9.6 Morphisms

This little discussion is about some tools and techniques that can be used to compare two different entities for common properties. For example, if A is an alphabet, then we know that a string over A is different from a list over A. In other words, we know that A^* and lists(A) contain different kinds of objects. But we also know that A^* and lists(A) have a lot in common. For example, we know that the operations on A^* are similar to the operations on lists(A). We know that the algebra of strings and the algebra of lists both have an empty object and that they construct new objects in a similar way. In fact, we know that strings can be represented by lists.

On the other hand, we know that A^* is quite different from the set of binary trees over A. For example, the construction of a string is not at all like the construction of a binary tree.

The Transformation Problem

We would like to be able to decide whether two different entities are alike in some way. When two things are alike, we are often more familiar with one of the things. So we can apply our knowledge about the familiar one and learn something about the unfamiliar one. This is a bit vague. So let's start off with a general problem of computer science:

The Transformation Problem

Transform an object into another object with some particular property.

This is a very general statement. So let's look at a few interpretations. For example, we may want the transformed object to be "simpler" than the original object. This usually means that the new object has the same meaning as the given object but uses fewer symbols. For example, the expression $x + 1$ might be a simplification of $(x^2 + x)/x$, and the FP program f @ (true $\rightarrow$ c; d)

can be simplified to $f @ c$.

We may want the transformed object to act as the meaning of the given object. For example, we usually think of the meaning of the expression 3 + 4 as its value, which is 7. On the other hand, the meaning of the expression x + 1 is $x + 1$ if we don't know the value of x.

Whenever a light bulb goes on in our brain and we finally understand the meaning of some idea or object, we usually make statements like "Oh yes, I see it now" or "Yes, I understand." These statements usually mean that we have made a connection between the thing we're trying to understand and some other thing that is already familiar to us. So there is a transformation (i.e., a function) from the new idea to a familiar old idea.

Introductory Example: Semantics of Numerals

Suppose we want to describe the meaning of the base 10 numerals (i.e., nonempty strings of decimal digits) or the base 2 numerals (i.e., nonempty strings of binary digits). Let m_{ten} denote the "meaning" function for base 10 numerals, and let m_{two} denote the meaning function for base 2 numerals. If we can agree on anything, we probably will agree that $m_{\text{ten}}(16) = m_{\text{two}}(10000)$ and $m_{\text{ten}}(14) = m_{\text{two}}(1110)$. Further, if we let m_{rom} denote the meaning function for Roman numerals, then we probably also agree that $m_{\text{rom}}(\text{XII}) = m_{\text{ten}}(12) = m_{\text{two}}(1100)$.

For this example we'll use the set **N** of natural numbers to represent the meanings of the numerals. For base 10 and base 2 numerals, there may be some confusion because, for example, the string 25 denotes a base 10 numeral and it also represents the natural number that we call 25. Given that this confusion exists, we have

$$m_{\text{ten}}(25) = m_{\text{two}}(11001) = m_{\text{rom}}(\text{XXV}) = 25.$$

So we can write down three functions from the three kinds of numerals (the syntax) to natural numbers (the semantics):

$$m_{\text{ten}}: \text{DecimalNumerals} \rightarrow \mathbf{N},$$
$$m_{\text{two}}: \text{BinaryNumerals} \rightarrow \mathbf{N},$$

$$m_{\text{rom}}: \text{RomanNumerals} \rightarrow \mathbf{N} - \{0\}.$$

Can we give definitions of these functions? Sure. For example, a natural definition for m_{ten} can be given as follows: If $d_{\text{k}}\ d_{k-1}\ ...\ d_1 d_0$ is a base 10 numeral, then

$$m_{\text{ten}}(d_{\text{k}}\ d_{k-1}\ ...\ d_1 d_0) = 10^k d_{\text{k}} + 10^{k-1} d_{k-1} + ... + 10 d_1 + d_0.$$

Preserving Operations

What properties, if any, should a semantics function possess? Certain operations defined on numerals should be, in some sense, "preserved" by the semantics function. For example, suppose we let $+_{bi}$ denote the usual binary addition defined on binary numerals. We would like to say that the meaning of the binary sum of two binary numerals is the same as the result obtained by adding the two individual meanings in the algebra $\langle \mathbf{N}; + \rangle$. In other words, for any binary numerals x and y, the following equation holds:

$$m_{\text{two}}(x +_{bi} y) = m_{\text{two}}(x) + m_{\text{two}}(y).$$

The idea of a function preserving an operation can be defined in a general way. Let $f : A \rightarrow A'$ be a function between the carriers of two algebras. Suppose ω is an n-ary operation on A. We say that f ***preserves*** the operation ω if there is a corresponding operation ω' on A' such that, for every $x_1, ..., x_n \in A$, the following equality holds:

$$f(\omega(x_1, ..., x_n)) = \omega'(f(x_1), ..., f(x_n)).$$

Of course, if ω is a binary operation, then we can write the above equation in its infix form as follows:

$$f(x\ \omega\ y) = f(x)\ \omega'\ f(y).$$

For example, the binary numeral meaning function m_{two} preserves $+_{bi}$. We can write the equation using the prefix form of $+_{bi}$ as follows:

$$m_{\text{two}}(+_{bi}(x, y)) = +(m_{\text{two}}(x), m_{\text{two}}(y)).$$

Here's the thing to remember about an operation that is preserved by a function $f : A \to A'$: You can apply the operation to arguments in A and then use f to map the result to A', or you can use f to map each argument from A to A' and then apply the corresponding operation on A' to these arguments. In either case, you get the same result.

Figure 9.6.1 illustrates this property for two binary operators $\circ$ and $\circ'$. In other words, if $a \circ b = c$ in A, then $f(a \circ b) = f(c) = f(a) \circ' f(b)$ in A'.

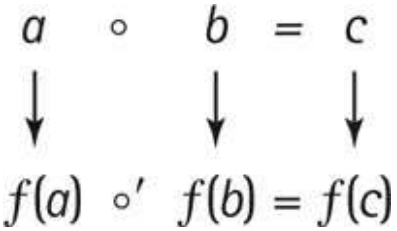

Figure 9.6.1 Preserving a binary operation.

Definition of *Morphism*

We say that $f : A \to A'$ is a ***morphism*** (also called a ***homomorphism***) if every operation in the algebra of A is preserved by f. If a morphism is injective, then it's called a ***monomorphism***. If a morphism is surjective, then it's called an ***epimorphism***. If a morphism is bijective, then it's called an ***isomorphism***. If there is an isomorphism between two algebras, we say that the algebras are ***isomorphic***. Two isomorphic algebras are very much alike, and, hopefully, one of them is easier to understand.

For example, m_{two} is a morphism from $\langle \text{BinaryNumerals}; +b_i \rangle$ to $\langle \mathbf{N}; + \rangle$. In fact, we can say that m_{two} is an epimorphism because it's surjective. Notice that distinct binary numerals like 011 and 11 both represent the number 3. Therefore, m_{two} is not injective, so it is not a monomorphism, and thus it is not an isomorphism.

Example 1 A Morphism

Suppose we define $f : \mathbf{Z} \to \mathbf{Q}$ by $f(n) = 2^n$. Notice that

$$f(n + m) = 2^{n+m} = 2^n \cdot 2^m = f(n) \cdot f(m).$$

So f is a morphism from the algebra $\langle \mathbf{Z}; + \rangle$ to the algebra $\langle \mathbf{Q}; \cdot \rangle$. Notice that $f(0) = 2^0 = 1$. So f is a morphism from the algebra $\langle \mathbf{Z}; +, 0 \rangle$ to the algebra $\langle \mathbf{Q}; \cdot, 1 \rangle$. Notice that $f(-n) = 2^{-n} = (2n)^{-1} = f(n)^{-1}$. Therefore, f is a morphism from the algebra $\langle \mathbf{Z}; +, -, 0 \rangle$ to the algebra $\langle Q; -, {}^{-1}, 1 \rangle$. It's easy to see that f is injective and that f is not surjective. Therefore, f is a monomorphism, but it is neither an epimorphism nor an isomorphism.

Example 2 The Mod Function

Let $m > 1$ be a natural number, and let the function $f : \mathbf{N} \to \mathbf{N}_m$ be defined by $f(x) = x \bmod m$. We'll show that f is a morphism from $\langle \mathbf{N}, +, \cdot\, 0, 1 \rangle$ to the algebra $\langle \mathbf{N}_m, +_m, \cdot_m, 0, 1 \rangle$. For f to be a morphism we must have $f(0) = 0$, $f(1) = 1$, and for all $x, y \in \mathbf{N}$:

$$f(x + y) = f(x) +_m f(y) \text{ and } f(x \cdot y) = f(x) \cdot_m f(y).$$

It's clear that $f(0) = 0$ and $f(1) = 1$. The other equations are just restatements of the congruences (9.3.1).

