The Description and Manipulation of Sequences*

Ralph E. Griswold

TR 83-15a

October 26, 1983; revised June 9, 1984

Department of Computer Science

The University of Arizona

Tucson, Arizona 85721

*This work was supported by the National Science Foundation under Grant MCS81-01916.

The Description and Manipulation of Sequences

The following development of sequences and operations on them is motivated by manipulating sequences in computational contexts. Several commonly used notations are adapted to that context. Some specific notational details used here were influenced by the Icon programming language [1] and earlier work on result sequences in that language [2, 3].

There are obvious analogies between some aspects sequences as viewed here and those of Lisp, APL, ISWIM [4], and FP [5]. The sequences here should be thought of as abstract mathematical objects rather than as data structures. In particular, the development here considers the processes by which sequences can be produced. This formulation is designed to provide a foundation for a programming language to manipulate sequences as data objects. The emphasis here, however, is on notation related to sequences and not on programming facilities. An experimental language based on these concepts is described in a companion report [6].

1. Basic Concepts and Notation

1.1 Sequences

A sequence is an ordered list of values $x_1, x_2, x_3, \ldots$. The values in a sequence are called its *elements* . Sequences may be finite or infinite. Examples of infinite sequences are the positive integers,

$$\mathcal{I}_+ = \{1, 2, 3, \ldots\}$$

and the nonnegative integers,

$$\mathcal{I}_0 = \{0, 1, 2, \ldots\}$$

Another familiar infinite sequence consists of the Fibonacci numbers:

$$\mathcal{F} = \{1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots\}$$

A sequence consisting of a single element is called a *unit sequence*, as in {1}. The empty sequence containing no elements, { }, is denoted by Φ.

Frequently the elements of sequences are integers, as in the examples above, and the corresponding sequences are called *integer sequences* . The elements of sequences, however, may be real numbers, strings, or other *scalar* values. The elements of sequences also can be sequences. An example is

$$\{1, \{1, 1\}, \{1, 1, 1\}, \ldots\}$$

A sequence containing a sequence as an element is not the same as a sequence consisting of the corresponding scalar values. In particular,

$$\{x , \{ y \}\} \neq \{x , y \}$$

Different letters are used to distinguish different types of values in this report. For example, i and j denote integers, s denotes strings, while x , y , and z denote scalar values of unspecified types. The examples here are confined to integers and strings. Corresponding capital letters are used to denote sequences.

The number of elements in a sequence X is its *length* and is denoted by $| X |$. For example, $| \mathcal{I}_+ | = \infty$ and $| \Phi | = 0$. Note that

$$| \{\{1, 2\}\} | = 1$$

Two sequences

$$X = \{x_1, x_2, x_3, \ldots\}$$

and

$$Y = \{y_1, y_2, y_3, \ldots\}$$

are equal, denoted by $X = Y$, if and only if $|X| = |Y|$ and

$$x_i = y_i, i = 1, 2, 3, \ldots, |X|$$

For $1 \leq i \leq |X|$, $X!i$ denotes the i th element of X. For example,

$$\mathcal{F}!6 = 8$$

$X!i$ is undefined if $i < 1$ or $i > |X|$.

1.2 Concatenation

The *concatenation* of two sequences

$$X = \{x_1, x_2, x_3, \ldots\}$$

and

$$Y = \{y_1, y_2, y_3, \ldots\}$$

is denoted by $X \oplus Y$ and consists of

$$\{x_1, x_2, x_3, \ldots, y_1, y_2, y_3, \ldots\}$$

For example,

$$\{1, 2, 3\} \oplus \{6, 7, 8, 9\} = \{1, 2, 3, 6, 7, 8, 9\}$$

The sequences in a concatenation may be infinite. For example,

$$\{-2, -1\} \oplus \mathcal{G}_0 = \{-2, -1, 0, 1, 2, 3, \ldots\}$$

Φ is the identity with respect to the concatenation of sequences: $(X \oplus \Phi) = (\Phi \oplus X) = X$. The concatenation of a sequence with itself is denoted by superscripts: $(X \oplus X) = X^2$. A sequence can be concatenated with itself indefinitely: $(X \oplus X \oplus \ldots) = X^\infty$. Note that $\Phi^\infty = \Phi$.

