CHAPTER 2

Sets

2.1 Types

A *set* is a collection of components called *elements* or *members*. The Z notation uses *typed* set theory: all the possible values of a set are considered to have something in common, and are said to have the same *type*. For example, we will have sets of *persons*, or sets of *numbers*, but not just sets that can contain *any* sort of element. Any set is therefore considered to be a *subset* of its type. A *subset* is any collection of values from a set. For example, the even numbers are a subset of the integer numbers.

The notion of type helps in two ways:

- it avoids certain mathematical paradoxes;
- it allows checks to be made that statements about sets make sense.

The checks can be automated by means of computer *software tools* which check the consistency of the mathematical text in a Z document in the same sort of way that a spelling and grammar checker checks the consistency of a document in a natural language.

2.2 The built-in type Integer

The built-in type *integer* can be used in any Z document without the need to introduce it explicitly. It is designated by the symbol

$$\mathbb{Z}$$

and consists of the whole numbers

$$\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$$

The symbol is sometimes hand-written 'ZZ' and it is pronounced 'integer' or 'zed-zed' or 'fat Z'. It is an example of an *infinite* set.

2.2.1 The standard set Natural

Often, when specifying, we know that a number can never be negative. (For example, someone's age, or the number of elements in a set). To highlight this we can indicate that it is a *natural* number. The *set natural* is a standard subset of the *type* integer that adds the constraint that the value

must always be non-negative. Because it is so useful it has a standard designation:

$\mathbb{N}$

It is pronounced 'natural' or 'en-en' or 'fat N'.

In some mathematical contexts the natural numbers are not considered to include zero. The set of natural numbers *excluding zero* can be written as follows:

$\mathbb{N}_1$

2.2.2 Operations on integers

The following operators are defined for the type integer and its subsets:

+	addition
–	subtraction
*	multiplication
div	(integer) division
mod	modulus (remainder after division)

Examples

23 div 5 = 4
23 mod 5 = 3

The normal rules of precedence between operators hold. For example:

5 + 4 * 3 is the same as 5 + (4 * 3)

because the operators *, *div* and *mod* have *higher precedence* (which means that they bind more tightly) than + and –.

2.2.3 Numerical relations

The following relational operators are applicable to integers:

=	equal to
≠	not equal to
<	less than
≤	less than or equal to
>	greater than
≥	greater than or equal to

Note the following:

x = y is the opposite of x ≠ y
x < y is the opposite of x ≥ y
x ≤ y is the opposite of x > y
x > y is the opposite of x ≤ y
x ≥ y is the opposite of x < y

2.3 Basic types

Basic types are also called *given types*. The basic types of a specification are declared without concern for how their actual elements are to be represented. For example, a specification might refer to the set of all possible car registrations without considering how such registrations are represented as characters. A basic type is declared by writing its name in square brackets, with a comment to indicate its intended meaning. The set of car registrations might be called *REGISTRATION* and written:

```
[REGISTRATION]          the set of all possible car registrations
```

By convention, the name of a basic type is written entirely in capital letters and a singular noun is used.

Another basic type might be the set of all persons:

```
[PERSON]                the set of all persons
```

Several types can be given in one line:

```
[REGISTRATION, PERSON]
```

In general, basic types should be chosen to be as widely encompassing as possible. Furthermore, it should be assumed that the elements of the type are uniquely identifiable.

2.4 Free types

Sometimes it is convenient to introduce a type by listing the identifiers of its elements. This can be done with a *free type*. The general format is:

$$freeType ::= element_1 \mid element_2 \mid \ldots \mid element_n$$

Examples

```
RESPONSE ::= yes | no
STATUS ::= inUse | free | onHold | outOfOrder
```

2.5 Declaring variables

Each variable name designating a value must be declared. That means it must be introduced and the type of value it refers to must be stated. For example, to introduce a variable *chauffeur* to be of the basic type *PERSON* we write:

```
chauffeur: PERSON
```

This can be pronounced '*chauffeur* is one of the set of values *PERSON*' or '*chauffeur* is drawn from the set *PERSON*' or '*chauffeur* is a *PERSON*'.

