10 FUNCTIONS

In the previous chapter we looked at binary relations. In this chapter we see that a function is just a binary relation, but with some restrictions. We see how to define functions, how to get answers from functions and how to use functions.

10.1 FUNCTIONS

A function is just a binary relation in which all the first components are different. For example:

$$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \}$$

is a function but

$$\{ 1 \mapsto a, 2 \mapsto b, 1 \mapsto c \}$$

is not because the first component, 1, appears twice.

Examples of functions include:

$STUDENTID \leftrightarrow STUDENT$	because no two students have the same id
$REGISTRATATION \leftrightarrow VEHICLE$	because no two vehicles have the same registration number
$BANKACCOUNT \leftrightarrow OWNER$	because each bank account is unique.

An owner could be one person, a couple or a company, for example. An owner could have several bank accounts. But a bank account belongs to just one owner. If you said: "who owns this bank account?" you will get just one answer such as *Mr Jones*, or *Mr & Mrs Smith*, or *tmarrisdotcom*.

EXERCISE 10.1

1 Name three further examples of functions. Explain why each example is a function.

10.2 FUNCTIONS AS BINARY RELATIONS

A function is a binary relation and so we can use the usual binary relation operations - but with care.

domain:	$\mathrm{dom}\ \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \{ 1, 2, 3 \}$
range:	$\mathrm{ran}\ \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \{ a, b \}$
domain restriction:	$\{ 2 \} \lhd \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} = \{ 2 \mapsto b \}$
range restriction:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \rhd \{ a \} = \{ 1 \mapsto a, 3 \mapsto a \}$
relational image:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} (\!\{ 2 \}\!) = \{ b \}$
override:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \oplus \{ 3 \mapsto c, 4 \mapsto d \} = \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 4 \mapsto d \}$
composition:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \fatsemi \{ a \mapsto X, b \mapsto Y \} = \{ 1 \mapsto X, 2 \mapsto Y, 3 \mapsto X \}$

The inverse of a function is not necessarily another function. For example:

$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \}^{\sim} = \{ a \mapsto 1, b \mapsto 2, a \mapsto 3 \}$	is not a function because *a* appears twice in the result domain
$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto c \}^{\sim} = \{ a \mapsto 1, b \mapsto 2, c \mapsto 3 \}$	is a function because each first component in the result domain is unique

And because a function is a binary relation and a binary relation is a set, we can use the usual set operations, but again with care.

intersection:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \cap \{ 1 \mapsto a, 3 \mapsto a \} = \{ 1 \mapsto a, 3 \mapsto a \}$
difference:	$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \setminus \{ 1 \mapsto a, 3 \mapsto a \} = \{ 2 \mapsto b \}$

The union of two functions is not necessarily another function.

$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \cup \{ 2 \mapsto c \} = \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a, 2 \mapsto c \}$	is not a function because 2 appears twice in the result domain
$\{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a \} \cup \{ 4 \mapsto c \} = \{ 1 \mapsto a, 2 \mapsto b, 3 \mapsto a, 4 \mapsto c \}$	is a function because each first component is unique

EXERCISE 10.2

1 Evaluate:

a dom { 999 ↦ *emergency*, 112 ↦ *emergency*, 100 ↦ *operator*, 155 ↦ *international* }

b ran { 0800 ↦ *free*, 0808 ↦ *free*, 0845 ↦ 3*pPerMin*, 0870 ↦ 8*pPerMin* }

c { 061 ↦ *australia*, 091 ↦ *india*, 064 ↦ *newZealand*, 044 ↦ *uk* } ⩤ { 064, 061 }

d { *blair* ↦ *uk*, *bush* ↦ *USA*, *howard* ↦ *australia*, *clark* ↦ *newZealand* } ⩥ {*uk*, *australia*}

e { 1 ↦*mon*, 2 ↦ *tue*, 3 ↦ *wed*, 4 ↦ *thu*, 5 ↦ *fri* } ⦇ { 3 } ⦈

f { *tom* ↦ 25, *dee* ↦ 27, *harry* ↦ 24 } ⊕ { *dee* ↦ 25, *may* ↦ 23 }

g { *A* ↦ *distinction*, *B* ↦ *pass*, *C* ↦ *fail*, *X* ↦ *query* } ⨾
{ *distinction* ↦ 80, *pass* ↦ 40, *fail* ↦ 0 }

