in the early 1960s, and this language influenced Backus's FP programming language. Robin Milner designed the ML programming language in the early 1970s. David Turner developed Miranda in the mid-1980s. The Haskell programming language was released in the late 1980s.

Miranda Functional Programming Language

Miranda was developed by David Turner at the University of Kent in the mid-1980s [5]. It is a non-strict functional programming language: i.e. the arguments to a function are not evaluated until they are actually required within the function being called. This is also known as lazy evaluation, and one of its main advantages is that it allows an infinite data structures to be passed as an argument to a function. Miranda is a pure functional language in that there are no side effect features in the language. The language has been used for

- Rapid prototyping
- Specification language
- Teaching Language

A Miranda program is a collection of equations that define various functions and data structures. It is a strongly typed language with a terse notation.

[12]Lisp is a multi-paradigm language rather than a functional programming language.

[13]Iverson received the Turing Award in 1979 for his contributions to programming language and mathematical notation. The title of his Turing award paper was 'Notation as a tool of thought'.

$$
\begin{aligned}
z = \mathrm{sqr}\,p/\mathrm{sqr}\,q \\
&\mathrm{sqr}\,k = k * k \\
&p = a + b \\
&q = a - b \\
&a = 10 \\
&b = 5
\end{aligned}
$$

The scope of a formal parameter (e.g. the parameter k above in the function sqr) is limited to the definition of the function in which it occurs.

One of the most common data structures used in Miranda is the list. The empty list is denoted by [ ], and an example of a list of integers is given by [1, 3, 4, 8]. Lists may be appended to by using the '++' operator. For example

$$[1,3,5] +\!+ [2,\ 4] = [1,3,5,2,4].$$

The length of a list is given by the '#' operator

$$\#[1,3] = 2$$

The infix operator ':' is employed to prefix an element to the front of a list. For example

$$5 : [2,4,6] \text{ is equal to } [5,2,4,6]$$

The subscript operator '!' is employed for subscripting: For example

$$\mathrm{Nums} = [5,2,4,6] \qquad \text{then} \quad \mathrm{Nums!0\ is\ 5}.$$

The elements of a list are required to be of the same type. A sequence of elements that contains mixed types is called a tuple. A tuple is written as follows:

$$\mathrm{Employee} = (\text{''Holmes''},\ \text{''222 Baker St. London''},\ 50,\ \text{''Detective''})$$

A tuple is similar to a record in Pascal whereas lists are similar to arrays. Tuples cannot be subscripted but their elements may be extracted by pattern matching. Pattern matching is illustrated by the well-known example of the factorial function

$$
\begin{aligned}
&\mathrm{fac}\,0 = 1 \\
&\mathrm{fac}(n+1) = (n+1) * \mathrm{fac}\,n
\end{aligned}
$$

The definition of the factorial function uses two equations, distinguished by the use of different patterns in the formal parameters. Another example of pattern matching is the reverse function on lists

$$\text{reverse}\,[] = []$$
$$\text{reverse}\,(a : x) = \text{reverse}\,x +\!\!+\, [a]$$

Miranda is a higher order language, and it allows functions to be passed as parameters and returned as results. Currying is allowed and this allows a function of n-arguments to be treated as n applications of a function with 1-argument. Function application is left associative: i.e. f x y means (f x) y. That is, the result of applying the function f to x is a function, and this function is then applied to y. Every function with two or more arguments in Miranda is a higher order function.

2.6 Review Questions

1. What is a set? A relation? A function?
2. Explain the difference between a partial and a total function.
3. Explain the difference between a relation and a function.
4. Determine A $\times$ B where A = $\{a, b, c, d\}$ and B = $\{1, 2, 3\}$
5. Determine the symmetric difference A Δ B where A = $\{a, b, c, d\}$ and B = $\{c, d, e\}$
6. What is the graph of the relation $\leq$ on the set A = $\{2, 3, 4\}$.
7. What is the composition of S and R (i.e. S o R), where R is a relation between A and B, and S is a relation between B and C. The sets A, B, C are defined as A = $\{a, b, c, d\}$, B = $\{e, f, g\}$, C = $\{h, i, j, k\}$ and R = $\{(a, e), (b, e), (b, g), (c, e), (d, f)\}$ with S = $\{(e, h), (e, k), (f, j), (f, k), (g, h)\}$
8. What is the domain and range of the relation R where R = $\{(a, p), (a, r), (b, q)\}$.
9. Determine the inverse relation R^{-1} where R = $\{(a, 2), (a, 5), (b, 3), (b, 4), (c, 1)\}$.
10. Determine the inverse of the function f: $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ defined by

$$f(x) = \frac{x-2}{x-3} (x \neq 3) \quad \text{and} f(3) = 1$$

11. Give examples of injective, surjective and bijective functions.

12. Let $n \geq 2$ be a fixed integer. Consider the relation $\equiv$ defined by$\{(p, q)$: $p, q \in \mathbb{Z}, n \mid (q - p)\}$

    a. Show $\equiv$ is an equivalence relation.
    b. What are the equivalence classes of this relation?

13. Describe the differences between imperative programming languages and functional programming languages.

2.7 Summary

This chapter provided an introduction to set theory, relations and functions. Sets are collections of well-defined objects; a relation between A and B indicates relationships between members of the sets A and B; and functions are a special type of relation where there is at most one relationship for each element $a \in A$ with an element in B.

A set is a collection of well-defined objects that contain no duplicates. There are many examples of sets such as the set of natural numbers $\mathbb{N}$, the integer numbers $\mathbb{Z}$ and so on.

The Cartesian product allows a new set to be created from existing sets. The Cartesian product of two sets S and T (denoted $S \times T$) is the set of ordered pairs $\{(s, t) \mid s \in S, t \in T\}$.

A binary relation R (A, B) is a subset of the Cartesian product $(A \times B)$ of A and B where A and B are sets. The domain of the relation is A and the codomain of the relation is B. The notation aRb signifies that there is a relation between a and b and that $(a, b) \in R$. An n-ary relation R $(A_1, A_2, \ldots A_n)$ is a subset of $(A_1 \times A_2 \times \ldots \times A_n)$.

A total function f: $A \rightarrow B$ is a special relation such that for each element $a \in A$ there is exactly one element $b \in$ B. This is written as $f(a) = b$. A function is a relation but not every relation is a function.

The domain of the function (denoted by **dom** f) is the set of values in A for which the function is defined. The domain of the function is A provided that f is a total function. The codomain of the function is B.

Functional programming is quite distinct from imperative programming in that there is no change of state, and the value of the variable x remains the same during program execution. This makes functional programs easier to reason about than imperative programs.

References

1. Software Fundamentals. Collected Papers by David *L.* Parnas. Edited by Daniel Hoffman and David Weiss. Addison Wesley. 2001.
2. A Relational Model of Data for Large Shared Data Banks. E.F. Codd. Communications of the ACM 13 (6): 377–387. 1970.
3. An Introduction to Database Systems. 3rd Edition. C.J. Date. The Systems Programming Series. 1981.
4. Introduction to the History of Computing. Gerard O'Regan. Springer Verlag. 2016.
5. Miranda. David Turner. Proceedings IFIP Conference, Nancy France, Springer LNCS (201). September 1985.