1.3 Subsequences

X is a *subsequence* of Y, written $X \subseteq Y$, if there exist sequences U and W such that $Y = U \oplus X \oplus W$. X is a *proper* subsequence of Y, $X \subset Y$, if there exist subsequences U and W such that $Y = U \oplus X \oplus W$ and $|U| > 0$ or $|W| > 0$. X is an *initial* subsequence of Y if there exists a sequence W such that $Y = X \oplus W$. X is a *terminal* subsequence of Y if there exists a sequence U such that $Y = U \oplus X$

For $1 \leq i \leq j \leq |X|$, $X_{i\ j}$ denotes the subsequence of X consisting of elements i through j, inclusive. For example,

$$\mathcal{F}_{3\ 5} = \{2, 3, 5\}$$

and

$$\mathcal{F}_{5\ \infty} = \{5, 8, 13, 21, 34, \ldots\}$$

If $j > |X|$ then $X_{i\ j} = X_{i\ |X|}$. If $i > |X|$, $X_{i\ j} = \Phi$.

Two common kinds of subsequences result from *pre-truncation* and *post-truncation*, denoted respectively by

$$(X \downarrow i) \equiv X_{i+1\ |X|}, i \geq 0$$

$$(X \uparrow i) \equiv X_{1\ i}, i > 0$$

$\downarrow X$ is an abbreviation for $X \downarrow 1$.

1.4 Strings

Strings are used in a number of the examples that follow. They are shown in bold, as in the string sequence

{a, ab, b, ba}

An example of an infinite string sequence is the lexically ordered closure of **a**, **b**, and **c**:

$$\mathfrak{ABC} = \{\phi, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{aa}, \mathbf{ab}, \mathbf{ac}, \mathbf{ba}, \mathbf{bb}, \mathbf{bc}, \mathbf{ca}, \mathbf{cb}, \mathbf{cc}, \mathbf{aaa}, \ldots\}$$

where ϕ denotes the empty string.

The concatenation of two strings s_1 and s_2 is denoted by $s_1 \parallel s_2$. For example,

$$\mathbf{aaa} \parallel \mathbf{baaa} = \mathbf{aaabaaa}$$

s^i indicates a string consisting of s concatenated i times. For example, $\mathbf{a}^3 = \mathbf{aaa}$. $\mathbf{a}^0 = \phi$. The length of a string is the number of characters that it contains and is denoted by $|s|$. For example, $|\mathbf{aaabaaa}| = 7$. $|\phi| = 0$.

2. Generators

In many cases a sequence consists of the values of a function $f(i)$ for $i = 1, 2, 3, \ldots$:

$$\{f(1), f(2), f(3), \ldots\}$$

Such sequences are denoted by[1]

$$[f(i)]$$

The expression in the brackets is called the *generation expression*. Note that i takes on successive values from the *generation sequence* $\mathcal{G}_+$. Examples of generators are

$$[i] = \mathcal{G}_+$$

$$[i - 1] = \{0, 1, 2, \ldots\} = \mathcal{G}_0$$

$$[i^2] = \{1, 4, 9, \ldots\}$$

$$[i \bmod 3] = \{1, 2, 0, 1, 2, 0, \ldots\} = \{1, 2, 0\}^\infty$$

$$[\mathbf{a}^i] = \{\mathbf{a}, \mathbf{aa}, \mathbf{aaa}, \ldots\}$$

Note that

$$[1] = \{1, 1, 1, \ldots\} = \{1\}^\infty$$

and

$$[\mathbf{aa}] = \{\mathbf{aa}, \mathbf{aa}, \mathbf{aa}, \ldots\} = \{\mathbf{aa}\}^\infty$$

2.1 Generation Sequences

In the examples above, the generation sequence is implicit and is $\mathcal{G}_+$. Any sequence can be used as a generation sequence. If such a sequence is specified, it precedes the generation expression and is separated from it by a colon:

$$[I : X]$$

Note that

$$[I : f(i)] = \{f(I!1), f(I!2), f(I!3), \ldots\}$$

[1] The usual notation employs angular brackets, as in

$$\langle f(i) \rangle$$

Square brackets are used here for typographical clarity.