h { 061 ↦ *australia*, 091 ↦ *india*, 064 ↦ *newZealand*, 044 ↦ *uk* }~

i { 101 ↦ *UsingZ*, 103 ↦ *ZPractice*, 107 ↦ *ZWays* } ∩ { 105 ↦ *ZNotes* }

j { 101 ↦ *UsingZ*, 103 ↦ *ZPractice*, 107 ↦ *ZWays* } \ { 105 ↦ *ZNotes* }

k { 101 ↦ *UsingZ*, 103 ↦ *ZPractice*, 107 ↦ *ZWays* } ∪ { 105 ↦ *ZNotes* }

10.3 DECLARING A FUNCTION

We introduce as a given type *STUDENT*, the set of all possible students. We intend that no two students in a class have the same identity number.

We define *class* as a function from $\mathbb{Z}$ to *STUDENT*.

$$class : \mathbb{Z} \pfun STUDENT$$

The double vertical bars on the arrow mean that the function is *partial* - not the whole of $\mathbb{Z}$ is used, and that the function is *finite* - we can count the number of members in the class Because the function is finite we can apply the cardinality operator, #, and restrict the size of class to, say, between 0 and 20 students.

$$class : \mathbb{Z} \pfun STUDENT$$

$$\#class \geqslant 0$$
$$\#class \leqslant 20$$

An example class would be

$$class = \{ 1 \mapsto peter, 2 \mapsto paul, 3 \mapsto mary \}$$

The name of a function is usually (but not always) singular and chosen to reflect the range.

EXERCISE 10.3

1 Introduce appropriate types and declare the functions described below. Give an example of a set for each function.

a Each branch of every bank in the UK is uniquely identified by a six-digit sort code. Declare a function from sort code to bank branch.

b Every person has a biological mother. Declare a function from child to mother.

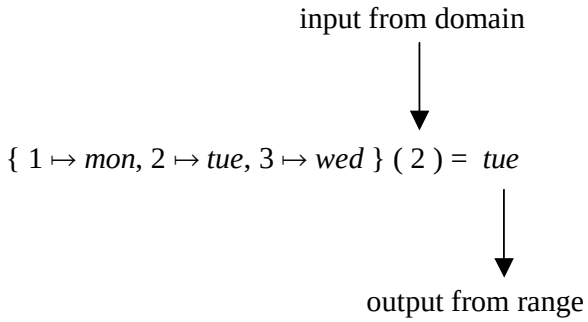
c In a library books are known as copies. There may be several copies of a book with the same title. Each copy has its own, unique number known as an accession number. Declare a function from accession number to copy.

10.4 FUNCTION APPLICATION

Because the relational image on a function must always must always give a set with just one element, we can use *function application* notation. For example:

$$\{ 1 \mapsto mon, 2 \mapsto tue, 3 \mapsto wed \} (2) = tue$$

We can think of function application in terms of input and output.



The input is an element from the set of all first components; the output is the corresponding second component.

A function always has just one output value for any input value. So

$$\{ 1 \mapsto mon, 2 \mapsto tue, 3 \mapsto wed \} (4)$$

is undefined because 4 is not in the domain of the function.

If *day* is a function as defined below:

$$day : \mathbb{Z} \pfun DAY$$

$$day = \{ 1 \mapsto mon, 2 \mapsto tue, 3 \mapsto wed \}$$

then we may write:

$$day\ (2) = tue$$

and

day (4) is undefined

EXERCISE 10.4

1 Given $menu = \{ f \mapsto file, e \mapsto edit, v \mapsto view, i \mapsto insert, o \mapsto format, t \mapsto tools \}$ evaluate:

