Chapter 9

Constructed types

In earlier chapters we used basic mathematical types for our variables, all of them numbers of various kinds. In this chapter we are more ambitious, and expand our repertoire considerably by using the types we have already to make other types, and those to make others still. Our tools are powersets, bags, sequences, functions and relations.

9.1 Powersets

9.1.1 Making powersets

Given any type we can form its *powerset*, the type that contains all its subsets: given a type T, the type **set** T has as elements all subsets of T. Thus values of the type **set** $\mathbb{N}$ include $\{\}$, $\{17\}$, $\{2, 6\}$, the set of non-negative even numbers $\{0, 2, 4, \cdots\}$ and of course $\mathbb{N}$ itself.

As a special case, the type **finset** T has as elements all *finite* subsets of T. (Thus $\mathbb{N} \notin$ **finset** $\mathbb{N}$.)

That's all there is to making powersets; but when we introduce a new type, or type construction, we must also decide how to describe individual elements of the type and what operators will be available to use with them.

9.1.2 Using powersets: set enumeration and operators

Finite sets can be written by giving their elements explicitly, by *enumeration* between set brackets $\{\cdots\}$. For example, the set $\{1, 2, 3\}$ contains three elements exactly: 1, 2, and 3. The order of elements does not matter in set enumerations ($\{3, 2, 1\}$ is the same as $\{1, 2, 3\}$); and if by chance a value is written more than once, it makes no difference ($\{1, 2, 2, 3\}$ is still the same).

$\cup$	union	functions
$\cap$	intersection	
$-$	(set) subtraction	
$\times$	Cartesian product	
$\in$	membership	relations
$\subseteq$	inclusion	
$\subset$	strict inclusion	
$\#$	cardinality	(a function to $\mathbb{N}$)

Figure 9.1 Basic set operators

Set enumerations cannot describe *infinite* sets, except informally, because we cannot write all the elements down: although we might say that the set of even numbers is

$$\{0, 2, 4, \cdots\} \ ,$$

we cannot really give '$\cdots$' any precise meaning in general.

Our numeric types have operators like $+$ and $-$, and our set types have their own operators, like $\cup$ (union) and $\cap$ (intersection). Those are what we use in terms to combine given sets to form others. Figure 9.1 gives a selection of set operators.

9.1.3 Set comprehension

Set *comprehensions* define sets by some characteristic of their elements, rather than by writing them out, and they apply equally well to both finite and infinite sets. For example, the even numbers are those natural numbers n for each of which there is another natural number m with $n = 2m$:

$$\{n : \mathbb{N} \mid (\exists\, m : \mathbb{N} \cdot n = 2m)\} \ .$$

The general set comprehension has three parts (although the example above had just two). The first is a list of *bound variables* and their types. In the above, that list contains just one variable n and its type $\mathbb{N}$. If there are several variables and types, they are separated by a semicolon, for example $m : \mathbb{N};\ \ n : \mathbb{N}^+$. (As in variable declarations, a repeated type may be omitted: thus we can write $m, n : \mathbb{N}$ for $m : \mathbb{N};\ \ n : \mathbb{N}$.)

The set is formed by allowing the bound variables to range over their types.

The second part is a formula, called the *range*; in the comprehension above, it is $(\exists\, m : \mathbb{N} \cdot n = 2m)$. The formula usually refers to bound variables of the

$$
\begin{array}{rcl}
\{n : \mathbb{N} \mid n > 0\} & = & \{1, 2, 3, \cdots\} = \mathbb{N}^+ \\
\{n : \mathbb{N} \mid n < 0\} & = & \{\} \\
\{m, n : \mathbb{N} \mid m < n\} & = & \{(0,1), (0,2), \cdots, (1,2), (1,3), \cdots\} \\
\{n, m : \mathbb{N} \mid m < n\} & = & \{(1,0), (2,0), \cdots, (2,1), (3,1), \cdots\} \\
\{n : \mathbb{N} \bullet -n\} & = & \{0, -1, -2, \cdots\} \\
\{m, n : \mathbb{N} \bullet m + n\} & = & \{0, 1, 2, \cdots\} = \mathbb{N} \\
\{m, n : \mathbb{N} \mid m \geq n \bullet m^2 - n^2\} & = & \{0, 1, 3, 4, 5, 7, \cdots\} \\
\{m : \mathbb{N};\;\; n : \{0, 1, 3\} \bullet 4m + n\} & = & \{0, 1, 3, 4, 5, 7, \cdots\}
\end{array}
$$

Figure 9.2 Set comprehensions

comprehension (but need not). It can also refer to other variables, in which case the set formed depends on the value of those. In any case, only values that satisfy the formula are considered while the bound variables range over their types; other values are just ignored. Sometimes the formula is just the formula `true`, in which case it (and the $|$, pronounced 'such that') can be left out.

The third part of the comprehension is a *term*. For each possible value of the bound variables, it is the value of the term that is put into the set. In the above, the term is left out because it is just the bound variable n itself. In general, a missing term is understood to be the *tuple* formed from the bound variables taken in order. (A 'monotuple', containing just one component, is just the component itself.)

If the term is present, it is preceded by a spot $\bullet$ (pronounced 'make'). Here is another definition of the set of even numbers:

$$\{m : \mathbb{N} \bullet 2m\} \ .$$

Figure 9.2 gives further examples.

9.1.4 Promoted relations

If we have a type T with some relation $\odot$ on its elements, we can use the same symbol $\odot$ for a *promoted* relation between sets $s1, s2 : \textbf{set}\ T$ as follows:

$$s1 \odot s2 \ \widehat{=}\ (\forall\, t1 : s1;\ t2 : s2 \bullet t1 \odot t2) \ .$$

And given a single element $t : T$ we further define

$$
\begin{array}{lcl}
t \odot s2 & \widehat{=} & (\forall\, t2 : s2 \bullet t \odot t2) \\
s1 \odot t & \widehat{=} & (\forall\, t1 : s1 \bullet t1 \odot t) \ .
\end{array}
$$

The convenience of promotion usually outweighs the danger of confusing the two relations denoted by the symbol.