In the following examples elements will be drawn from the set *EU*, the set of all countries in the European Union. To allow for membership to change we should declare this as a *basic* type, but we declare it as a free type here for the sake of illustration since it contains only fifteen members, and for the convenience of identifying the elements by their international car registration letters:

Austria	A	Ireland	IRL
Belgium	B	Italy	I
Denmark	DK	Luxembourg	L
France	F	Netherlands	NL
Finland	SF	Portugal	P
Germany	D	Spain	E
Great Britain	GB	Sweden	S
Greece	GR		

EU ::= A | B | DK | F | SF | D | GB | GR | IRL | I | L | NL | P | E | S
the set of countries currently in the European Union

2.6 Single value from a type

To introduce a *variable* called *homeland* which can refer to *one* country in the European Union we would write:

homeland: EU

2.7 Set values

The value of a set can be written by listing its values within *braces* ('curly brackets'):

benelux = {B, NL, L}

The order in which the elements appear does not matter:

{B, NL, L} = {NL, B, L} = { L, B, NL} = {B, L, NL} = {NL, L, B} = {L, NL, B}

Repeating a value does not matter; although it may appear in the list of values several times it can only occur in a *set* only once:

{B, NL, L, NL} = {B, NL, L}

2.8 The empty set

It is possible to have a set with no values; it is called the *empty set*:

∅

The empty set can also be written as empty braces:

{}

2.9 Singleton set

A set that contains only one element is called a *singleton* set. For example:

{GB}

Note that a set containing a single element has different type from the element itself: {*GB*} does not have the same type as *GB*. {*GB*} is the *set* of countries containing just Great Britain, but *GB* is a country, not a set.

2.10 Ranges of integers

The range of values

m .. n

where *m* and *n* are integers, stands for the set of integers *m* to *n* inclusive. Note that if

m > n

then

m .. n = Ø

Examples

3 .. 5 = {3, 4, 5}
2 .. 2 = {2}
3 .. 2 = Ø

2.11 Operators

2.11.1 Equivalence

Two values of the same type can be tested to see if they are the same by using the equals sign, as in

x = y

Two sets are equal if they contain exactly the same elements.

Example

{B, NL, L} = {NL, B, L} these sets contain the same elements

2.11.2 Non-equivalence

Similarly, two values of the same type can be tested to see if they are *not* the same by using the not-equals sign, as in

x ≠ y

Two sets are not-equal if they do not contain exactly the same elements.

Example

$\{B, NL\} \neq \{B, NL, L\}$ these sets contain different elements

2.11.3 Membership

The *membership* operator is written:

$\in$

and is pronounced 'is an element of' or 'is a member of'. The expression involving it is true if the value is an element (member) of the set and false otherwise.

Example

$NL \in \{B, NL, L\}$ this is true; the Netherlands is in 'Benelux'

('Benelux' is a name sometimes used for the group of European countries consisting of Belgium, the Netherlands and Luxembourg.)

The diagrams in this chapter used to illustrate the set operators are called *Venn* diagrams. Figure 2.1 shows the set *S*, a subset of some type *X*. The element *x* of type *X* is a member of the set *S*:

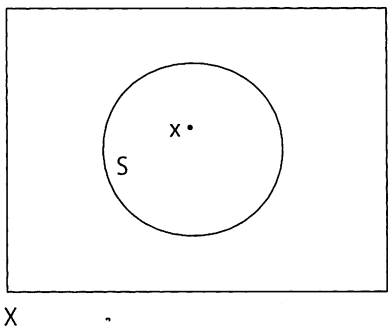
Figure 2.1

$[X]$

$S: \mathbb{P}\, X$

$x: X$

$x \in S$



2.11.4 Non-membership

The *non-membership* operator is written

$\notin$

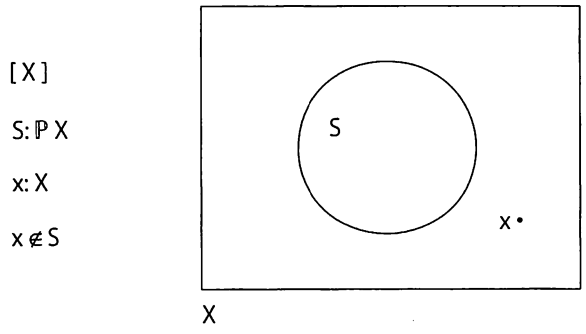
and is pronounced 'is not an element of' or 'is not a member of'. The expression involving it is true if the value is *not* an element (member) of the set and false otherwise.

