Sets, Functions, and Domains

Functions are fundamental to denotational semantics. This chapter introduces functions through set theory, which provides a precise yet intuitive formulation. In addition, the concepts of set theory form a foundation for the theory of *semantic domains*, the value spaces used for giving meaning to languages. We examine the basic principles of sets, functions, and domains in turn.

2.1 SETS

A *set* is a collection; it can contain numbers, persons, other sets, or (almost) anything one wishes. Most of the examples in this book use numbers and sets of numbers as the members of sets. Like any concept, a set needs a representation so that it can be written down. Braces are used to enclose the members of a set. Thus, $\{1, 4, 7\}$ represents the set containing the numbers 1, 4, and 7. These are also sets:

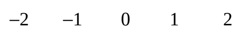
$\{1, \{1, 4, 7\}, 4\}$
$\{ \mathit{red}, \mathit{yellow}, \mathit{grey} \}$
$\{\}$

The last example is the *empty set*, the set with no members, also written as $\emptyset$.

When a set has a large number of members, it is more convenient to specify the conditions for membership than to write all the members. A set S can be defined by $S = \{x \mid P(x)\}$, which says that an object a belongs to S iff (if and only if) a has property P, that is, $P(a)$ holds true. For example, let P be the property ''is an even integer.'' Then $\{x \mid x \textit{ is an even integer}\}$ defines the set of even integers, an infinite set. Note that $\emptyset$ can be defined as the set $\{x \mid x \neq x\}$. Two sets R and S are equivalent, written $R = S$, if they have the same members. For example, $\{1, 4, 7\} = \{4, 7, 1\}$.

These sets are often used in mathematics and computing:

1. *Natural numbers*: $\mathbb{N} = \{\, 0, 1, 2, \cdots \,\}$
2. *Integers*: $\mathbb{Z} = \{\, \cdots, -2, -1, 0, 1, 2, \cdots \,\}$
3. *Rational numbers*: $\mathbb{Q} = \{\, x \mid \textit{for } p \in \mathbb{Z} \text{ and } q \in \mathbb{Z},\ q \neq 0,\ x = p/q \,\}$
4. *Real numbers*: $\mathbb{R} = \{\, x \mid$ *x is a point on the line*



$\}$

5. *Characters*: $\mathbb{C} = \{\, x \mid x$ is a character $\}$
6. *Truth values (Booleans)*: $\mathbb{B} = \{\, \textit{true}, \textit{false} \,\}$

The concept of membership is central to set theory. We write $x \in S$ to assert that x is a member of set S. The membership test provides an alternate way of looking at sets. In the above examples, the internal structure of sets was revealed by ''looking inside the braces'' to see all the members inside. An external view treats a set S as a closed, mysterious object to which we can only ask questions about membership. For example, ''does $1 \in S$ hold?,'' ''does $4 \in S$ hold?,'' and so on. The internal structure of a set isn't even important, as long as membership questions can be answered. To tie these two views together, set theory supports the *extensionality principle*: a set R is equivalent to a set S iff they answer the same on all tests concerning membership:

$R = S$ if and only if, for all x, $x \in R$ holds iff $x \in S$ holds

Here are some examples using membership:

$1 \in \{\, 1, 4, 7 \,\}$ holds
$\{1\} \in \{\, 1, 4, 7 \,\}$ does not hold
$\{1\} \in \{\, \{1\}, 4, 7 \,\}$ holds

The extensionality principle implies the following equivalences:

$\{\, 1, 4, 7 \,\} = \{\, 4, 1, 7 \,\}$
$\{\, 1, 4, 7 \,\} = \{\, 4, 1, 7, 4 \,\}$

A set R is a *subset* of a set S if every member of R belongs to S:

$R \subseteq S$ if and only if, for all x, $x \in R$ implies $x \in S$

For example,

$\{1\} \subseteq \{\, 1, 4, 7 \,\}$
$\{\, 1, 4, 7 \,\} \subseteq \{\, 1, 4, 7 \,\}$
$\{\,\} \subseteq \{\, 1, 4, 7 \,\}$

all hold true but $\{1\} \not\subseteq \{\, \{1\}, 4, 7 \,\}$.

