In this chapter we describe the additional features in Anna for specifying and annotating operations on arrays, records, and access objects (also sometimes called pointers for brevity). Such annotations require language features that are not available in Ada. Typically, to specify operations on composite objects, it is convenient to have expressions denoting the composite values in which only the important component values appear. The values of other components may be unknown or irrelevant. (Ada aggregates, for example, are complete composite values, and require all component values to be given.) Array states and record states are introduced in Anna for this purpose.

We also need expressions that denote parts of the Ada computation state that are not accessible to the Ada programmer. For example, in the case of an access type, we will define the concept of a collection of the objects designated by all the values of that access type. Collections enable us to write annotations that constrain side effects of pointer manipulations and aliasing between pointer structures. It is difficult to express such constraints without collections.

These new features are similar to the package state concept discussed in Chapter 4, so we have delayed describing these features until after the discussion of packages.

7.1 Array States

Specifications of operations on arrays and records often refer to values resulting from sequences of operations. To aid writing such specifications, expressions denoting the value of an array or record should contain the
sequence of operations leading to that value. Since array and record types can be viewed as simple examples of Anna package state types, a notation similar to the successor state notation for package states suits this purpose very well. Anna provides such a notation in the form of two new kinds of expressions called *array states* and *record states*.

To explain array states, consider one-dimensional arrays of type VECTOR:

```ada
type VECTOR is array (INTEGER range <>) of INTEGER;
```

The value of a vector A after U has been placed in component I is denoted by an array state

```ada
A[I => U].
```

This is an expression of type VECTOR written in an Ada-like notation. Its value is defined more formally by the following indexing rule.

**• Indexing Rule**

```ada
A[J => V](K) = if K = J then V else A(K) end if.
```

Consequently, the following Ada assignment and Anna assertion are always consistent:

```ada
A(I) := U;
--| A = in A[I => U];
```

Array state expressions may be used only in annotations. Here are some examples of their use.

**Example: Values of array states.**

```ada
declare
  A : VECTOR(1 .. 3) := (1, 2, 3);
begin
  A(3) := A(3) - A(2);
  --| A = (1, 2, 1); (1)
  --| A = in A[3 => in A(1)]; (2)
  --| A[3 => A(3) + A(2)] = in A; (3)
end;
```

**Commentary**

Assertion (1) gives the value of A as an Ada aggregate. Assertion (2) expresses that this value is equal to an array state that encodes a different computation on A to get the value. Assertion