Example

$GB \notin \{B, NL, L\}$ this is true; GB is not a member of Benelux

Figure 2.2 shows the set *S*, a subset of some type *X*. The element *x* of type *X* is *not* a member of the set *S*:

Figure 2.2



2.11.5 Validity of membership test

Note that the value to be tested for membership must be an element of the underlying type of the set. For example, the expression:

USA ∈ {B, NL, L}

is neither true nor false but *illegal*, since *USA* is not an element of the type *EU*.

2.11.6 Size, cardinality

The number of values in a set is called its *size*, or *cardinality*, and is signified by the *hash* sign:

# {B, NL, L}	= 3
# {GB}	= 1
# GB	illegal, GB is not a set
# Ø	= 0

2.11.7 Powersets

The *powerset* of a set *S* is written

ℙS

and is the set of all its subsets. For example, the subsets of Benelux are:

ℙ {B, NL, L} =	
{ Ø,	the empty set
{B}, {NL}, {L},	all the singletons
{B, NL}, {B, L}, {NL, L},	all the pairs
{B, NL, L}}	the three elements

When a variable is to be declared to have a type that is *set* of elements, the type is the powerset of the type of the elements:

benelux: ℙEU

This can be read '*benelux* is a *subset* of the set of countries *EU*', or '*benelux* is a set of *EU* countries'.

Note that the size of the powerset of a set is equal to two raised to the power of the size of the set:

# (ℙS)	$= 2^{\#S}$ (for any set S)
# {B, NL, L}	= 3
# (ℙ {B, NL, L})	= 8

2.11.8 Set inclusion

The operator

⊆

is pronounced 'is included in' or 'is contained in' or 'is a subset of' and tests whether the first set is included in the second set; whether the first is a subset of the second. Every element of the first set is also an element of the second.

In Figure 2.3 the sets *S* and *T* are subsets of some type *X*. *T* is a subset of *S*:

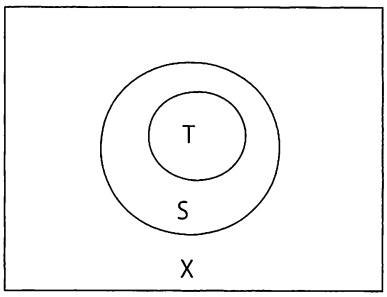
Figure 2.3

[X]

S: ℙ X

T: ℙ X

T ⊆ S



{B, NL} ⊆ {B, NL, L}	this is true
Ø ⊆ {B, NL, L}	this is true
{B, NL, L} ⊆ {B, NL, L}	this is true

The empty set is a *subset* of every set, including itself. But note that the empty set is not a *member* of every set. (The empty set can only be a member of a *set* of *sets*). An example of an empty set of European Union countries is the set of those countries bordering the Pacific Ocean (that is, none of them).

Ø ⊆ S	this is true for any set S
Ø ⊆ Ø	empty set is a subset of empty set

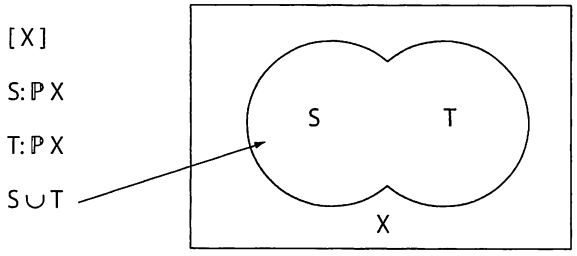
Note that testing for the inclusion of a singleton set in another set is the same as testing for that singleton set's element's membership in the set:

{x} ⊆ S is the same as x ∈ S

2.11.9 Union

The *union* of two sets is the set containing all the elements that are in *either* the first set *or* the second set *or* both. The union symbol is sometimes pronounced 'cup'. In Figure 2.4 S and T are both subsets of some type X. The shaded area is the union of S and T:

Figure 2.4



$S \cup T = T \cup S$ for any sets S and T; ('commutative')
$S \cup \emptyset = S$ for any set S

