Chapter 17

Introduction

If we were to identify the most fundamental ideas in computer science, we would probably end up converging on a list that includes abstract data types (ADTs) and the data structures used to implement them. Unlike an algorithm the data structures need to balance the cost of different functions within the ADT interface, often involving a tradeoff.

This part covers an ADT that mimics the mathematical concept of a sequence.

ADT for Sequences. Recall that an ADT is defined in terms of an interface consisting of a collection of functions (and possibly values) on a given abstract type, and without reference to the implementation. ADT Chapter defines such an interface for sequences, specifying the type and semantics for each of the functions. Many of the functions we define, such as *map*, *reduce*, *filter* and *scan*, are particularly useful in developing parallel algorithms. The chapter also covers a shorthand syntax we use in this book for these functions.

Cost Specifications for Sequences. Beyond the interface itself we need to know something about the costs of each of the functions. As discussed in an earlier chapter , a data type can have many different implementations with different asymptotic costs, and the idea of a cost specification is to capture the cost of a class of implementations, without reference to the actual implementation. In a cost specification, the costs (work and span) for each function are defined asymptotically as a function of size (number of elements, in the case of sequences). Costs Chapter covers three different cost specifications for the sequence ADT. One is based on arrays, one on trees, and one on lists. None of these fully dominate each other. In all cases some functions are asymptotically more expensive in one and some in the other.

Implementations of Sequences. Cost specifications are meant to abstract away from the specific implementation and be useful for users of an ADT, but someone needs to implement a data structure that abide by the bounds. In Array Sequences Chapter we describe how to match the bounds for the array based cost specification. We start with a small set of

primitive operations with given costs and show how to implement the rest of the interface within the bounds given by the specification.

In Examples Chapter , we present some examples using the sequences ADT, including several algorithms for computing prime numbers. In Ephemeral Sequences Chapter , we describe a reduced interface for sequences and a cost specification for the interface that makes updates faster. The cost specification is different from the others in that it is non-pure—costs will depend on the context.

1 Defining Sequences

From a mathematical standpoint it is possible to define sequences in several ways. One way is to use set theory. Another way to take a more formal approach based on constructive logic and define them inductively. Here we use basic set theory.

Mathematically, a sequence is an enumerated collection. As with a set, a sequence has ***elements***. The ***length*** of the sequence is the number of elements in the sequence.

Sequences allow for repetition: an element can appear at multiple positions. The position of an element is called its ***rank*** or its ***index***. Traditionally, the first element of the sequence is given rank 1, but, being computer scientists, we start at 0.

In mathematics, sequences can be finite or infinite but for our purposes in this book, finite sequences suffice. We therefore consider finite sequences only.

We define a sequence as a function whose domain is a contiguous set of natural numbers starting at zero. This definition, stated more precisely below, allows us to specify the semantics of various operations on sequences succinctly.

Definition 17.1 (Sequences)**.** An α ***sequence*** is a mapping (function) from $\mathbb{N}$ to α with domain $\{0, \ldots, n-1\}$ for some $n \in \mathbb{N}$.

Example 17.1. Let $A = \{0, 1, 2, 3\}$ and $B = \{\texttt{'a'}, \texttt{'b'}, \texttt{'c'}\}$. The function

$$R = \{(0, \texttt{'a'}), (1, \texttt{'b'}), (3, \texttt{'a'})\}$$

from A to B has domain $\{0, 1, 3\}$. The function is not a sequence, because its domain has a gap.

The function

$$Z = \{(1, \texttt{'b'}), (3, \texttt{'a'}), (2, \texttt{'a'}), (0, \texttt{'a'})\}$$

from A to B is a sequence. The first element of the sequence is $\texttt{'a'}$ and thus has rank 0. The second element is $\texttt{'b'}$ and has rank 1. The length of the sequence is 4.

Remark. Notice that in the definition sequences are parametrized by the type (i.e., set of possible values) of their elements.

Note. This mathematical definition might seem pedantic but it is useful for at least several reasons.

- It allows for a concise and precise definition of the semantics of the functions on sequences.
- Relating sequences to mappings creates a symmetry with the abstract data types such as tables or dictionaries for representing more general mappings.

Syntax 17.2 (Sequences and Indexing)**.** As in mathematics, we use a special notation for writing sequences. The notation

$$\langle\, a_0, a_1, \ldots, a_{n-1} \,\rangle$$

is shorthand for the sequence

$$\{(0, a_0), (1, a_1), \ldots, ((n-1), a_{n-1})\}\,.$$

For any sequence a

- $a[i]$ refers to the element of a at position i,
- $a[l \cdots h]$ refers to the subsequence of a restricted to the position between l and h.

Example 17.2. Some example sequences follow.

- For the sequence $a = \langle\, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 \,\rangle$, we have
  - $a[0] = 2$,
  - $a[2] = 5$, and
  - $a[1 \cdots 4] = \langle\, 3, 5, 7, 11 \,\rangle$.
- A $\mathbb{Z} \to \mathbb{Z}$ function sequence:

$$\begin{array}{ll}\langle & \texttt{lambda}\; x\,.\,x^2, \\ & \texttt{lambda}\; y\,.\,y+2, \\ & \texttt{lambda}\; x\,.\,x-4 \\ \rangle\,. & \end{array}$$

Syntax 17.3 (Ordered Pairs and Strings)**.** We use special notation and terminology for sequences with two elements and sequences of characters.

- An ***ordered pair*** (x, y) is a pair of elements in which the element on the left, x, is identified as the ***first*** entry, and the one on the right, y, as the ***second*** entry.
- We refer to a sequence of characters as a ***string***, and use the standard syntax for them, e.g., $'c_0c_1c_2\ldots c_{n-1}'$ is a string consisting of the n characters $c_0, \ldots, c_{n-1}$.

Example 17.3 (Ordered Pairs and Strings)**.**

- A character sequence, or a string: $\langle\, 's', 'e', 'q' \,\rangle \equiv 'seq'$.
- An integer-and-string sequence: $\langle\, (10, 'ten'), (1, 'one'), (2, 'two') \,\rangle$.
- A string-and-string-sequence sequence: $\langle\, \langle\, 'a' \,\rangle, \langle\, 'nested', 'sequence' \,\rangle \,\rangle$.