Often the relation $\odot$ is a total order, and the promoted relation allows formulae of this kind:

1. $s1 < s2$: every element of $s1$ is less than every element of $s2$.
2. $n \in s \wedge n \leq s$: the minimum value in s is n.

One must be especially careful, however, with transitivity: a promoted relation is not necessarily transitive, even if it is based on a transitive relation: from $1 < \{\}$ and $\{\} < 0$ (both true) we cannot conclude that $1 < 0$. (See Exercise 9.3.)

9.2 Bags

9.2.1 Bag enumeration and operators

A *bag*, like a set, is a collection of elements. Unlike a set, however, an element can belong to a bag 'more than once'. Given a type T, the type of all bags of its elements is written **bag** T.

For bag b and element e, we write $b.e$ for the number of times e occurs in b, and the formula $e \in b$ is true if and only if $b.e \neq 0$. Traditional set operators carry across to bags:

$$\begin{array}{rcl} (b1 \cup b2).e & = & b1.e \sqcup b2.e \\ (b1 \cap b2).e & = & b1.e \sqcap b2.e \\ (b1 - b2).e & = & (b1.e - b2.e) \sqcup 0 \ . \end{array}$$

As well, there is a new operation of *bag addition*:

$$(b1 + b2).e \ \widehat{=} \ b1.e + b2.e \ .$$

Like sets, bags can be explicitly enumerated; the elements are written between bag brackets $\llbracket$ and $\rrbracket$. Unlike set enumerations, bag enumerations are sensitive to how many times an element is written: if it is written twice, then it occurs twice in the bag. But order is still ignored. Figure 9.3 gives examples of bags and bag operations.

9.2.2 Conversion between bags and sets

The function set converts a bag to a set by 'forgetting' multiplicity:

$$\mathsf{set} : \mathbf{bag}\ T \rightarrow \mathbf{set}\ T \ .$$

For example, $\mathsf{set}\llbracket 1,2,2,3 \rrbracket = \{1,2,3\}$. In general, for $s = \mathsf{set}\, b$ we have that $e \in s$ iff $e \in b$.

The function bag_n goes the other way, converting a set into a bag:

$$\mathsf{bag} : \mathbb{N} \rightarrow \mathbf{set}\ T \rightarrow \mathbf{bag}\ T \ .$$

$$
\begin{array}{rcl}
\llbracket 1,2,2,3 \rrbracket.1 & = & 1 \\
\llbracket 1,2,2,3 \rrbracket.2 & = & 2 \\
\llbracket 1,2,2,3 \rrbracket.3 & = & 1 \\
\llbracket 1,2,2,3 \rrbracket.4 & = & 0 \\
\llbracket 1 \rrbracket \cup \llbracket 1 \rrbracket & = & \llbracket 1 \rrbracket \\
\llbracket 1 \rrbracket + \llbracket 1 \rrbracket & = & \llbracket 1,1 \rrbracket \\
\llbracket 1 \rrbracket - \llbracket 1,1 \rrbracket & = & \llbracket\ \rrbracket \\
\llbracket 1,1 \rrbracket - \llbracket 1 \rrbracket & = & \llbracket 1 \rrbracket
\end{array}
$$

Figure 9.3 Examples of bags and bag operations

$$
\begin{array}{rcl}
\llbracket n : \llbracket 1,2,2,3 \rrbracket \cdot 2n \rrbracket & = & \llbracket 2,4,4,6 \rrbracket \\
\llbracket m,n : \mathbb{N} \cdot m+n \rrbracket & = & \llbracket 0,1,1,2,2,2,\cdots \rrbracket \\
\llbracket m,n : \mathbb{N} \mid m<n \cdot m+n \rrbracket & = & \llbracket 1,2,3,3,4,4,\cdots \rrbracket \\
\llbracket m,n : \mathbb{N} \mid m<n \cdot n \rrbracket & = & \llbracket 1,2,2,3,3,3,4,4,4,4,\cdots \rrbracket
\end{array}
$$

Figure 9.4 Examples of bag comprehensions

Each element in the set is given multiplicity n in the resulting bag; if n is omitted, it is taken to be 1. For example, $\mathsf{bag}_2\ \{1,2\} = \llbracket 1,1,2,2 \rrbracket$. Whenever $b = \mathsf{bag}_n\ s$, we have that $b.e = n$ if $e \in s$, and $b.e = 0$ if $e \notin s$. Finally, the two functions are complementary in the sense that for any set s, $\mathsf{set}\,\mathsf{bag}\,s = s$.

Promoted relations between bags are available as for sets.

9.2.3 Bag comprehension

Bag comprehensions, like set comprehensions, define bags by some characteristic of their elements; the difference is that they are written between bag brackets instead of set brackets. Unlike set comprehensions, the types of the bound variables are bags themselves. (If they are written as sets, then they are first implicitly converted to bags by the function bag.)

Multiplicity in bag comprehensions can arise in two ways. First, particular values of bound variables can occur more than once (since they are themselves taken from bags). Second, if the comprehension makes a term, it is possible for the same value to result from evaluations of the term with *different* values for the bound variables: thus $\llbracket n\colon \llbracket -1,0,1 \rrbracket \cdot n^2 \rrbracket = \llbracket 1,0,1 \rrbracket$.

Figure 9.4 gives examples of bag comprehensions.

9.3 Sequences

9.3.1 Sequence enumerations and operations

A *sequence* is a collection of elements in which the order (and multiplicity) is significant. Given a type T, the type **seq** T has as elements all finite sequences of elements of T (of any length, including 0). For the type of all sequences of T with fixed length L we write $\mathbf{seq}_L\, T$. As a special case we use $\mathbf{seq}_\infty\, T$ for the type of all strictly infinite sequences of T.

Sequence enumerations are written between the sequence brackets $\langle$ and $\rangle$. For sequence q and natural number n, we write $q[n]$ for the element occupying the n^{th} position of q, with the first position being index 0.