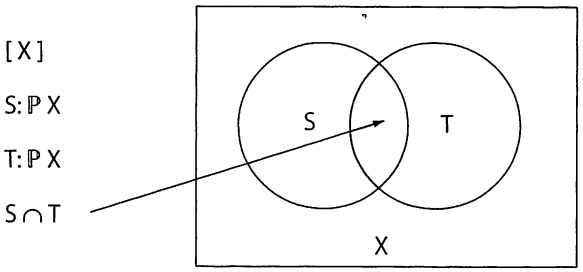
Examples

{B, D, DK} ∪ {D, DK, F, I} = {B, D, DK, F, I}
{B, D, DK} ∪ {DK, F, I} = {B, D, DK, F, I}
{B, NL, L} ∪ {GB, IRL} = {B, NL, L, GB, IRL}
{B, D, DK} ∪ Ø = {B, D, DK}

2.11.10 Intersection

The *intersection* of two sets is the set containing all the elements that are in both the first set and the second set. The intersection symbol is sometimes pronounced 'cap'. In Figure 2.5 S and T are both subsets of some type X. The area pointed to is the intersection of S and T:

Figure 2.5



$S \cap T = T \cap S$ for any sets S and T; ('commutative')
$S \cap \emptyset = \emptyset$ for any set S

Examples

{B, D, DK} ∩ {D, DK, F, I} = {D, DK}
Germany and Denmark are in both sets

$\{B, D, DK\} \cap \{DK, F, I\} = \{DK\}$
only Denmark is in both sets. Note {DK}, not just DK

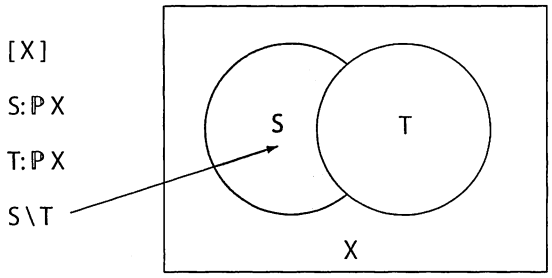
$\{B, NL, L\} \cap \{GB, IRL\} = \emptyset$
No country is in both sets

$\{B, D, DK\} \cap \emptyset = \emptyset$

2.11.11 Difference

The *difference* of two sets is the set containing all those elements of the first set that are *not* in the second set. It is as if the first set is 'eclipsed' by the second. In Figure 2.6 S and T are both subsets of some type X. The shaded area is the difference of S and T:

Figure 2.6



$S \setminus T \neq T \setminus S$	for any sets S and T (in general not commutative)
$S \setminus \emptyset = S$	for any set S
$\emptyset \setminus S = \emptyset$	for any set S

Example

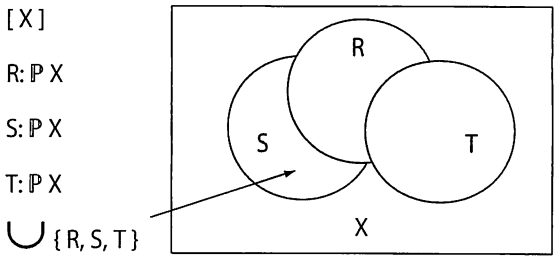
$\{B, D, DK, F, I\} \setminus \{B, D, GR\} = \{DK, F, I\}$
$\{B, D, DK\} \setminus \emptyset = \{B, D, DK\}$

2.11.12 Distributed union

The operations described so far have been applied to *two* sets. Sometimes it is useful to be able to refer to the union of several sets; in fact, of a *set* of sets. This can be done with the *distributed union* operator, written as an oversized union operator sign, which applies to a set of sets and results in a set. The distributed union of a set of sets is the set containing just those elements that occur in *at least one of* the component sets.

In Figure 2.7 R, S and T are all subsets of some type X. The shaded area is the distributed union of R, S and T:

Figure 2.7



Example

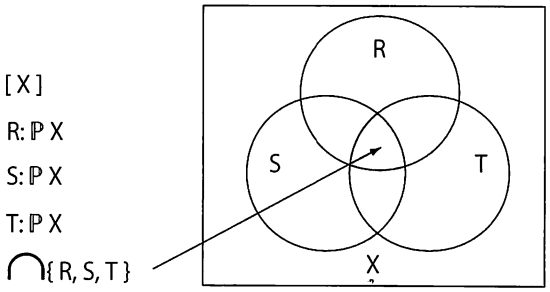
$$\bigcup \{\{B, NL, L\}, \{F, D, L, I\}, \{GB, GR, B, IRL, DK, E, P\}\}$$
$$= \{B, NL, L, F, D, I, GB, GR, IRL, DK, E, P\}$$

2.11.13 Distributed intersection

The *distributed intersection* of a set of sets is the set of elements which are in all of the component sets. The distributed intersection of a set of sets is the set containing just those elements that occur in *all of* the component sets.