a $menu(o)$

b $menu(h)$

2 Given $age = \{ steve \mapsto 23, chris \mapsto 21, sonu \mapsto 24 \}$ evaluate:

a $age \oplus \{ chris \mapsto age\ (chris) + 1 \}$

b $age \oplus \{ sam \mapsto age(sam) - 1 \}$

3 Given $rooms = \{ 1 \mapsto occupied, 2 \mapsto vacant, 3 \mapsto occupied, 5 \mapsto vacant, 6 \mapsto occupied \}$ evaluate:

a $\#rooms$

b $rooms \cup \{ 4 \mapsto empty \}$

c $rooms \oplus \{ 2 \mapsto occupied \}$

d $rooms \oplus \{ 1 \mapsto vacant \}$

e $rooms(5)$

f $rooms \setminus \{ 5 \mapsto rooms(5) \}$

10.5 USING FUNCTIONS

First, we set the scene. A hotel management system is used to maintain a record of the current state of the hotel rooms, whether they are occupied or not.

We introduce *ROOM*, the set of all rooms.

[*ROOM*]

We also introduce *STATUS* which describes whether a room is occupied or vacant.

STATUS ::= *occupied* | *vacant*

The hotel room system comprises a finite number of rooms. We intend that each room is uniquely identified, and that each room is either occupied or vacant.

Hotel

$room : ROOM \pfun STATUS$

We have no predicates defined at this stage.

Initally, there are no rooms.

InitHotel

Hotel

$room = \{\}$

We add a new room to the system. A new room is always vacant. We check that the room does not already exist and we add the new room to the system.

Commission

$\Delta Hotel$

$room? : ROOM$

$room? \notin \text{dom } room$

$room' = room \cup \{ room? \mapsto vacant \}$

When a guest arrives we inform the system that a room is now occupied. We check that the room exists and is not occupied, and we update our room records.

Occupy

$\Delta Hotel$

$room? : ROOM$

$room? \in \text{dom } room$

$room\ (room?) = vacant$

$room' = room \oplus \{ room? \mapsto occupied \}$

When a guest departs we inform the system that the room is now vacant. We check that the room exists and is occupied, and we update our room records.

Vacate

$\Delta Hotel$

$room? : ROOM$

$room? \in \text{dom } room$

$room\ (room?) = occupied$

$room' = room \oplus \{ room? \mapsto vacant \}$

We want to know how many rooms are occupied: we just count them.

QueryOccupiedRooms

$$\Xi Hotel$$
$$occupied! : \mathbb{Z}$$

$$occupied! = \# room$$

From time-to-time we decommission a room and make it unavailable to guests. We ensure that the room exists and is vacant, and we remove the empty room from the system.

Decommission

$$\Delta Hotel$$
$$room? : ROOM$$

$$room? \in \mathrm{dom}\ room$$
$$room\ (room?) = vacant$$
$$room' = room \setminus \{ room? \mapsto room\ (room?) \}$$

We define *REPORT*, whose values will be used to indicate the success (or otherwise) of our operations.

$$REPORT ::= ok \mid alreadyAdded \mid noSuchRoom \mid notEmpty \mid notOccupied$$

Success reports *ok*. We shall use it combined with our other operations.

Success

$$report! : REPORT$$

$$report! = ok$$

We cannot add the same room twice to our system. *DuplicateRoom* reports that the room has already been added.

DuplicateRoom

$\Xi Hotel$
room? : *ROOM*
report! : *REPORT*

---

room? $\in$ dom *room*
report! = *alreadyAdded*

We cannot place a guest in a room that does not exist, or remove a room that does not exist. *NoSuchRoom* reports that the room is not in our system.

NoSuchRoom

$\Xi Hotel$
room? : *ROOM*
report! : *REPORT*

---

room? $\notin$ dom *room*
report! = *noSuchRoom*

We cannot place a guest in a room that is already occupied by another guest. It is also an error to remove a room from the system with a guest still occupying it. *RoomNotEmpty* reports that the room is still occupied.

RoomNotEmpty

$\Xi Hotel$
room? : *ROOM*
report! : *REPORT*

---

room (*room*?) $\neq$ *vacant*
report! = *notEmpty*

We cannot remove a guest from a room that is not occupied. *RoomNotOccupied* reports that the room has no guests.

RoomNotOccupied

$\Xi Hotel$
room? : *ROOM*
report! : *REPORT*

---

room (*room*?) ≠ *occupied*
report! = *notOccupied*

Now we define our operations completely.

