Arrays as Bounded Sequences

Corky Cartwright

Department of Computer Science

Rice University

Motivation

- Computational models are built using a small number of primitives (just like objects in mathematics), namely sequences (contiguous and linked), trees, graphs (multi-linked structures), and functions.
- Essentially the same data representations and operations on those representations appear over and over again
- Libraries and frameworks catalog these repeated bodies of code.
- Unfortunately, the abstraction mechanisms in our programming languages limit the reuse of cataloged code. Our parameterization mechanisms are restrictive and add overhead.

Arrays

- An *array* of *T* is a contiguous (as allocated in memory) sequence of data values of type *T* . The length of the array is fixed at the time the array is created. The type *T* can be a fixed-length primitive machine type (represented by a fixed-length sequence of bytes) or a fixed length address. The key issue is that the representation of *T* has a fixed size and hence the representation of the array has a fixed size.
- The Java type for an array of *T* is written `T[]`. Arrays are a special form of object. The array class cannot be extended, but arrays can be used anywhere arbitrary objects can be used. When an array is allocated, the size of the array is inserted between the square brackets: `new int[17]` . By default, the elements of the array are initialized to the default value for the element type (whatever the bit representation 0 means for that type).
- An alternate syntax for the `new` operation on arrays `new int[] {0,1,2}` explicitly lists the initial contents of the array and leaves the length implicit.

Arrays cont.

- To extract an element from an array `a`, we simply use the postfix operation `[`*`index`*`]` where *`index`* is an integer expression with a value in the range [0, n) For example,

```
new int[] {0,1,2} [0] => 0
new int[] {0,1,2} [1] => 1
new int[] {0,1,2} [2] => ArrayIndexOutOfBoundsException
```

- To update an array, the index expression for accessing the element to be update is used on the left-hand size of a conventional assignment statement:

```
int[] a = new int[2];
a[0] = 0;
a[1] = 1;
```

- The meanings of `equals` and `toString` in array classes are the standard `Object` defaults. Beware. `equals` means object identity and `toString` prints `L<eltType>@address`, where `eltType` is a Java abbreviation for the element type and `address` is the hashCode of the object expressed in hexadecimal.

Arrays cont.

- The only interesting member of array classes is the field `length`. Beware: object classes similar to array typically use a `size()` method instead.
- Since the array class cannot be extended and the default members of the class provide little functionality beyond arrays in C, most Java programmers use `ArrayList` (preferred) or `Vector` (archaic) instead. Arrays are lighter weight than corresponding object classes and are the only first-class generic type in Java, so they are important in practice. Moreover, computations on `ArrayList` involve essentially the same models and tradeoffs as computations on arrays.
- To explain how to write clean code for processing arrays, we identify them with lists and use some cleverness in representing the tails of arrays..
- It is straightforward to write tail-recursive code to perform array computations from the perspective that arrays are restricted lists. But to produce good array code in Java we must go one step farther and convert that tail-recursive code to loop code.

Recipes for Processing Arrays

- Conceptually arrays are a restricted form of list where:
  - There is no `rest()` operation, but there is an indexed accessor operation (often called get).
  - Updates are destructive and can only be performed on elements not on tails.
- The array type overlooks the conceptual roots of arrays as lists, which have an itype *inductive* definition. To recover these roots, we need to define some conceptual operations on arrays
- Given an array $A = \{a_0, ..., a_n\}$, we define the *slice* $A\langle k,l\rangle$ where $0 \leq k \leq l \leq n$ as the sequence of elements $a_k, ..., a_{l-1}$. We can pass a slice as an argument using A, k, and l. If we fix k at zero or l at n, then we only need pass A and and index i.
- We can write functions on arrays in essentially the same way as on lists if we encode the results of rest operations using slices represented by the array object and indices.

Recipes for Processing Arrays cont.

- Assume that we want to sum the elements of an `int[]` array. We can express the naive solution using structural recursion as follows:

```
class ArrayUtil {
  public static int sum(int[] a) { return sumHelp(a, 0); }
  /** sumHelp(a, i) sums a[i],..,a[a.length - 1]  */
  public static int sumHelp(int[] a, int i) {
    if (i >= a.length) then return 0;
    else return a[i] + sumHelp(a, i+1);
  }
}
```

  We introduced a help function because auxiliary arguments are required to describe list tails.

- We can improve this naïve program by converting it to tail-recursive form:

```
class ArrayUtil {
  public static int sum(int[] a) { return sumHelp(a, 0, 0); }
  /** Returns the sum of A[0],…,A[a.length-1] given 0 <= i < a.length
      and accum = A[0],…A[i-1] */
  public static int sumHelp(int[] a, int i, int accum) {
    if (i >= a.length) return accum;
    else return sumHelp(a, i+1, a[i] + accum);
  }
}
```

Tail Recursion Is Not Enough!

- Java does not generally optimize tail calls (it is implementation dependent and unsupported in Sun JVMs which means you cannot rely on tail recursion.) In my opinion, this was a grievous error in the definition of Java (an opinion shared by Guy Steele who wrote the original edition of the JLS).
- Implication: must use Java loops instead of tail recursive help functions to get good performance and memory utilization.

Connection between Loops and Tail Recursion

- What is a while loop? The code
  `while (test) update`
  evaluates the boolean expression *`test`* and falls through to the next statement if *`test`* is false. Otherwise, it evaluates *`update`* and executes the loop again.
- If we model mutations in the code containing the while loop as changes to fields of the enclosing object, then the while loop is equivalent to calling a method

```
void whileFun() {
if (! test) return;
else {
  update;
  whileFun();
}
```

- Note that this function template is simply a restricted form of tail recursion. Let's convert our `sum` function to while loop form. The state will be the pair of variables `(i, accum)` formulated as parameters in the tail recursive code.

Loops and Tail Recursion

- Given

```
class ArrayUtil {
  public static int sum(int[] a) { return sumHelp(a, 0, 0); }
  /** Returns the sum of A[0],…,A[a.length-1] given 0 <= i < a.length
      and accum = A[0],…A[i-1] */
  public static int sumHelp(int[] a, int i, int accum) {
    if (i >= a.length) return accum;
    else return sumHelp(a, i+1, a[i] + accum);  /* update i, accum */
  }
}
```

- The same code in loop form (no recursion):

```
class ArrayUtil {
   public static int sum(int[] a) {
     /** Returns the sum of A[0],…,A[a.length-1] */
     accum = 0; i = 0; /* bind parms i, accum */
     /* Invariant: accum = a[0] + … + a[I-1] */
     while (i < a.length)  {
       accum = accum + a[i]; i++ /* update i, accum */
     }
     return accum;
   }
 }
```

Condensed Loop for `sum`

- In Java/C

```
for ( init; test; incr ) body
```

- abbreviates

```
init;
while (test) {
  body;
  incr;
}
```

- Hence, we can rewrite our loop:

```
class ArrayUtil {
  public static int sum(int[] a) {
    /** Returns the sum of A[0],…,A[a.length-1] */
    int accum = 0;
    /*  Invariant: accum = a[0] + … + a[I-1] */
    for (int i = 0;  i < a.length; i++) {
      accum = accum + a[i];
    }
    return accum;
  }
}
```

For Next Class

- Homework due on Wednesday. Should be working on reformulating interpreter methods as visitors.
- To handle big tests (like `bigDataX`), you need to enlarge the stack of the DrJava interactions JVM. Insert the argument string
  `-Xss64M`
  in the dialog box labeled JVM args for Interactions JVM in the Miscellaneous panel of DrJava Preferences.