For sequences of integers, we allow a special ellipsis notation: the term $m{\rightarrow}n$ denotes the sequence starting at m and ending *just before* n. Thus $1{\rightarrow}4$ is the same as $\langle 1, 2, 3\rangle$.

The principal operations on sequences are cons ':', concatenation '++', head hd, tail tl, front fr, last lt, and length '#'. They are summarised in Figure 9.5, and Figure 9.6 gives examples of their use.

9.3.2 Conversions between sequences, bags, and sets

The function $\mathsf{seq}_\leq$ converts a bag or set to a sequence whose elements are ascending in the total order $\leq$: if a occurs before b in the sequence, then $a \leq b$. If the order $\leq$ is omitted, it is understood to be the standard order on the element type of the bag or set. For $\mathsf{seq}_\leq$ to be well-defined, the order $\leq$ should be such that every non-empty set has a least element with respect to the order.

If an element occurs n times in the bag, it is repeated n times in the sequence; when seq is applied to a set, however, each element occurs exactly once in the resulting sequence.

The function bag takes sequences to bags (as well as sets to bags), and the multiplicity in the sequence is preserved. Similarly, the function set takes a sequence to the set of its elements (in which case the multiplicity is not preserved).

Those functions too are complementary: for any set s and bag b,

$$\begin{array}{rcl} \mathsf{bag}\,\mathsf{seq}_\leq\; b & = & b \\ \mathsf{set}\,\mathsf{seq}_\leq\; s & = & s\,. \end{array}$$

We may omit the conversion functions altogether if their use is implicit in the context, and that allows some compact (and possibly confusing) idioms. Some examples are given in Figure 9.7.

Promoted relations are available between sequences as for sets and bags. (That follows in fact from the implicit conversion convention: the sequences are converted to sets first.)

$\#q$	The number of elements in q.
$e{:}q$	The sequence whose first element is e, and whose subsequent elements are those of q. We have $(e{:}q)[0] = e$ and for $0 < i \leq \#q$, $(e{:}q)[i] = q[i-1]$.
$q1 \mathbin{+\!\!+} q2$	The sequence that begins with $q1$ and carries on with $q2$. We have $(q1 \mathbin{+\!\!+} q2)[i]$ equals $q1[i]$, if $0 \leq i < \#q1$, and equals $q2[i - \#q1]$ if $0 \leq i - \#q1 < \#q2$.
$\mathsf{hd}\ q$	The first element of q, provided q is not empty. We have $\mathsf{hd}(\langle e\rangle \mathbin{+\!\!+} q) = e$.
$\mathsf{tl}\ q$	The second and subsequent elements of q, provided q is not empty. We have $\mathsf{tl}(\langle e\rangle \mathbin{+\!\!+} q) = q$.
$\mathsf{fr}\ q$	All but the last element of q, provided q is not empty. We have $\mathsf{fr}(q \mathbin{+\!\!+} \langle e\rangle) = q$.
$\mathsf{lt}\ q$	The last element of q, provided q is not empty. We have $\mathsf{lt}(q \mathbin{+\!\!+} \langle e\rangle) = e$.

Figure 9.5 Operations on sequences: definitions

$$
\begin{array}{rcl}
\#\langle\rangle & = & 0 \\
\langle 1,2\rangle \mathbin{+\!\!+} \langle\rangle & = & \langle 1,2\rangle \\
\langle\rangle \mathbin{+\!\!+} \langle 1,2\rangle & = & \langle 1,2\rangle \\
1{:}\langle 2,3\rangle & = & \langle 1,2,3\rangle \\
\langle 1\rangle \mathbin{+\!\!+} \langle 2,3\rangle & = & \langle 1,2,3\rangle \\
\mathsf{hd}\langle 1,2,3\rangle & = & 1 \\
\mathsf{tl}\langle 1,2,3\rangle & = & \langle 2,3\rangle \\
\mathsf{fr}\langle 1,2,3\rangle & = & \langle 1,2\rangle \\
\mathsf{lt}\langle 1,2,3\rangle & = & 3
\end{array}
$$

Figure 9.6 Examples of operations on sequences

9.3.3 Sequence comprehension

Sequence comprehensions define sequences by some characteristic of their elements. The bound variables should be sequences themselves (though implicit conversion

$e \in q$	e occurs in the sequence q.
$q.e$	the number of times e occurs in sequence q.
$s[n]$	the n^{th}-from-least element of the set s.
$b[n]$	the n^{th}-from-least element of the bag b.
$q = \mathsf{bag}\ q$	the sequence q is in order.
$\mathsf{bag}\ q = \mathsf{set}\ q$	the sequence is without repetition.
$b = \mathsf{set}\ b$	the bag is without repetition.

Note that in the final three cases, it is the right-hand side that is converted implicitly, rather than the left (though either would achieve type compatibility). Where there is a choice, we take the conversion that *adds* information (for example, from bag to sequence).

Figure 9.7 Implicit conversion idioms

$$
\begin{array}{rcl}
\langle n : \mathbb{N} \cdot 2n \rangle & = & \langle 0, 2, 4, \cdots \rangle \\
\langle i : 0 \rightarrow 10 \mid i^2 > 50 \rangle & = & \langle 8, 9 \rangle \\
\langle i, j : \mathbb{N} \mid j < i \rangle & = & \langle (1,0), (2,0), (2,1), (3,0), \cdots \rangle \\
\langle n : \mathbb{N} \cdot n^2 \rangle & = & \langle 0, 1, 4, \cdots \rangle \ (= sq, \text{ say}) \\
\langle n : \mathbb{N} \cdot n^3 \rangle & = & \langle 0, 1, 8, \cdots \rangle \ (= cb, \text{ say}) \\
\langle i : sq;\ \ j : cb \mid i = j \cdot i \rangle & = & \langle 0, 1, 64, 729, \cdots \rangle
\end{array}
$$

Figure 9.8 Sequence comprehensions

may operate, if the element types have standard orders); then the order of the resulting sequence is determined by taking the values of the bound variables, in order, from their types. If there are several bound variables, then the rightmost varies fastest. Figure 9.8 gives examples of sequence comprehensions.