2.1.1 Constructions on Sets

The simplest way to build a new set from two existing ones is to *union* them together; we write $R \cup S$ to denote the set that contains the members of R and S and no more. We can define set union in terms of membership:

for all x, $x \in R \cup S$ if and only if $x \in R$ or $x \in S$

Here are some examples:

$\{ 1, 2 \} \cup \{ 1, 4, 7 \} = \{ 1, 2, 4, 7 \}$
$\{ \} \cup \{ 1, 2 \} = \{ 1, 2 \}$
$\{ \{ \} \} \cup \{ 1, 2 \} = \{ \{ \}, 1, 2 \}$

The union operation is commutative and associative; that is, $R \cup S = S \cup R$ and $(R \cup S) \cup T = R \cup (S \cup T)$. The concept of union can be extended to join an arbitrary number of sets. If $R_0, R_1, R_2, \cdots$ is an infinite sequence of sets, $\bigcup_{i=0}^{\infty} R_i$ stands for their union. For example, $\mathbb{Z} = \bigcup_{i=0}^{\infty} \{ -i, \cdots, -1, 0, 1, \cdots, i \}$ shows how the infinite union construction can build an infinite set from a group of finite ones.

Similarly, the *intersection* of sets R and S, $R \cap S$, is the set that contains only members common to both R and S:

for all x, $x \in R \cap S$ if and only if $x \in R$ and $x \in S$

Intersection is also commutative and associative.

An important concept that can be defined in terms of sets (though it is not done here) is the *ordered pair*. For two objects x and y, their pairing is written (x,y). Ordered pairs are useful because of the indexing operations *fst* and *snd*, defined such that:

$fst(x,y) = x$
$snd(x,y) = y$

Two ordered pairs P and Q are equivalent iff $fst\,P = fst\,Q$ and $snd\,P = snd\,Q$. Pairing is useful for defining another set construction, the product construction. For sets R and S, their *product* $R \times S$ is the set of all pairs built from R and S:

$R \times S = \{ (x,y) \mid x \in R \text{ and } y \in S \}$

Both pairing and products can be generalized from their binary formats to n-tuples and n-products.

A form of union construction on sets that keeps the members of the respective sets R and S separate is called *disjoint union* (or sometimes, *sum*):

$R + S = \{ (zero, x) \mid x \in R \} \cup \{ (one, y) \mid y \in S \}$

Ordered pairs are used to ‘‘tag’’ the members of R and S so that it is possible to examine a member and determine its origin.

We find it useful to define operations for assembling and disassembling members of $R+S$. For assembly, we propose in$R$ and in$S$, which behave as follows:

for $x\in R$, in$R(x) = (zero, x)$
for $y\in S$, in$S(y) = (one, y)$

To remove the tag from an element $m\in R+S$, we could simply say $snd(m)$, but will instead resort to a better structured operation called *cases*. For any $m\in R+S$, the value of:

```
cases $m$ of
    is$R(x) \rightarrow \cdots x \cdots$
    [] is$S(y) \rightarrow \cdots y \cdots$
end
```

is ‘‘ $\cdots x \cdots$ ’’ when $m = (zero, x)$ and is ‘‘ $\cdots y \cdots$ ’’ when $m = (one, y)$. The *cases* operation makes good use of the tag on the sum element; it checks the tag before removing it and using the value. Do not be confused by the is$R$ and is$S$ phrases. They are not new operations. You should read the phrase is$R(x) \rightarrow \cdots x \cdots$ as saying, ‘‘if m is an element whose tag component is R and whose value component is x, then the answer is $\cdots x \cdots$.’’ As an example, for:

```
$f(m) =$ cases $m$ of
        is$\mathbb{N}(n) \rightarrow n+1$
        [] is$\mathbb{B}(b) \rightarrow 0$
        end
```

$f($in$\mathbb{N}(2)) = f(zero, 2) = 2+1 = 3$, but $f($in$\mathbb{B}(true)) = f(one, true) = 0$.

Like a product, the sum construction can be generalized from its binary format to n-sums.

Finally, the set of all subsets of a set R is called its *powerset*:

$$\mathbb{P}(R) = \{\, x \mid x \subseteq R \,\}$$

$\{\,\} \in \mathbb{P}(R)$ and $R \in \mathbb{P}(R)$ both hold.