The complete commission process is *Commission* and *Succes,* or *DuplicateRoom*.

$$CommissionTot \mathrel{\widehat=} (Commission \land Success) \lor DuplicateRoom$$

The complete occupy process is *Occupy* and *Success,* or *NoSuchRoom,* or *RoomNotEmpty*.

$$OccupyTot \mathrel{\widehat=} (Occupy \land Success) \lor NoSuchRoom \lor RoomNotEmpty$$

The complete vacate process is *Vacate* and *Success*, or *NoSuchRoom,* or *RoomNotOccupied*.

$$VacateTot \mathrel{\widehat=} (Vacate \land Success) \lor NoSuchRoom \lor RoomNotOccupied$$

EXERCISE 10.5

1 Explain each line of Z, as if to a beginning student, for the Hotel Rooms speciifcation defined in section 10.5 above.

2 A college library loans out copies of books to its borrowers. Write and explain a Z specification for the Library Loans System described below.

Use Case	**AddBorrower**
Goal	To add a new borrower to the library system
Pre-condition	The borrower has not already been added to the system
Initiating Actor	Librarian

Main Success Scenario

1 Librarian inputs borrower
2 System confirms borrower is added to the system
3 Exit success

Exceptions

1a Borrower already a member of the library
1a1 Exit failure

Use Case	**AddCopy**
Goal	To add a new copy of a book to the library loans system
Pre-condition	The copy has not already been added
Initiating Actor	Librarian

Main Success Scenario

1 Librarian inputs copy information
2 System confirms copy is now added
3 Exit success

Exceptions

1a Copy is already in the library loans system
2a1 Exit failure

Use Case	**Issue**
Goal	To loan a copy of a book to a borrower
Pre-condition	The copy is part of the loan stock and the borrower is a member of the library and the borrower has not reached their lending limit and the copy is not already out on loan
Initiating Actor	Librarian

Main Success Scenario

1 Librarian inputs borrower
2 Librarian inputs copy
3 System confirms copy is loaned to borrower
4 Exit success

Exceptions

1a Borrower not a member of the library
1a1 Exit failure
1b Borrower has reached maximum number of loans allowed
1b1 Exit failure
2a Copy not part of loan stock
2a1 Exit failure
2b Copy out on loan
2b1 Exit failure

Use Case	**Return**
Goal	To discharge the loan of a copy to its borrower
Pre-condition	The copy is out on loan
Initiating Actor	Librarian

Main Success Scenario

1 Librarian inputs copy information
2 System confirms copy is now returned
3 Exit success

Exceptions

1a Copy is not out on loan
2a1 Exit failure

Use Case	**QueryWhoHasCopy**
Goal	To show who has borrowed a given copy
Pre-condition	The copy is part of the loan stock
Initiating Actor	Librarian

Main Success Scenario

1 Librarian inputs copy
2 System shows who has borrowed the copy
3 Exit success

Exceptions

1a Copy is not part of the loan stock
2a1 Exit failure

Use Case	**QueryBorrowersLoans**
Goal	To show the copies currently out on loan to a given borrower
Pre-condition	The borrower is a member of the library
Initiating Actor	Librarian

Main Success Scenario

1 Librarian inputs borrower
2 System shows copies out on loan to the given borrower
3 Exit success

Exceptions

1a The borrower is not a member of the library
2a1 Exit failure

REVIEW

We explored the concept of a function - a binary relation in which all the first components of each pair are unique. We saw that what applied to binary relations could also be applied to functions - but with care. We saw how to define functions, how to get answers from functions and how to use functions. We used override to update a function. We used union to extend a function.

BIBLIOGRAPHY

BARDEN R., STEPNEY S. & COOPER D. 1994 *Z in Practic*e Prentice Hall Hemel Hempstead UK pp 168, 179

JACKY J. 1997 *The Way of Z* Cambridge University Press Cambridge UK pp 36, 88, 90, 268

SPIVEY J.M. 1992 *The Z Notation* Prentice Hall Hemel Hempstead UK pp 27, 60, 107