Sequences of sequences allow multi-dimensional structures, and for $q[i][j]$ we allow the abbreviation $q[i,j]$. The 'i^{th} row' of q is just $q[i]$. The 'j^{th} column' of q is $\langle i : 0 \rightarrow \#q \cdot q[i,j] \rangle$.

9.3.4 Sequence idioms

We list below some convenient operations on sequences that can be defined by comprehension.

Filter
A *filter* is a one-place predicate p which can be used to select those elements of a sequence to be retained, in their original order. For sequence q, its filter by p is $\langle e : q \mid \mathsf{p}\, e\rangle$. More succinctly, we can write just $\mathsf{p} \lhd q$.

Composition
Given some sequence i of natural numbers, the *composition* of a sequence q with i is made by taking the elements of q indexed by the elements of i. It is written $q[i]$ (distinguished from the ordinary indexing by the fact that i is a sequence), and is equal to $\langle n : i \cdot q[n]\rangle$. Note that $q[i]$ is not necessarily a subsequence of q, because i itself might be out of order:

$$\langle 0,2,4,6,8\rangle[\langle 2,1\rangle] \quad = \quad \langle 4,2\rangle \ .$$

The operation is called composition because of its being related to functional composition when the sequences are considered to be functions from their indices to their elements.

Subsequence
A *subsequence* is a sequence composition taken in the original order. For that, we take a set s of natural numbers, and write $q[s]$. The implicit conversion takes s to a sequence — in ascending order — and the resulting composition then selects the elements in that order. A sequence $q2$ is a subsequence of another $q1$ iff there is a set of natural numbers that produces $q2$ from $q1$ in the above way; in that case we write $q2 \ll q1$, which is defined to be $(\exists\, s : \mathbf{set}\, \mathbb{N} \cdot q2 = q1[s])$. For example,

$$\begin{array}{rcl} \langle 0,2,4,6,8\rangle[\{3,1\}] & = & \langle 2,6\rangle \\ \langle 2,6\rangle & \ll & \langle 0,2,4,6,8\rangle \ . \end{array}$$

We also allow the complementary $q\backslash s$, which is the subsequence formed by excluding the indices in s: it is $\langle i : 0{\rightarrow}\#q \mid i \notin s \cdot q[i]\rangle$. For example,

$$\langle 0,2,4,6,8\rangle\backslash\{3,1\} \quad = \quad \langle 0,4,8\rangle \ .$$

Subsegment
A *subsegment* is a contiguous subsequence (without 'gaps'). For that, we compose with a sequence $m{\rightarrow}n$ for some natural numbers m and n; the resulting subsegment of q is $q[m{\rightarrow}n]$. A sequence $q2$ is a subsegment of another $q1$ iff there is a pair of natural numbers that produces $q2$ from $q1$ in the above way; in that case we write $q2 \preceq q1$, which is defined to be $(\exists\, m, n : \mathbb{N} \cdot q2 = q1[m{\rightarrow}n])$. For example,

$$\begin{array}{rcl} \langle 0,2,4,6,8\rangle[1{\rightarrow}3] & = & \langle 2,4\rangle \\ \langle 2,4\rangle & \preceq & \langle 0,2,4,6,8\rangle \ . \end{array}$$

Prefix
A *prefix* is a subsegment that begins the sequence. We compose with a sequence $0 \rightarrow n$ for some natural number n; the resulting prefix is $q[0 \rightarrow n]$, which we can write $q \uparrow n$ and pronounce 'q take n'. A sequence $q2$ is a prefix of another $q1$ iff there is a natural number that produces $q2$ from $q1$ in the above way; in that case we write $q2 \subseteq q1$, which is defined to be $(\exists\, n : \mathbb{N} \cdot q2 = q1 \uparrow n)$. For example,

$$\begin{array}{rcl} \langle 0,2,4,6,8\rangle \uparrow 4 & = & \langle 0,2,4,6\rangle \\ \langle 0,2,4,6\rangle & \subseteq & \langle 0,2,4,6,8\rangle \ . \end{array}$$

Suffix
A *suffix* is a subsegment that ends the sequence. For sequence q, we compose with a sequence $n \rightarrow \#q$ that removes the first n elements; thus the suffix is $q[n \rightarrow \#q]$. We can write that $q \downarrow n$; it is pronounced 'q drop n'. Note that $q \uparrow n +\!\!+\, q \downarrow n = q$ for all n such that $0 \leq n \leq \#q$. For example,

$$\langle 0,2,4,6,8\rangle \downarrow 4 \quad = \quad \langle 8\rangle \ .$$

9.4 Distributed operators

The set, bag, and sequence comprehensions have in common the ideas of bound variable, range, and term: each makes a set, bag, or sequence respectively as indicated by the surrounding brackets: $\{\cdots\}$, $[\![\cdots]\!]$, or $\langle\cdots\rangle$. That convention can be generalised, as we see below. But first we consider some properties of binary operators on their own.

A binary operator $\oplus$ is *associative* if for all a,b,c of appropriate type we have

$$(a \oplus b) \oplus c \quad = \quad a \oplus (b \oplus c) \ .$$

Many of the arithmetic operators are associative (Figure 6.2); in particular we have associative $+$, $\times$, $\sqcup$ and $\sqcap$. The set operators $\cup$, $\cap$, the bag operator $+$, and the sequence operator $+\!\!+$ are associative as well.

To each associative operator corresponds a *distributed* operator that can be applied to a whole sequence. For example, corresponding to $+$, which sums two numbers, we have $\sum$, which sums a sequence of numbers. In general, distributed operators are written

$$(\oplus x : q \mid \mathcal{R} \cdot E) \ ,$$

where $\oplus$ is the associative binary operator, x is the bound variable, $\mathcal{R}$ is the range formula, and E is the term. The elements x of the sequence q are considered, one-by-one, in order; those satisfying $\mathcal{R}$ are retained; the term E is formed for each; and finally $\oplus$ is applied 'between' the resulting values, in order. Since $\oplus$ is associative, it does not matter how the applications of $\oplus$ are grouped.