2.2 FUNCTIONS

Functions are rather slippery objects to catch and examine. A function cannot be taken apart and its internals examined. It is like a ‘‘black box’’ that accepts an object as its input and then transforms it in some way to produce another object as its output. We must use the ‘‘external approach’’ mentioned above to understand functions. Sets are ideal for formalizing the method. For two sets R and S, f is a *function* from R to S, written $f: R \rightarrow S$, if, to each member of R, f associates exactly one member of S. The expression $R \rightarrow S$ is called the *arity* or *functionality* of f. R is the *domain* of f; S is the *codomain* of f. If $x\in R$ holds, and the element paired to x by f is y, we write $f(x) = y$. As a simple example, if $R = \{\, 1, 4, 7 \,\}$, $S = \{\, 2, 4, 6 \,\}$, and f maps R to S as follows:

$$\begin{tikzcd}
R & & S \\
& f & \\
1 & & 2 \\
4 & & 4 \\
7 & & 6
\end{tikzcd}$$

then f is a function. Presenting an argument a to f is called *application* and is written $f(a)$. We don't know *how* f transforms 1 to 2, or 4 to 6, or 7 to 2, but we accept that somehow it does; the results are what matter. The viewpoint is similar to that taken by a naive user of a computer program: unaware of the workings of a computer and its software, the user treats the program as a function, as he is only concerned with its input-output properties. An extensionality principle also applies to functions. For functions $f: R \rightarrow S$ and $g: R \rightarrow S$, f is equal to g, written $f = g$, iff for all $x \in R$, $f(x) = g(x)$.

Functions can be combined using the composition operation. For $f: R \rightarrow S$ and $g: S \rightarrow T$, $g \circ f$ is the function with domain R and codomain T such that for all $x: R$, $g \circ f(x) = g(f(x))$. Composition of functions is associative: for f and g as given above and $h: T \rightarrow U$, $h \circ (g \circ f) = (h \circ g) \circ f$.

Functions can be classified by their mappings. Some classifications are:

1. *one-one*: $f: R \rightarrow S$ is a *one-one* (1-1) function iff for all $x \in R$ and $y \in R$, $f(x) = f(y)$ implies $x = y$.
2. *onto*: $f: R \rightarrow S$ is an *onto* function iff $S = \{\, y \mid \textit{there exists some } x \in R \textit{ such that } f(x) = y \,\}$.
3. *identity*: $f: R \rightarrow R$ is the *identity* function for R iff for all $x \in R$, $f(x) = x$.
4. *inverse*: for some $f: R \rightarrow S$, if f is one-one and onto, then the function $g: S \rightarrow R$, defined as $g(y) = x$ iff $f(x) = y$ is called the *inverse function of f*. Function g is denoted by f^{-1}.

Functions are used to define many interesting relationships between sets. The most important relationship is called an *isomorphism*: two sets R and S are *isomorphic* if there exist a pair of functions $f: R \rightarrow S$ and $g: S \rightarrow R$ such that $g \circ f$ is the identity function for R and $f \circ g$ is the identity function for S. The maps f and g are called *isomorphisms*. A function is an isomorphism if and only if it is one-one and onto. Further, the inverse f^{-1} of isomorphism f is also an isomorphism, as $f^{-1} \circ f$ and $f \circ f^{-1}$ are both identities. Here are some examples:

1. $R = \{\, 1, 4, 7 \,\}$ is isomorphic to $S = \{\, 2, 4, 6 \,\}$; take $f: R \rightarrow S$ to be $f(1)=2$, $f(4)=6$, $f(7)=4$; and $g: S \rightarrow R$ to be $g(2)=1$, $g(4)=7$, $g(6)=4$.
2. For sets A and B, $A \times B$ is isomorphic to $B \times A$; take $f: A \times B \rightarrow B \times A$ to be $f(a,b) = (b,a)$.
3. $\mathbb{N}$ is isomorphic to $\mathbb{Z}$; take $f: \mathbb{N} \rightarrow \mathbb{Z}$ to be:

$$f(x) = \begin{cases} x/2 & \textit{if } x \textit{ is even} \\ -((x+1)/2) & \textit{if } x \textit{ is odd} \end{cases}$$

You are invited to calculate the inverse functions in examples 2 and 3.

2.2.1 Representing Functions as Sets

We can describe a function via a set. We collect the input-output pairings of the function into a set called its *graph.* For function $f: R \to S$, the set:

$$graph(f) = \{ (x, f(x)) \mid x \in R \}$$

is the graph of f. Here are some examples:

1. $f: R \to S$ in example 1 above:
   $graph(f) = \{ (1,2), (4,6), (7,4) \}$
2. the successor function on $\mathbb{Z}$:
   $graph(succ) = \{ \cdots, (-2,-1), (-1,0), (0,1), (1,2), \cdots \}$
3. $f: \mathbb{N} \to \mathbb{Z}$ in example 3 above:
   $graph(f) = \{ (0,0), (1,-1), (2,1), (3,-2), (4,2), \cdots \}$

In every case, we list the domain and codomain of the function to avoid confusion about which function a graph represents. For example, $f: \mathbb{N} \to \mathbb{N}$ such that $f(x)=x$ has the same graph as $g: \mathbb{N} \to \mathbb{Z}$ such that $g(x)=x$, but they are different functions.