Example 3 Strings and Lists

For any alphabet A we can define a function $f : A^* \to \text{lists}(A)$ by mapping any string to the list consisting of all letters in the string. For example, $f(\Lambda) = \langle\,\rangle$, $f(a) = (a)$, and $f(aba) = \langle a, b, a \rangle$. We can give a formal definition of f as follows:

$$f(\Lambda) = \langle\,\rangle,$$

$$f(a \cdot t) = a :: f(t) \text{ for every } a \in A \text{ and } t \in A^*.$$

For example, if $a \in A$, then $f(a) = f(a \,.\, \Lambda) = a :: f(\Lambda) = a :: \langle\,\rangle = \langle a \rangle$. It's easy to see that f is bijective because any two distinct strings get mapped to two distinct lists and any list is the image of some string.

We'll show that f preserves the concatenation of strings. Let "cat" denote both the concatenation of strings and the concatenation of lists. Then we must verify that $f(\text{cat}(s, t)) = \text{cat}(f(s), f(t))$ for any two strings s and t. We'll do it by induction on the length of s. If $s = \Lambda$, then we have

$$f(\text{cat}(\Lambda, t)) = f(t) = \text{cat}(\langle\,\rangle, f(t)) = \text{cat}(f(\Lambda), f(t)).$$

Now assume that s has length $n > 0$ and $f(\text{cat}(u, t)) = \text{cat}(f(u), f(t))$ for all strings u of length less than n. Since the length of s is greater than 0, we can write $s = a \cdot x$ for some $a \in A$ and $x \in A^*$. Then we have

$$\begin{aligned}
f\,(\text{cat}\,(a \cdot x, t)) &= f\,(a \cdot \text{cat}\,(x, && \text{(definition of string cat)}\\
&\quad t)) \\
&= a :: f\,(\text{cat}\,(x, && \text{(definition of } f\text{)}\\
&\quad t)) \\
&= a :: \text{cat}\,(f(x), && \text{(induction assumption)}\\
&\quad f(t)) \\
&= \text{cat}\,(a :: f(x), && \text{(definition of list cat)}\\
&\quad f(t)) \\
&= \text{cat}\,(f(a \cdot x), && \text{(definition of } f\text{).}\\
&\quad f\,(t))
\end{aligned}$$

Therefore, f preserves concatenation. Thus f is a morphism from the algebra $\langle A^*; \text{cat}, \Lambda\rangle$ to the algebra $\langle \text{lists}(A); \text{cat}, \langle\,\rangle\rangle$. Since f is also a bijection, it follows that the two algebras are isomorphic.

Constructing Morphisms

Now let's consider the problem of constructing a morphism. We'll demonstrate the ideas with an example. Suppose we need a function $f : \mathbf{N}_8 \rightarrow \mathbf{N}_8$ with the property that $f\,(1) = 3$; and also, f must be a morphism from the algebra $(\mathbf{N}_8; +_8, 0)$ to itself, where $+_8$ is the operation of addition mod 8. We'll finish the definition of f . For f to be a morphism, it must preserve $+_8$ and 0. So we must set $f\,(0) = 0$. What value should we assign to $f\,(2)$? Notice that we can write $2 = 1 +_8 1$. Since $f\,(1) = 3$ and f must preserve the operation $+_8$, we can obtain the value $f\,(2)$ as follows:

$$f(2) = f(1 +_8 1) = f(1) +_8 f(1) = 3 +_8 3 = 6.$$

Now we can compute $f(3) = f(1 +_8 2) = f(1) +_8 f(2) = 3 +_8 6 = 1$. Continuing, we get the following values: $f(4) = 4$, $f(5) = 7$, $f(6) = 2$, and $f(7) = 5$. So the two facts $f(0) = 0$ and $f(1) = 3$ are sufficient to define f.

But does this definition of f result in a morphism? We must be sure that $f(x +_8 y) = f(x) +_8 f(y)$ for all $x, y \in \mathbf{N}_8$. For example, is $f(3 +_8 6) = f(3) +_8 f(6)$? We can check it out easily by computing the left- and right-hand sides of the equation:

$$f(3 +_8 6) = f(1) = 3 \text{ and } f(3) +_8 f(6) = 1 +_8 2 = 3.$$

Do we have to check the function for all possible pairs (x, y)? No. Our method for defining f was to force the following equation to be true:

$$f(1 +_8 \ldots +_8 1) = f(1) +_8 \ldots +_8 f(1).$$

Since any number in N_8 is a sum of 1's, we are assured that f is a morphism. Let's write this out for an example:

$$\begin{aligned} f(3 +_8 4) &= f(1 +_8 1 +_8 1 +_8 1 +_8 1 +_8 1 +_8 1) \\ &= f(1) +_8 f(1) +_8 f(1) +_8 f(1) +_8 f(1) +_8 f(1) +_8 f(1) \\ &= [f(1) +_8 f(1) +_8 f(1)] +_8 [f(1) +_8 f(1) +_8 f(1) +_8 f(1)] \\ &= f(1 +_8 1 +_8 1) +_8 f(1 +_8 1 +_8 1 +_8 1) \\ &= f(3) +_8 f(4). \end{aligned}$$