$(+i : 0 \rightarrow n \cdot i^2)$	The sum of the first n squares.
$(\times d : \mathbb{N} \mid (d \mid n))$	The product of the divisors of $n : \mathbb{N}$.
$(\sqcap z : \mathbb{Z} \mid z \geq r)$	The ceiling $\lceil\,\rceil$ of $r : \mathbb{R}$.
$(\sqcup q' : \mathbf{seq}\,\mathbb{Z} \mid q' \preceq q \wedge q' < 0 \cdot \# q')$	The length of the longest subsegment of $q : \mathbf{seq}\,\mathbb{Z}$ all of whose elements are negative.

Figure 9.9 Examples of distributed operators

The same conventions apply to distributed operators as to comprehensions: the range $\mathcal{R}$ defaults to true, and the term E defaults to the bound variable x. Hence we can define the sum of a sequence $q : \mathbf{seq}\ \mathbb{Z}$, say, as

$$\sum q \;\widehat{=}\; (+x : q \mid \mathsf{true} \cdot x) = (+x : q)\ .$$

Distributed operators are applicable to the empty sequence only if the original operator has an identity, a value e such that for all a we have $e \oplus a = a \oplus e = a$. In that case the result is the identity e, and for example we have therefore that $\sum\langle\rangle = 0$.

An operator $\oplus$ is *commutative* if for all a and b we have

$$a \oplus b \quad = \quad b \oplus a\ .$$

If an operator is commutative and associative, it can be distributed over bags as well as sequences. A non-commutative operator (like $+\!\!+$) cannot be distributed over a bag, because the result depends on the order in which elements are taken, and a bag has no order.

Finally, an operator $\oplus$ is *idempotent* if for all a

$$a \oplus a \quad = \quad a\ .$$

Any operator having all three properties can be distributed over sets as well. Thus the maximum of a set s of numbers is simply

$$(\sqcup x : s)\ .$$

If a non-commutative operator is distributed over a bag, or a non-idempotent operator over a set, we implicitly convert the bag to a sequence, or set to a bag, as appropriate. Thus $(+x : s)$ is the sum of the elements in the set s.

Figure 9.9 gives examples of distributed operators.

9.4.1 Quantifiers with ranges

Conjunction $\wedge$ and disjunction $\vee$ are associative, commutative, and idempotent too, and they are operators over the Boolean type $\{\mathsf{true}, \mathsf{false}\}$. They are also propositional connectives — symbols we use within formulae — and we can exploit their properties there as well: we say that distributing $\wedge$ gives $\forall$, and distributing $\vee$ gives $\exists$.

In fact, the notations of Sections 2.5.1 and 2.5.3 are deliberately close already to that for distributed operators, and the correspondence can be made exact by introducing ranges for quantifiers. (See Predicate laws A.54 and A.55.)

9.5 Functions

9.5.1 Partial functions, domain and range

The square root function $\sqrt{}$, taking real numbers to real numbers, has type

$$\mathbb{R} \nrightarrow \mathbb{R} \; .$$

The left-hand $\mathbb{R}$, the *source*, is the set from which the arguments are drawn; the right-hand $\mathbb{R}$, the *target*, is the set within which the results lie. The direction of the arrow indicates which is which (*from* source *to* target), and the stroke on the arrow indicates that the function is *partial*: there are some elements of its source for which it is undefined.

There is no reason in principle, given the freedom we already allow ourselves with abstract programs, that we could not declare a variable of that same type and assign $\sqrt{}$ to it:

$$\begin{array}{l} \mathbf{var}\ f : \mathbb{R} \nrightarrow \mathbb{R} \bullet \\ f := (\sqrt{}) \; . \end{array}$$

(We have enclosed $\sqrt{}$ in parentheses to make it clear that we mean $\sqrt{}$ *as a function*, not $\sqrt{}$ expecting some further argument.)

In fact we shall do exactly what is suggested above, allowing for any two types S and T the function type $S \nrightarrow T$ which, itself a type, can be used in declarations or to build still further types.

Our mathematical view of functions is that they are sets of pairs, with each pair containing one element from the domain and the corresponding element from the range. Thus these pairs are some of the elements of $\sqrt{}$:

$$\begin{array}{l} (0, 0) \\ (1, 1) \\ (1.21, 1.1) \\ (\pi^2, \pi) \cdots \end{array}$$

All of the pairs are elements of the Cartesian product of $\mathbb{R}$ with $\mathbb{R}$, written $\mathbb{R} \times \mathbb{R}$: in general the elements of the set $S \times T$ are pairs (s,t) with one element drawn from S and the other from T. Thus any function in $S \nrightarrow T$ is a subset of $S \times T$.

Associated with functions, as a type, are certain operations. The *domain* of a function is that subset of its source on which it is defined: for $f : S \nrightarrow T$,

$$\mathsf{dom}\, f \ \widehat{=}\ \{s : S;\ t : T \mid (s,t) \in f \bullet s\}\ .$$

Since f is itself a set, we can write that more succinctly as

$$\{(s,t) : f \bullet s\}$$

if we allow tuples as bound variables (which therefore we do).

The *range* of a function is that subset of the target which it might actually produce (given the right arguments):

$$\mathsf{ran}\, f \ \widehat{=}\ \{(s,t) : f \bullet t\}\ .$$

Thus $\mathsf{dom}(\surd) = \mathsf{ran}(\surd) =$ 'the non-negative reals'.

Note that $\mathsf{ran}(\surd) =$ 'the non-negative reals' means in particular that every non-negative real number is the square root of something.

9.5.2 Total functions

For any function $f : S \nrightarrow T$, we have $\mathsf{dom}\, f \subseteq S$ and $\mathsf{ran}\, f \subseteq T$. When equality holds in either case, we can be more specific: function f above is total when it is defined on all its source. In other words,

f is *total* means that $\mathsf{dom}\, f = S$.