We can understand function application and composition in terms of graphs. For application, $f(a)=b$ iff (a,b) is in $graph(f)$. Let there be a function *apply* such that $f(a) = apply(graph(f), a)$. Composition is modelled just as easily; for graphs $f: R \to S$ and $g: S \to T$:

$$graph(g \circ f) = \{ (x,z) \mid x \in R \textit{ and there exists a } y \in S \textit{ such that } (x,y) \in graph(f) \textit{ and } (y,z) \in graph(g) \}$$

Functions can have arbitrarily complex domains and codomains. For example, if R and S are sets, so is $R \times S$, and it is reasonable to make $R \times S$ the domain or codomain of a function. If it is the domain, we say that the function ‘‘needs two arguments’’; if it is the codomain, we say that it ‘‘returns a pair of values.’’ Here are some examples of functions with compound domains or codomains:

1. $add: (\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$
   $graph(add) = \{ ((0,0), 0), ((1,0), 1), ((0,1), 1), ((1,1), 2), ((2,1), 3), \cdots \}$
2. $duplicate: R \to (R \times R)$, where $R = \{ 1, 4, 7 \}$
   $graph(duplicate) = \{ (1, (1,1)), (4,(4,4)), (7, (7,7)) \}$
3. $which\text{-}part: (\mathbb{B}+\mathbb{N}) \to S$, where $S = \{ isbool, isnum \}$
   $graph(which\text{-}part) = \{ ((zero, true), isbool), ((zero, false), isbool), ((one, 0), isnum), ((one,1), isnum), ((one, 2), isnum), \cdots, ((one, n), isnum), \cdots \}$
4. $make\text{-}singleton: \mathbb{N} \to \mathbb{P}(\mathbb{N})$
   $graph(make\text{-}singleton) = \{ (0, \{0\}), (1, \{1\}), \cdots, (n, \{n\}), \cdots \}$
5. $nothing: \mathbb{B} \cap \mathbb{N} \to \mathbb{B}$
   $graph(nothing) = \{ \}$

The graphs make it clear how the functions behave when they are applied to arguments. For example, $apply(graph(which\text{-}part), (one, 2)) = isnum$. We see in example 4 that a function can return a set as a value (or, for that matter, use one as an argument). Since a function can be represented by its graph, which is a set, we will allow functions to accept other functions as arguments and produce functions as answers. Let the *set of functions from R to S* be a set whose members are the graphs of all functions whose domain is R and codomain is S. Call this set $R \rightarrow S$. Thus the expression $f: R \rightarrow S$ also states that f's graph is a member of the set $R \rightarrow S$. A function that uses functions as arguments or results is called a *higher-order function.* The graphs of higher-order functions become complex very quickly, but it is important to remember that they do exist and everything is legal under the set theory laws. Here are some examples:

6. *split-add* : $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$. Function *split-add* is the addition function ''split up'' so that it can accept its two arguments one at a time. It is defined as *split-add*$(x) = g$, where $g: \mathbb{N} \rightarrow \mathbb{N}$ is $g(y) = add\,(x,y)$. The graph gives a lot of insight:

$$\begin{aligned} graph\,(split\text{-}add) = \{\, &(\,0, \{\,(0,0), (1,1), (2,2), \cdots\,\}\,),\\ &(\,1, \{\,(0,1), (1,2), (2,3), \cdots\,\}\,),\\ &(\,2, \{\,(0,2), (1,3), (2,4), \cdots\,\}\,), \cdots\,\} \end{aligned}$$

Each argument from $\mathbb{N}$ is paired with a graph that denotes a function from $\mathbb{N}$ to $\mathbb{N}$. Compare the graph of *split-add* to that of *add;* there is a close relationship between functions of the form $(R \times S) \rightarrow T$ to those of the form $R \rightarrow (S \rightarrow T)$. The functions of the first form can be placed in one-one onto correspondence with the ones of the second form— the sets $(R \times S) \rightarrow T$ and $R \rightarrow (S \rightarrow T)$ are isomorphic.