Some examples of the use of generation sequences are

$$[\mathcal{I}_+ : f(i)] \equiv [f(i)]$$

$$[\mathcal{I}_0 : f(i)] = [f(i-1)]$$

$$[\mathcal{F} : f(i)] = \{f(1), f(1), f(2), f(3), \ldots\}$$

$$[[i^2] : \mathcal{F}!i] = [\{1, 4, 9, \ldots\} : \mathcal{F}] = \{1, 3, 34, \ldots\}$$

The generation sequence can be finite, as in

$$[\{1, 3, 5\} : \mathcal{ABC}!i] = \{\phi, \mathbf{b}, \mathbf{aa}\}$$

and

$$[\mathcal{I}_+ \uparrow 5 : \mathcal{ABC}!i] = \{\phi, \mathbf{a}, \mathbf{b}, \mathbf{aa}, \mathbf{ab}\}$$

2.2 Generation Expressions

The generation expression may be a sequence, in which case the generator denotes the concatenation of the sequences over the generation sequence. For example,

$$[\{i, i\}] = \{1, 1\} \oplus \{2, 2\} \oplus \{3, 3\} \oplus \ldots = \{1, 1, 2, 2, 3, 3, \ldots\}$$

Note that $[\Phi] = \Phi$ and $[X] = X^{\infty}$.

It is convenient to view the generation expression as always producing a sequence with the convention that a scalar value in place of a sequence is an abbreviation for the corresponding unit sequence. Thus

$$[f(i)] \equiv [\{f(i)\}] = \{f(1)\} \oplus \{f(2)\} \oplus \{f(3)\} \oplus \ldots$$

$$= \{f(1), f(2), f(3), \ldots\}$$

This interpretation of scalar values in sequence contexts is merely an abbreviation. Elements are not equivalent to their corresponding unit sequences. That is, $x \neq \{x\}$.

2.3 Generation Variables

The generation expression may contain *unbound* variables in addition to the bound *generation variable* that takes on values from the index sequence. The generation variable is assumed to be i unless another variable is specified using lambda notation, as in

$$[\lambda(j)\, i + j] = \{i + 1, i + 2, i + 3, \ldots\}$$

Similarly,

$$[\mathcal{F} : \lambda(j)\, i + j] = \{i + 1, i + 1, i + 3, i + 5, \ldots\}$$

Note that

$$[\lambda(j)\, f(j)] \equiv [f(i)]$$

The generation sequence need not be an integer sequence. An example is

$$[\mathcal{ABC} : \lambda(s)\, s^2] = \{\phi, \mathbf{aa}, \mathbf{bb}, \mathbf{cc}, \mathbf{aaaa}, \ldots\}$$

That is, the generation variable is treated in a purely formal manner.

2.4 Nested Generators

Generators may be nested to obtain "cross-product" sequences. For example, if

$$U = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$$

and

$$L = \{\mathbf{a}, \mathbf{b}\}$$

then

$$[\lambda(i)[\lambda(j)\ U!i\ \|\ L!j]] = \{\mathbf{Aa}, \mathbf{Ab}, \mathbf{Ba}, \mathbf{Bb}, \mathbf{Ca}, \mathbf{Cb}\}$$

However,

$$[\lambda(i)[\lambda(j)\ U!j\ \|\ L!i]] = \{\mathbf{Aa}, \mathbf{Ba}, \mathbf{Ca}, \mathbf{Ab}, \mathbf{Bb}, \mathbf{Cb}\}$$