If f can produce every element of its range, we say that it is onto (or surjective):

f is *onto*, or *surjective*, means that $\mathsf{ran}\, f = T$.

For total functions we have the special notation of 'uncrossed' arrow, so that declaring $f : S \rightarrow T$ is the same as declaring $f : S \nrightarrow T$ and stating additionally that f is total. Put another way,

$$\mathbf{var}\ f : S \rightarrow T$$

has the same effect as $\mathbf{var}\ f : S \nrightarrow T;\ \mathbf{and}\ \mathsf{dom}\, f = S$.

Totality of a function is relative to its declared source: although partial over $\mathbb{R}$, the square root function is total over the non-negative reals. The same applies to whether the function is onto: thus square root is total and onto if declared from non-negative reals to non-negative reals.

9.5.3 Function application and overriding

Given $f : S \nrightarrow T$ and some $s : \mathsf{dom} f$ (which implies that $s \in S$ as well), we apply the function f to its argument s by writing $f\ s$. The result is an element of $\mathsf{ran} f$, and of T (since $\mathsf{ran} f \subseteq T$). An alternative way of writing the application is $f[s]$.

('Ordinarily', such function application is written $f(s)$. We have chosen instead to reserve parentheses for grouping, and indicate application by simple juxtaposition. The $f[s]$ variant is suggested by analogy with sequences, since they are functions from their indices to their values.)

As an 'abuse' of notation (actually a convenience), we allow $f[ss]$, given $ss : \mathbf{set}\ S$ as a *set* of values, and by it we mean the set of results obtained by applying f to elements of ss separately (and ignoring those that are undefined). Thus

$$f[ss] \ \widehat{=}\ \{(s,t) : f \mid s \in ss \cdot t\} \ .$$

We can modify a function at one or more of its source values, so that

$$f[s := t]$$

is the function f *overridden* by $s := t$. Letting g be $f[s := t]$, we have

$$\begin{array}{rrcll} & g[s] & = & t & \text{(no matter what } f[s] \text{ is),} \\ \text{and} & g[s'] & = & f[s'] & \text{for any } s' \neq s. \end{array}$$

If $s \neq s'$ and f is not defined at s', then neither is g.

Similarly,

$$\begin{array}{rcll} (f[ss := t])[s] & = & t & \text{if } s \in ss \\ & = & f[s] & \text{if } s \notin ss \ . \end{array}$$

More generally still, we can override f by another function g; the resulting function $f \oplus g$ behaves like g if it can (if g is defined at that argument), otherwise like f. Thus

$$\begin{array}{rcll} (f \oplus g)[s] & = & g[s] & \text{if } s \in \mathsf{dom}\, g \\ & = & f[s] & \text{otherwise.} \end{array}$$

If neither f nor g is defined at s, then $f \oplus g$ is undefined there also.

In terms of sets,

$$f \oplus g \quad = \quad \{(s,t) : f \cup g \mid s \in \mathsf{dom}\, g \Rightarrow (s,t) \in g\} \ .$$

That last formulation takes undefinedness automatically into account.

Our earlier notations for overriding can now be seen as special cases of the above, because $f[s := t]$ is just f overridden by the (singleton) function $\{(s,t)\}$, which takes s to t but is undefined everywhere else. In the $f[ss := t]$ case the overriding function is $\{s : ss \cdot (s,t)\}$, defined only on ss.

The overriding notations '$[s := \cdots]$' apply to sequences also, as they are a special case of functions.

9.5.4 Restriction and corestriction

Finally we have operators for restricting functions to smaller domains and ranges. Given $f : S \nrightarrow T$, $ss : \mathsf{set}\, S$ and $tt : \mathsf{set}\, T$, we define

$$\begin{array}{lcl} ss \lhd f & \hat{=} & \{(s,t) : f \mid s \in ss\} \\ ss \ntriangleleft f & \hat{=} & \{(s,t) : f \mid s \notin ss\} \\ f \rhd tt & \hat{=} & \{(s,t) : f \mid t \in tt\} \\ f \ntriangleright tt & \hat{=} & \{(s,t) : f \mid t \notin tt\} \ . \end{array}$$

An immediate use for $\ntriangleleft$ is an even more compact definition of overriding:

$$f \oplus g \quad = \quad ((\mathsf{dom}\ g) \ntriangleleft f) \cup g \ .$$

9.6 Relations

9.6.1 Generalised functions

Relations are a generalisation of functions: for source S and target T the corresponding relational type is written

$$S \leftrightarrow T \ ,$$

and, like functions, relations are sets of pairs. In fact,

$$S \leftrightarrow T \quad = \quad \mathbf{set}(S \times T) \ ,$$

which means that *any* subset of $S \times T$ is a relation. In contrast, only some subsets of $S \times T$ are functions; just which subsets they are we shall see shortly.

The generalisation of relations beyond functions is that relations are 'multi-valued': whereas for function f and source value s there is at most one $f[s]$, for relation r there may be many related target values.

Compare for example the function pred of type $\mathbb{N} \nrightarrow \mathbb{N}$ (it subtracts 1 from positive natural numbers) with the relation 'less than' $<$, of type $\mathbb{N} \leftrightarrow \mathbb{N}$. The two agree on source element 0 (because pred is undefined there, and no natural number is less than 0), and on source element 1 (because $\mathsf{pred}\, 1 = 0$ and the only natural number less than 1 is 0). But beyond 1 we find that $<$ is more generous:

source value s	pred s	less than s
2	1	0,1
3	2	0,1,2
4	3	0,1,2,3

The function returns just one value, whereas the relation relates s to many values: the predecessor of s is *one of* the values less than it.

As sets, we have

$$\begin{array}{rcl} \mathsf{pred} &=& \{(1,0),(2,1),(3,2),\cdots\} \\ (<) &=& \{(1,0),(2,1),(2,0),(3,2),(3,1),(3,0),\cdots\}\ , \end{array}$$

and thus we see clearly the difference between a relation and a function: in this case it is just that $\mathsf{pred} \subseteq (<)$.