7. *first-value* : $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$. The function looks at the value its argument produces when applied to a zero; *first-value*$(f) = f(0)$, and:

$$\begin{aligned} graph\,(first\text{-}value) = \{\, &\cdots, (\{\,(0,1), (1,1), (2,1), (3,6), \cdots\,\}, 1\,),\\ &\cdots, (\{\,(0,49), (1,64), (2,81), (3,100), \cdots\,\}, 49\,),\\ &\cdots\,\} \end{aligned}$$

Writing the graph for the function is a tedious (and endless) task, so we show only two example argument, answer pairs.

8. *make-succ* : $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$. Function *make-succ* builds a new function from its argument by adding one to all the argument function's answers: *make-succ*$(f) = g$, where $g: \mathbb{N} \rightarrow \mathbb{N}$ and $g\,(x) = f(x)+1$.

$$\begin{aligned} graph\,(make\text{-}succ) = \{\, &\cdots,\\ &(\{\,(0,1), (1,1), (2,1), (3,6), \cdots\,\},\\ &\quad \{\,(0,2), (1,2), (2,2), (3,7), \cdots\,\}\,),\\ &\cdots,\\ &(\{\,(0,49), (1,64), (2,81), (3,100), \cdots\,\},\\ &\quad \{\,(0,50), (1,65), (2,82), (3,101), \cdots\,\}\,)\\ &\cdots\,\} \end{aligned}$$

9. *apply* : $((\mathbb{N} \rightarrow \mathbb{N}) \times \mathbb{N}) \rightarrow \mathbb{N}$. Recall that *apply*$\,(f,x) = f(x)$, so its graph is:

$$
\begin{aligned}
graph(apply) = \{ \cdots, & ((\{ (0,1), (1,1), (2,1), (3,6), \cdots \}, 0), 1),\\
& ((\{ (0,1), (1,1), (2,1), (3,6), \cdots \}, 1), 1),\\
& ((\{ (0,1), (1,1), (2,1), (3,6), \cdots \}, 2), 1),\\
& ((\{ (0,1), (1,1), (2,1), (3,6), \cdots \}, 3), 6),\\
& \cdots,\\
& ((\{ (0,49), (1,64), (2,81), (3,100), \cdots \}, 0), 49),\\
& ((\{ (0,49), (1,64), (2,81), (3,100), \cdots \}, 1), 64),\\
& \cdots \}
\end{aligned}
$$

The graph of *apply* is little help; things are getting too complex. But it is important to understand why the pairs are built as they are. Each pair in *graph*(*apply*) contains an argument and an answer, where the argument is itself a set, number pair.

2.2.2 Representing Functions as Equations

The graph representation of a function provides insight into its structure but is inconvenient to use in practice. In this text we use the traditional equational format for specifying a function. Here are the equational specifications for the functions described in examples 1-5 of Section 2.2.1:

1. $add: (\mathbb{N} \times \mathbb{N}) \rightarrow \mathbb{N}$
   $add(m,n) = m + n$
2. $duplicate: R \rightarrow (R \times R)$
   $duplicate(r) = (r, r)$
3. $whichpart: (\mathbb{B} + \mathbb{N}) \rightarrow S$
   $which\text{-}part(m) =$ cases m of

   is$\mathbb{B}(b) \rightarrow isbool$

   [] is$\mathbb{N}(n) \rightarrow isnum$

   end
4. $make\text{-}singleton: \mathbb{N} \rightarrow \mathbb{P}(\mathbb{N})$
   $make\text{-}singleton(n) = \{n\}$
5. $nothing: \mathbb{B} \cap \mathbb{N} \rightarrow \mathbb{B}$ has no equational definition since its domain is empty

The equational format is so obvious and easy to use that we tend to take it for granted. Nonetheless, it is important to remember that an equation $f(x) = \alpha$, for $f: A \rightarrow B$, *represents* a function. The actual function is determined by a form of evaluation that uses substitution and simplification. To use f's equational definition to map a specific $a_0 \in A$ to $f(a_0) \in B$, first, *substitute* a_0 for all occurrences of x in α. The substitution is represented as $[a_0/x]\alpha$. Second, *simplify* $[a_0/x]\alpha$ to its underlying value.

Here is the process in action: to determine the the value of $add(2,3)$, we first substitute 2 for m and 3 for n in the expression on the right-hand side of *add*'s equation, giving $add(2,3) = [3/n][2/m]m+n = 2+3$. Second, we simplify the expression 2+3 using our knowledge of the primitive operation + to obtain $2+3 = 5$. The substitution/simplification process produces a value that is consistent with the function's graph.