In nested generators, one generation sequence can control another, as in

$$\begin{aligned}[\mathcal{G}_0 : \lambda(j)[(\mathcal{G}_+ \downarrow j) \uparrow (j+1) : i]] &= [(\mathcal{G}_+ \downarrow 0) \uparrow 1 : i] \oplus [(\mathcal{G}_+ \downarrow 1) \uparrow 2 : i] \oplus [(\mathcal{G}_+ \downarrow 2) \uparrow 3 : i] \oplus \ldots \\ &= \{1\} \oplus \{2, 3\} \oplus \{3, 4, 5\} \oplus \ldots \\ &= \{1, 2, 3, 3, 4, 5, 4, 5, 6, 7, \ldots\}\end{aligned}$$

3. Operations on Sequences

3.1 Extension of Operations on Scalar Values to Sequences

An operation on scalar values, such as $i + j$, is extended to sequences by distributing it over the corresponding elements of the two sequences to produce a "dot product" sequence. For example, given two integer sequences

$$I = \{i_1, i_2, i_3, \ldots\}$$

and

$$J = \{j_1 0, j_2, j_3, \ldots\}$$

then

$$(I + J) = \{i_1 + j_1, i_2 + j_2, i_3 + j_3, \ldots\}$$

Similarly,

$$(\{\mathbf{a}, \mathbf{ab}, \mathbf{c}\}\ \|\ \{\mathbf{c}, \mathbf{b}, \mathbf{a}\}) = \{\mathbf{ac}, \mathbf{abb}, \mathbf{ca}\}$$

Likewise

$$\mathfrak{ABC}\ \|\ \mathfrak{ABC} = \{\phi, \mathbf{aa}, \mathbf{bb}, \mathbf{cc}, \mathbf{aaaa}, \mathbf{abab}, \mathbf{acac}, \ldots\}$$

When an operation is applied to sequences X, Y, ... that have different lengths, the operation is applied element-wise to the first μ elements of the sequences, where $\mu = \min(|X|, |Y|, \ldots)$. For example,

$$(\{1, 2, 3, 4\} + \{1, 4\}) = \{2, 6\}$$

and

$$(\{\mathbf{a}, \mathbf{ab}, \mathbf{c}\}\ \|\ \{\mathbf{x}\}) = \{\mathbf{ax}\}$$

In general, for a function $f(x, y, \ldots)$,

$$f(X, Y, \ldots) \equiv [I \uparrow \mu : f(X!i, Y!i, \ldots)]$$

where $\mu = \min(|X|, |Y|, \ldots)$.

3.2 Selection

The conditional selection of a sequence is denoted by

$$\textit{if } e \textit{ then } X \textit{ else } Y$$

where e is a scalar conditional. An example is

$$\textit{if } |X| > 10 \textit{ then } X \textit{ else } \Phi$$

which selects X if its length is greater than 10 but selects Φ otherwise.

An omitted *else* clause is an abbreviation for

$$else\ \Phi$$

so that the generator above can be written as

$$if\ \ |X| > 10\ then\ X$$

As in other contexts where sequences are expected, a scalar value can be used as an abbreviation for its corresponding unit sequence. Thus

$$[if\ \ I!i = 0\ then\ \{0\}\ else\ \{1\}]$$

can be written as

$$[if\ \ I!i = 0\ then\ 0\ else\ 1]$$

3.3 Reduction

Reduction over a dyadic operation $\circ$ is denoted by $Red_{\circ}(X)$ and produces the result of applying $\circ$ to the elements of X. $Red_{\circ}(X)$ is undefined if $|X| = 0$, while $Red_{\circ}(X) = X!1$ if $|X| = 1$. For example,

$$Red_{+}(\mathcal{I}_{+} \uparrow 5) = 1+2+3+4+5 = 15$$

$$[Red_{+}(\mathcal{F} \uparrow i)] = \{1, 2, 4, 7, 12, \ldots\}$$

$$[Red_{\|}(\mathfrak{ABC} \uparrow (i+1))] = \{\mathbf{a}, \mathbf{ab}, \mathbf{abc}, \mathbf{abcaa}, \mathbf{abcaaab}, \ldots\}$$

4. Generator Paradigms

Generators can be used in various ways to describe operations on sequences. Some useful paradigms follow.