9.6.2 Functions are relations

Functions and relations are both sets of pairs — but functions have the special property of being single-valued: for relation r in $S \leftrightarrow T$, we say that r is *functional*, or *single-valued*, if for all s in S there is at most one r-related t in T. That is, $r : S \leftrightarrow T$ is *functional* iff

$$(\forall\, s : S;\, t, t' : T \bullet (s,t) \in r \wedge (s,t') \in r \Rightarrow t = t')\ .$$

Thus a function is just a functional relation.

A related property is injectivity (or being one-to-one): a relation $r : S \leftrightarrow T$ is *injective* if

$$(\forall\, s, s' : S;\, t : T \bullet (s,t) \in r \wedge (s',t) \in r \Rightarrow s = s')\ .$$

The same notion applies to functions — because they are relations — and so an injection (or one-to-one) function is one that cannot deliver the same result for different arguments. Thus pred is injective, because $a - 1 = b - 1 \Rrightarrow a = b$; but sqr is not, because for example

$$\mathsf{sqr}(-1) \quad = \quad 1 \quad = \quad \mathsf{sqr}\,1\ ,$$

but $-1 \neq 1$. (Function sqr returns the square of its argument. Don't worry about the font convention for functions — it is only a convention, after all — but we are using sans serif for specific, named functions like sqr, and italic for function variables like f.)

Most of the operators and notations we have defined for functions work for relations as well, and we summarise them here: for $r : S \leftrightarrow T$,

$$\begin{array}{rcl} \mathsf{dom}\ r &=& \{(s,t) : r \bullet s\} \\ \mathsf{ran}\ r &=& \{(s,t) : r \bullet t\} \\ r \text{ is total} &\text{iff}& \mathsf{dom}\ r = S \\ r \text{ is onto} &\text{iff}& \mathsf{ran}\ r = T \\ \text{if } ss : \textbf{set}\ S, \text{ then } r[ss] &=& \{(s,t) : r \mid s \in ss \bullet t\}\ . \end{array}$$

For overriding we have

$$r \oplus r' \quad = \quad ((\mathsf{dom}\ r') \lhd r) \cup r'\ ,$$

and for the more specific cases then

$$\begin{array}{rcl} r[s := t] & = & r \oplus \{(s,t)\} \\ r[ss := t] & = & r \oplus \{s : ss \cdot (s,t)\} \ . \end{array}$$

Thus for example $r[s := t]$ replaces *all* associations from s with a *single* new association to t.

Finally, for *applying* a relation one might be inclined to define

$$r[s] \quad = \quad \{t : T \mid (s,t) \in r\} \ ,$$

but there is a potential confusion there in that we would not know for example whether $\mathsf{pred}\,1$ was 0 (taking pred as a function) or $\{0\}$ (taking pred as a relation).

For writing that s and t are related by r, we have two possibilities: either $(s,t) \in r$, relying on the set-based nature of r, or $s\langle r\rangle t$, a special notation for relations.

The fact that functions are relations causes no problems here, since $1\langle\mathsf{pred}\rangle 0$ means the same as $0 = \mathsf{pred}\,1$. (Note however the confusion caused by the general convention of writing function types from *left* to right, but supplying their arguments on the *right*!)

9.6.3 Inverses

Given $r : S \leftrightarrow T$, its inverse is written r^{-1} and is of type $T \leftrightarrow S$. The value of r^{-1} is obtained from r simply by reversing the pairs:

$$r^{-1} \quad = \quad \{(s,t) : r \cdot (t,s)\} \ .$$

Thus $(<)^{-1} = (>)$, and $\mathsf{pred}^{-1} = \mathsf{succ}$ (where function succ adds 1 to its argument).

That pred^{-1} is a function (rather than a relation) is just a bit of good luck: it is because pred is injective. In contrast, inverting sqr does not give a function, because for example

$$\{(1,-1),(1,1)\} \subset \mathsf{sqr}^{-1} \ .$$

Since sqr is not injective, its inverse is not functional: it is a proper relation that for any argument supplies both the positive and the negative square root. Our earlier function $\sqrt{}$ is a proper subset of sqr^{-1}.

9.7 Exercises

Ex. 9.1 Evaluate these terms:

1. $\mathsf{bag}\{\}$
2. $\mathsf{set}[\![\,]\!]$

3. $\mathsf{bag}_0\ s$
4. $\mathsf{bag}\{1,1\}$
5. $\mathsf{set}\llbracket 1,1 \rrbracket$
6. $\mathsf{set}\llbracket m,n:\mathbb{N}\cdot m+n \rrbracket$.

Ex. 9.2 ♡ Write set comprehensions for the following:

1. The perfect squares.
2. The natural numbers whose prime factors are in the set $\{2,3,5\}$.
3. The prime numbers.
4. The complex n^{th} roots of unity.

Ex. 9.3 ♡ Consider this alternative definition of the promoted relation $\odot$:

$$s1 \odot s2 \ \widehat{=}\ s1 \neq \{\} \wedge s2 \neq \{\} \wedge (\forall\, t1 : s1;\ t2 : s2 \cdot t1 \odot t2)\ .$$

Now suppose that $\odot$ is transitive; is its promotion transitive as well? Why don't we define promotion as above?

Ex. 9.4 Evaluate these terms:

1. $\mathsf{seq}\,\mathbb{N}$.
2. $\mathsf{seq}\,\mathsf{bag}\langle 2,1,3,1\rangle$.
3. $\mathsf{seq}\,\mathsf{set}\langle 2,1,3,1\rangle$.
4. $\{n:\mathbb{N}\cdot 2n\}\,[7]$ (The 7^{th} element of the sequence formed from that set.)
5. $\langle m,n:\mathbb{N}\cdot m+n\rangle.99$ (The number of occurrences of 99 in the bag formed from that sequence.)