Often we choose to represent a function $f(x) = \alpha$ as $f = \lambda x.\, \alpha$; that is, we move the argument identifier to the right of the equals sign. The λ and . bracket the argument identifier. The choice of λ and . follows from tradition, and the format is called *lambda notation.* Lambda notation makes it easier to define functions such as *split-add*$: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ as *split-add*$(x) = \lambda y.\, x{+}y$ or even as *split-add*$= \lambda x.\lambda y.\, x{+}y$. Also, a function can be defined without giving it a name: $\lambda(x,y).\, x{+}y$ is the *add* function yet again. Functions written in the lambda notation behave in the same way as the ones we have used thus far. For example, $(\lambda(x,y).\, x{+}y)(2,3) = [3/y][2/x]x{+}y = 2{+}3 = 5$. Section 3.2.3 in the next chapter discusses lambda notation at greater length.

As a final addition to our tools for representing functions, we will make use of a function updating expression. For a function $f: A \rightarrow B$, we let $[\, a_0 \mapsto b_0\,]f$ be the function that acts just like f except that it maps the specific value $a_0 \in A$ to $b_0 \in B$. That is:

$([\, a_0 \mapsto b_0\,]f)(a_0) = b_0$
$([\, a_0 \mapsto b_0\,]f)(a) = f(a)$ for all other $a \in A$ such that $a \neq a_0$

2.3 SEMANTIC DOMAINS

The sets that are used as value spaces in programming language semantics are called *semantic domains.* A semantic domain may have a different structure than a set, but sets will serve nicely for most of the situations encountered in this text. In practice, not all of the sets and set building operations are needed for building domains. We will make use of *primitive domains* such as $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{B}$, . . ., and the following four kinds of *compound domains,* which are built from existing domains A and B:

1. Product domains $A \times B$
2. Sum domains $A + B$
3. Function domains $A \rightarrow B$
4. Lifted domains $A_\perp$, where $A_\perp = A \cup \{\perp\}$

The first three constructions were studied in the previous sections. The fourth, $A_\perp$, adds a special value $\perp$ (read ''bottom'') that denotes *nontermination* or ''no value at all.'' Since we are interested in modelling computing-related situations, the possibility exists that a function f applied to an argument $a \in A$ may yield no answer at all— $f(a)$ may stand for a nonterminating computation. In this situation, we say that f has functionality $A \rightarrow B_\perp$ and $f(a) = \perp$. The use of the codomain $B_\perp$ instead of B stands as a kind of warning: in the process of computing a B-value, nontermination could occur.

Including $\perp$ as a value is an alternative to using a theory of *partial functions*. (A *partial function* is a function that may not have a value associated with each argument in its domain.) A function f that is undefined at argument a has the property $f(a) = \perp$. In addition to dealing with undefinedness as a real value, we can also use $\perp$ to clearly state what happens when a function receives a nonterminating value as an argument. For $f: A_\perp \rightarrow B_\perp$, we write $f = \underline{\lambda} x.\, \alpha$ to denote the mapping:

$f(\perp) = \perp$
$f(a) = [\,a/x\,]\alpha$ for $a \in A$

The underlined lambda forces f to be a *strict* function, that is, one that cannot recover from a nonterminating situation. As an example, for $f : \mathbb{N}_\perp \rightarrow \mathbb{N}_\perp$, defined as $f = \underline{\lambda} n.0$, $f(\perp)$ is $\perp$, but for $g : \mathbb{N}_\perp \rightarrow \mathbb{N}_\perp$, defined as $g = \lambda n.0$, $g(\perp)$ is 0. Section 3.2.4 in the next chapter elaborates on nontermination and strictness.

2.3.1 Semantic Algebras

Now that the tools for building domains and functions have been specified, we introduce a format for presenting semantic domains. The format is called a *semantic algebra*, for, like the algebras studied in universal algebra, it is the grouping of a set with the fundamental operations on that set. We choose the algebra format because it:

1. Clearly states the structure of a domain and how its elements are used by the functions.
2. Encourages the development of standard algebra ‘‘modules’’ or ‘‘kits’’ that can be used in a variety of semantic definitions.
3. Makes it easier to analyze a semantic definition concept by concept.
4. Makes it straightforward to alter a semantic definition by replacing one semantic algebra with another.