Filters:

A sequence can be *filtered* to select elements that satisfy some condition. For example,

$$[if\ \ I!i\ mod\ 2 = 0\ then\ I!i]$$

consists of the even-valued elements of I. Similarly,

$$[if\ \ |\mathfrak{ABC}!i|\ mod\ 4 = 0\ then\ \mathfrak{ABC}!i]$$

consists of the elements of $\mathfrak{ABC}$ whose lengths are an even multiple of 4.

Transposition:

A reordering of the elements of a sequence is called a *transposition* of the sequence. For example,

$$[if\ \ i\ mod\ 2 = 1\ then\ X!(i+1)\ else\ X!(i-1)]$$

exchanges consecutive elements of X.

Combination:

The elements of a sequence can be combined in various ways to produce new sequences. For example, if

$$I = \{i_1, i_2, i_3, \ldots\}$$

then a new sequence can be formed by adding successive elements of I:

$$\{i_1 + i_2, i_3 + i_4, i_5 + i_6, \ldots\}$$

and is given by

$$[if\ \ i\ mod\ 2 = 1\ then\ I!i + I!(i+1)]$$

Indexing Sequences:

An *indexing sequence* consists of the indices of a sequence for which some condition is satisfied. For example, the indexing sequence of the odd-valued elements of $\mathcal{F}$ is

$$\{1, 2, 4, 5, 7, \ldots\}$$

Similarly,

$$[\, if \;\; I\,!i = 0 \; then \;\; i\,]$$

consists of the indices of the zero-valued elements of I.

5. Recurrence Relations

Many sequences, such as $\mathcal{F}$, are defined conveniently by recurrence relations:

$$fib\,(1) = 1$$

$$fib\,(2) = 1$$

$$fib\,(i) = fib\,(i-1) + fib\,(i-2)\,,\; i = 3, 4, \ldots$$

so that

$$\mathcal{F} = [fib\,(i)]$$

Recurrence relations are not limited to integers. The "Fibonacci strings" [7] are given by

$$fibs\,(1) = \mathbf{a}$$

$$fibs\,(2) = \mathbf{b}$$

$$fibs\,(i) = fibs\,(i-1) \;||\; fibs\,(i-2)\,,\; i = 3, 4, \ldots$$

so that

$$[fibs\,(i)] = \{\mathbf{a}, \mathbf{b}, \mathbf{ba}, \mathbf{bab}, \mathbf{babba}, \ldots\}$$

Recurrences may be nested. An example from Hofstadter [8] is:

$$g\,(0) = 0$$

$$g\,(i) = i - g\,(g\,(i-1))\,,\; i = 1, 2, 3, \ldots$$

so that

$$[g\,(i)] = \{1, 1, 2, 3, 3, 4, 4, 5, 6, \ldots\}$$

Note that $g\,(0)$ does not appear in this sequence, since the first value in $[g\,(i)]$ is $g\,(1)$. The complete sequence for this recurrence relation is given by $[g\,(i-1)]$ or $[\mathcal{G}_0 : g\,(i)]$.