Ex. 9.5 Show that the operations $\cup$, $\cap$, and $-$, applied to bags without repetitions, yield bags without repetitions. (That is why we can use the same symbols for operations on sets.)

Ex. 9.6 ♡ Consider a set comprehension in which *no* bound variables are given. What is the value of $\{\mid \mathsf{true}\cdot x\}$? Of $\{\mid \mathsf{false}\cdot x\}$? What is the value of $\langle i : 0 {\rightarrow} n \cdot x\rangle$?

Ex. 9.7 ♡ Define the *product* $\prod q$ of a sequence $q:\mathbf{seq}\,\mathbb{Z}$. What is $\prod\langle\rangle$?

Ex. 9.8 If we restrict the distributed maximum $\bigsqcup$ to sets $s:\mathbb{N}$ of *natural* numbers, what would $\bigsqcup\{\}$ be? Why?

Ex. 9.9 Write as a set comprehension the set of all permutations of a given sequence q.

Ex. 9.10 ♡ What is the effect of this operator on sets s:

$$(+x : s \cdot 1)\ ?$$

On bags? On sequences?

Ex. 9.11 In type expressions, we let Cartesian product bind more tightly than $\nrightarrow$ or $\rightarrow$, and the latter two both associate to the right.

Give the sizes of these types in terms of the sizes of their components:

1. $S1 \times S2 \rightarrow T$ — that is, $(S1 \times S2) \rightarrow T$
2. $S1 \rightarrow S2 \rightarrow T$ — that is, $S1 \rightarrow (S2 \rightarrow T)$
3. $S1 \times S2 \nrightarrow T$
4. $S1 \nrightarrow S2 \nrightarrow T$
5. $S1 \rightarrow S2 \nrightarrow T$
6. $S1 \nrightarrow S2 \rightarrow T$.

Explain carefully any discrepancy in size between types 3 and 4.

Ex. 9.12 A sequence $s : \mathbf{seq}_N\ T$ can be regarded as a function of type $\mathbb{N} \nrightarrow T$. What subset precisely of $\mathbb{N} \nrightarrow T$ is the set $\mathbf{seq}\ T$?

With s declared as above, what is $\mathsf{dom}\ s$?

Ex. 9.13 Which of these declarations are of total functions over $\mathbb{N}$?

1. $s : \mathbf{seq}_N\ T$
2. $s : \mathbf{seq}\ T$
3. $s : \mathbf{seq}_\infty\ T$

Ex. 9.14 Consider $f : \mathbb{N} \nrightarrow T$ and $s : \mathbf{seq}\ T$. For $n : \mathbb{N}$, when is $f[n] := t$ meaningful but $s[n] := t$ not?

Ex. 9.15 *Linear search* Assuming declarations $as : \mathbf{seq}_N\ A$ and $i : \mathbb{N};\ a : A$, show that

$$\begin{array}{ll} & i\colon [a \in as \Rightarrow a = as[i]] \\ \sqsubseteq & i := 0; \\ & \mathbf{do}\ i < N \wedge a \neq as[i] \rightarrow i := i + 1\ \mathbf{od}\ . \end{array}$$

Hint: Note that termination is required even if $a \notin as$: use invariant $a \notin as{\uparrow}i$. Do not worry about possible 'undefinedness' of $as[i]$ when $i = N$, since the first conjunct $i < N$ is false in that case anyway. (See Exercises 9.16 and 9.17 however in that connection.)

Ex. 9.16 ♡ *Linear search* Assume our programming language treats $as[i]$ as 'undefined' when i is not in the domain of as, and that programs evaluating $as[i]$ under those circumstances will behave unpredictably (like **abort**).[1] Thus (recalling Section 6.7) an evaluation of $as[i]$ will not be accepted *on its own* as code by our compiler: if as were declared $\mathbf{seq}_N\ A$, for example, then

[1]That is the usual situation, but differs from the convention in this text that all expressions terminate: we would say for i outside the domain of as that $as[i]$ returned some value but we do not know which. Our view is convenient for development, but requires more from compilers: either they must generate always-terminating code for expressions (as in Exercise 9.15), or they must carry out compile-time checks as illustrated in this exercise.

$$a := as[i]$$

would not be code, and would be treated as a syntax error (although it is nevertheless meaningful). We would have to write instead

$$\{i < N\}\ a := as[i]\ ,$$

assuming the declaration $i : \mathbb{N}$ (which guarantees $0 \le i$).

For possibly 'undefined' iteration guards the compiler would insist on 'wellformedness assumptions' (like $\{i < N\}$ above) placed as follows:

$$\begin{array}{l} \{\text{`}G \text{ is defined'}\} \\ \textbf{do } G \rightarrow \\ \quad \mathit{prog} \\ \quad \{\text{`}G \text{ is defined'}\} \\ \textbf{od}\ . \end{array}$$

Explain briefly why they should be placed like that.

Show that

$$\begin{array}{ll} & i\colon [a \in as\ ,\ a = as[i]] \\ \sqsubseteq & i := 0; \\ & \{i < N\} \\ & \textbf{do } a \ne as[i] \rightarrow \\ & \qquad i := i + 1 \\ & \qquad \{i < N\} \\ & \textbf{od}\ , \end{array}$$

and explain informally why the possible evaluation of the 'undefined' $as[i]$ is now acceptable. What general rule can you formulate about the connection between iteration invariants and definedness conditions for the iteration guards?

Ex. 9.17 ♡ Can Exercise 9.15 be done under the conditions of Exercise 9.16? Putting it rigorously, we are asking whether this refinement is valid:

$$\begin{array}{ll} & i\colon [a \in as \Rightarrow a = as[i]] \\ \sqsubseteq & i := 0; \\ & \{i < N\} \\ & \textbf{do } i < N \wedge a \ne as[i] \rightarrow \\ & \qquad i := i + 1 \\ & \qquad \{i < N\} \\ & \textbf{od}\ . \end{array}$$

If it is valid, show it to be so; if it is not, *show* it to be invalid, and give an operational explanation for the failure.