The above discussion might convince you that once we pick $f(1)$, then we know $f(x)$ for all x. But if the codomain is a different carrier, then things can break down. For example, suppose we want to define a morphism f from the algebra $\langle N_3; +_3, 0\rangle$ to the algebra $\langle \mathbf{N}_6;, +_6, 0\rangle$. Then we must have $f(0) = 0$. Now, suppose we try to set $f(1) = 3$. Then we must have $f(2) = 0$.

$$f(2) = f(1 +_3 1) = f(1) +_6 f(1) = 3 +_6 3 = 0.$$

Is this definition of f a morphism? The answer is No! Notice that $f(1 +_3 2) \neq f(1) +_6 f(2)$, because $f(1 +_3 2) = f(0) = 0$ and $f(1) +_6 f(2) = 3 +_6 0 = 3$. So morphisms are not as numerous as one might think.

Example 4 Language Morphisms

If A and B are alphabets, then a function $f : A^* \to B^*$ is called a ***language morphism*** if $f(\Lambda) = \Lambda$ and $f(uv) = f(u)f(v)$ for any strings $u, v \in A^*$. In other words, a language morphism from A^* to B^* is a morphism from the algebra $(A^*;$ cat, $\Lambda)$ to the algebra $(B^*;$ cat, $\Lambda)$. Since concatenation must be preserved, a language morphism is completely determined by defining the values $f(a)$ for each $a \in A$.

For example, let $A = B = \{a, b\}$ and define $f : \{a, b\}^* \to \{a, b\}^*$ by setting $f(a) = b$ and $f(b) = ab$. Then we can make statements like

$$f(bab) = f(b)f(a)f(b) = abbab \text{ and } f(b^2) = (ab)^2.$$

Language morphisms can be used to transform one language into another language with a similar grammar. For example, the grammar

$$S \to aSb \mid \Lambda$$

defines the language $\{a^n b^n \mid n \in \mathbf{N}\}$. Since $f(a^n b^n) = b^n(ab)^n$ for $n \in \mathbf{N}$, the set $\{a^n b^n \mid n \in \mathbf{N}\}$ is transformed by f into the set $\{b^n(ab)^n \mid n \in \mathbf{N}\}$. This language can be generated by the grammar $S \to f(a)Sf(b) \mid f(\Lambda)$, which becomes $S \to bSab \mid \Lambda$.

Example 5 Casting Out by Nines, Threes, etc.

An old technique for finding some answers and checking errors in some arithmetic operations is called “casting out by nines.” We want to study the technique and see why it works (so it’s not magic). Is 44,820 divisible by 9? Is $43 \cdot 768 + 9579 = 41593$? We can use casting out by nines to answer yes to the first question and no to the second question. How does the idea work? It’s a consequence of the following result:

Casting Out by Nines **(9.6.1)**

If K is a natural number with decimal representation $d_n \ldots d_0$, then

$$K \bmod 9 = (d_n \bmod 9) +_9 \ldots +_9 (d_0 \bmod 9).$$

Proof: For the two algebras $\langle \mathbf{N}; +, \cdot, 0, 1\rangle$ and $\langle \mathbf{N}_9; +_9, \cdot_9, 0, 1\rangle$, the function f :

$\mathbf{N} \rightarrow \mathbf{N}_9$ defined by $f(x) = x \bmod 9$ is a morphism. We can also observe that $f(10) = 1$, and in fact $f(10^n) = 1$ for any natural number n. Now, since $d_n \ldots d_0$ is the decimal representation of K, we can write

$$K = d_n \cdot 10^n + \ldots + d_1 \cdot 10 + d_0.$$

Now apply f to both sides of the equation to get the desired result.

f(k)=f(dn · 10n+...+d1 · 10+d0)=f(dn) · 9f(10n)+9...+9f(d1) · 9f(10)+9f(d0)=f(dn

Casting out by nines works because 10 mod 9 = 1. Therefore, casting out by threes also works because 10 mod 3 = 1. In general, for a base B number system, casting out by the predecessor of B works if we have the equation

$$B \bmod \text{pred}(B) = 1.$$

For example, in octal, casting out by sevens works. (Do any other numbers work in octal?) But in binary, casting out by ones does not work because 2 mod 1 = 0.

Learning Objectives

- ♦ Be familiar with morphisms of algebras.

Review Questions

- ♦ What is a morphism?
- ♦ What is an isomorphism?