In Figure 2.8 R, S and T are all subsets of some type X. The small area pointed to is the distributed intersection of R, S and T:

Figure 2.8



Example

$$\bigcap \{\{B, NL, L, GB\}, \{F, NL, D, L, GB, I\}, \{GB, GR, IRL, DK, NL, P\}\}$$
$$= \{GB, NL\}$$

2.12 Disjoint sets

Sets that are *disjoint* have no elements in common; their intersection is the empty set. This is easily expressed for two sets S and T

$$S \cap T = \emptyset$$

For more than two sets it becomes lengthier, since every *pair* of sets must have empty intersections. For example for sets A, B and C to be disjoint:

$$A \cap B = \emptyset \text{ and}$$
$$B \cap C = \emptyset \text{ and}$$
$$C \cap A = \emptyset$$

In general we can write:

disjoint ⟨S, T⟩
disjoint ⟨A, B, C⟩

[PERSON]
female, male: ℙPERSON

disjoint ⟨female, male⟩

The brackets here are *sequence brackets.* Sequences will be covered in Chapter 12.

2.13 Partition

A *sequence* of sets is said to *partition* another larger set if the sets are disjoint and their distributed union is the entire larger set. For example, the sets A, B and C partition T if:

disjoint ⟨A, B, C⟩

and

$$\bigcup \{ A, B, C \} = T$$

This can be written:

⟨A, B, C⟩ partition T

For example:

⟨female, male⟩ partition PERSON

2.14 Summary of notation

$\mathbb{Z}$	the *type* integer (the set of all whole numbers)
$\mathbb{N}$	the *set* of natural numbers ({0, 1, 2, ...})
$\mathbb{N}_1$	the *set* of *positive* natural numbers ({1, 2, ...})
$t \in S$	t is an element of S
$t \notin S$	t is not an element of S
$S \subseteq T$	S is contained in T
$\emptyset$ or $\{\}$	the empty set
$\{t_1, t_2, \ldots t_n\}$	the set containing $t_1, t_2, \ldots t_n$

$\mathbb{P}S$	Powerset: the set of all subsets of S
$S \cup T$	Union: elements that are either in S or T or both
$S \cap T$	Intersection: elements that are both in S and in T
$S \setminus T$	Difference: elements that are in S but not in T
$\#S$	Size (cardinality): the number of elements in S
$m \,..\, n$	the set of integers m to n inclusive
$\bigcup SS$	the distributed union of the set of sets SS
$\bigcap SS$	the distributed intersection of the set of sets SS
disjoint sqs	the sets in the sequence sqs are disjoint
sqs partition S	the sets in sqs partition S

EXERCISES

1. Certain people are registered as users of a computer system. At any given time, some of these users are 'logged-in' to the computer. Describe this situation using the concepts of Z covered so far.
2. Extend your description from Question 1 as follows:

   There is a limit (unspecified) to the number of users logged-in at any time.
3. Extend your description from Question 1 as follows:

   All users are either staff users or customer users.
4. Express the following statements using Z notations:

   All the currently logged-in users are staff.

   There are more customer users than staff users.
5. In a modular university course some modules are *acceptable* and others are *compulsory*. Use the names *acceptables* and *compulsories* for the sets of modules. Each student studies modules from two *fields*.

   The acceptable modules for the first field are called *firstAcc* and for the second *secondAcc*:

   $$firstAcc \subseteq acceptables$$
   $$secondAcc \subseteq acceptables$$

   (a) Write an expression to state that all compulsory modules are also acceptable.
   (b) Write an expression to state that there are three compulsory modules.
   (c) Write an expression to state that the acceptables for the first field are not the same as the acceptables for the second field.
   (d) Write an expression to state that some modules are acceptable for both the first field and the second field.