Many examples of semantic algebras are presented in Chapter 3, so we provide only one here. We use pairs of integers to simulate the rational numbers. Operations for creating, adding, and multiplying rational numbers are specified. The example also introduces a function that we will use often: the expression $e_1 \rightarrow e_2 \,[\,]\, e_3$ is the *choice function,* which has as its value e_2 if $e_1 = true$ and e_3 if $e_1 = false$.

2.1 Example: *Simulating the rational numbers*

Domain $Rat = (\mathbb{Z} \times \mathbb{Z})_\perp$

Operations

$makerat: \mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow Rat)$

$makerat = \lambda p.\lambda q.\ (q{=}0) \rightarrow \perp [\,]\ (p,q)$

$addrat: Rat \rightarrow (Rat \rightarrow Rat)$

$addrat = \underline{\lambda}(p_1,q_1).\underline{\lambda}(p_2,q_2).\ ((p_1 * q_2)+(p_2 * q_1),\ q_1 * q_2)$

$multrat: Rat \rightarrow (Rat \rightarrow Rat)$

$multrat = \underline{\lambda}(p_1,q_1).\underline{\lambda}(p_2,q_2).\ (p_1 * p_2,\ q_1 * q_2)$

Operation *makerat* groups the integers p and q into a rational p/q, represented by (p,q). If the denominator q is 0, then the rational is undefined. Since the possibility of an undefined rational exists, the *addrat* operation checks both of its arguments for definedness before performing the addition of the two fractions. *Multrat* operates similarly.

The following chapter explains, in careful detail, the notion of a domain, its associated construction and destruction operations, and its presentation in semantic algebra format. If you are a newcomer to the area of denotational semantics, you may wish to skip Chapter 3 and use it as a reference. If you decide to follow this approach, glance at Section 3.5 of the chapter, which is a summary of the semantic operations and abbreviations that are used in the text.

SUGGESTED READINGS

Naive set theory: Halmos 1960; Manna & Waldinger 1985
Axiomatic set theory: Devlin 1969; Enderton 1977; Lemmon 1969

EXERCISES

1. List (some of) the members of each of these sets:

   a. $\mathbb{N} \cap \mathbb{Z}$
   b. $\mathbb{Z} - \mathbb{N}$
   c. $\mathbb{B} \times (\mathbb{C} + \mathbb{B})$
   d. $\mathbb{N} - (\mathbb{N} \cup \mathbb{Z})$

2. Give the value of each of these expressions:

   a. *fst*(4+2, 7)
   b. *snd*(7, 7+*fst*(3–1, 0))
   c. cases in$\mathbb{N}$(3+1) of is$\mathbb{B}$(t) $\rightarrow$ 0 [] is$\mathbb{N}$(n) $\rightarrow$ n+2 end
   d. $\{\, true \,\} \cup (\mathbb{P}(\mathbb{B}) - \{\, \{true\} \,\})$

3. Using the extensionality principle, prove that set union and intersection are commutative and associative operations.

4. In ''pure'' set theory, an ordered pair $P = (x,y)$ is modelled by the set $P' = \{\, \{\, x \,\}, \{\, x,y \,\} \,\}$.

   a. Using the operations union, intersection, and set subtraction, define operations fst' and snd' such that $fst'(P') = x$ and $snd'(P') = y$.
   b. Show that for any other set Q' such that $fst'(Q') = x$ and $snd'(Q') = y$ that $P' = Q'$.

5. Give examples of the following functions if they exist. If they do not, explain why:

   a. a one-one function from $\mathbb{B}$ to $\mathbb{N}$; from $\mathbb{N}$ to $\mathbb{B}$.
   b. a one-one function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{R}$; from $\mathbb{R}$ to $\mathbb{N} \times \mathbb{N}$.
   c. an onto function from $\mathbb{N}$ to $\mathbb{B}$; from $\mathbb{B}$ to $\mathbb{N}$.
   d. an onto function from $\mathbb{N}$ to $\mathbb{Q}$; from $\mathbb{Q}$ to $\mathbb{N}$.

6. For sets R and S, show that:

   a. $R \times S \neq S \times R$ can hold, but $R \times S$ is always isomorphic to $S \times R$.
   b. $R + S \neq R \cup S$ always holds, but $R + S$ can be isomorphic to $R \cup S$.

7. Prove that the composition of two one-one functions is one-one; that the composition of two onto functions is onto; that the composition of two isomorphisms is an isomorphism. Show also that the composition of a one-one function with an onto function (and vice

versa) might not be either one-one or onto.

8. Using the definition of *split-add* in Section 2.2.1, determine the graphs of:

   a. $split\text{-}add(3)$
   b. $split\text{-}add(split\text{-}add(2)(1))$

9. Determine the graphs of:

   a. $split\text{-}sub : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$ such that $split\text{-}sub(x) = g$, where $g : \mathbb{Z} \to \mathbb{Z}$ is $g(y) = x - y$
   b. $split\text{-}sub : \mathbb{N} \to \mathbb{N} \to \mathbb{Z}$, where the function is defined in part a.

10. The previous two exercises suggest that there is an underlying concept for ''splitting'' a function. For a set D, we define $curryD : ((D \times D) \to D) \to (D \to (D \to D))$ to be $curryD(f) = g$, where $g(x) = h$, where $h(y) = f(x,y)$. Write out (part of) the graph for $curry\mathbb{B} : ((\mathbb{B} \times \mathbb{B}) \to \mathbb{B}) \to (\mathbb{B} \to (\mathbb{B} \to \mathbb{B}))$.

11. For $\mathbb{B} = \{\, true, false \,\}$ and $\mathbb{N} = \{\, 0, 1, 2, \cdots \,\}$, what are the functionalities of the functions represented by these graphs?

    a. $\{\, (true, 0), (false, 1) \,\}$
    b. $\{\, ((true, 0), (true, true)), ((true, 1), (true, false)), ((true, 2), (true, false)),$
    $\cdots, ((false, 0), (false, true)), ((false, 1), (false, false)), ((false, 2),$
    $(false, false)), \cdots \,\}$
    c. $\{\, (\{\, (true, true), (false, true) \,\}, true), (\{\, (true, true), (false, false) \,\}, false),$
    $\cdots, (\{\, (true, false), (false, true) \,\}, true), \cdots \,\}$

12. Use the definitions in Section 2.2.2 to simplify each of the following expressions:

    a. $make\text{-}singleton(add(3,2)) \cup \{\, 4 \,\}$
    b. $add(snd(duplicate(4)), 1)$
    c. $which\text{-}part(\mathrm{in}\mathbb{N}(add(2,0)))$
    d. $([\, 3 \mapsto \{\, 4 \,\} \,]make\text{-}singleton)(2)$
    e. $([\, 3 \mapsto \{\, 4 \,\} \,]make\text{-}singleton)(3)$

13. For the equational definition $fac(n) = (n{=}0) \to 1 \,[]\, n * fac(n-1)$, show that the following properties hold (hint: use mathematical induction):

    a. For all $n \in \mathbb{N}$, $fac(n)$ has a unique value, that is, *fac* is a function.
    b. For all $n \in \mathbb{N}$, $fac(n+2) > n$.

14. List the elements in these domains:

    a. $(\mathbb{B} \times \mathbb{B})_\perp$
    b. $\mathbb{B}_\perp \times \mathbb{B}_\perp$
    c. $(\mathbb{B} \times \mathbb{B}) + \mathbb{B}$
    d. $(\mathbb{B} + \mathbb{B})_\perp$

e. $\mathbb{B}_\perp + \mathbb{B}_\perp$
f. $\mathbb{B} \rightarrow \mathbb{B}_\perp$
g. $(\mathbb{B} \rightarrow \mathbb{B})_\perp$

15. Simplify these expressions using the algebra in Example 2.1:

    a. *addrat* (*makerat* (3) (2)) (*makerat* (1) (3))
    b. *addrat* (*makerat* (2) (0)) (*multrat* (*makerat* (3) (2)) (*makerat* (1) (3)))
    c. (λ*r*. *one*) (*makerat* (1) (0))
    d. ($\underline{\lambda}$*r*. *one*) (*makerat* (1) (0))
    e. ($\overline{\lambda}$(*r*, *s*). *addrat*(*r*) (*s*)) ((*makerat* (2) (1)), (*makerat* (3) (2)))
    f. (λ(*r*, *s*). *r*) ((*makerat* (2) (1)), (*makerat* (1) (0)))

16. The sets introduced in Section 2.1 belong to *naive set theory*, which is called such because it is possible to construct set definitions that are nonsensical.

    a. Show that the definition $\{ x \mid x \notin x \}$ is a nonsensical definition; that is, no set exists that satisfies the definition.
    b. Justify why the domain constructions in Section 2.3 always define sensical sets.