While there is a well-known closed form for the Fibonacci sequence, and closed forms can be determined for many other sequences given by recurrence relations (see [9] for a closed form for $g\,(i)$), some recurrence relations do not have known closed forms. An example from [8] is

$$q\,(1) = 1$$

$$q\,(2) = 1$$

$$q\,(i) = q\,(i - q\,(i-1)) + q\,(i - q\,(i-2))\,,\; i = 3, 4, \ldots$$

Recurrence relations may have parameters in addition to the recurrence variable. An example from [9] is:

$$gk\,(i, k) = 0\,,\; i < 1\,;\; k > 0$$

$$gk\,(i, k) = i - gk\,(gk\,(i-k, k), k)\,,\; i = 1, 2, 3, \ldots\,;\; k > 0$$

Thus

$$[\mathcal{G}_0 : gk\,(i\,,\,2)] = \{0,\ 1,\ 2,\ 2,\ 2,\ 3,\ 4,\ 5,\ 6,\ 6,\ 6,\ \ldots\}$$

Note that $gk\,(i\,,\,k\,)$ is defined to be a constant for all values of i less than 1.

In order to use the generator notation to describe recursively defined sequences, a name may be associated with a sequence using *define*, as in

$$(\mathit{define}\ \ G\ \ [\mathcal{G}_0 :\ \mathit{if}\ \ i = 0\ \mathit{then}\ \ 0\ \ \mathit{else}\ \ i - G\,!(G\,!(i-1))])$$

6. Discussion

The approach to describing and manipulating sequences given in this report allows

(1) meaningful computations on infinite sequences

(2) application of functions to sequences of different lengths

(3) the description of a variety of evaluation mechanisms, including "dot product" and "cross product"

(4) a view of sequences as being produced in time or existing in storage

(5) sequences as elements of sequences

The ability to manipulate infinite sequences is important. While lazy evaluation techniques [10] provide the potential for sequences of unbounded length, the semantics in most programming languages are still storage oriented.

The ability to apply functions to sequences of different lengths eliminates a common restriction on vector processing in a natural and useful way.

The way that functions are applied to sequences is typically fixed in programming languages. Examples are parallel operations in vector processing and goal-directed evaluation in pattern matching. The ability to specify how functions are applied removes these built-in constraints.

In many contexts it is natural to think of the elements of sequences as being produced as a process takes place. This concept is embodied in generators.

Generators produce values in the order of the elements in the generation sequence. Termination of generation is a significant problem. Part of the problem can be handled through the following interpretation of undefined operations: If an undefined operation (such as $X\,!i\,)$ occurs in the generation expression, the sequence produced by the generator terminates at that point. For example, if

$$S = \{\mathbf{aa},\ \mathbf{ab},\ \mathbf{ba},\ \mathbf{bb}\}$$

then

$$[S\,!(i+1)]$$

produces {**ab**, **ba**, **bb**}. Similarly,

$$[\{1,\ 3,\ 1,\ 5,\ 1,\ 7\} : S\,!i\,]$$

produces {**aa**, **ba**, **aa**}. Note that generation terminates because $S\,!5$ is undefined, even though the subsequent element, $S\,!1$, is defined.

The general problem of termination is another matter. Although $[\Phi] = \Phi$, there is no way, in general, to stop the generation process. Consider

$$[\,\mathit{if}\ \ I\,!i = 0\ \mathit{then}\ \ i\,]$$

Determining whether this generator terminates for an arbitrary integer sequence I is equivalent to the halting problem. Pragmatically, generation can be bounded by using finite generation sequences, as in

$$[\mathcal{G}_+ \uparrow 1000 :\ \mathit{if}\ \ I\,!i = 0\ \mathit{then}\ \ i\,]$$

or

$$([\,\mathit{if}\ \ I\,!i = 0\ \mathit{then}\ \ i\,]) \uparrow 1000$$

but this may prove cumbersome and unsatisfactory in practice.

More computational mechanisms are needed in order to use the ideas presented here in a programming context. Since the nature of these mechanisms might take many different forms, they are not discussed here. One possible context for computing with sequences is described in a companion report [6].

Acknowledgement

Dave Hanson and Tom Hicks provided a number of helpful suggestions on the presentation of the material in this report. Pete Downey contributed a number of important ideas and has suggested areas for further work. Special thanks go to Steve Wampler for reading numerous drafts of this report and for making important contributions to the concept of generators.

References

1. Griswold, Ralph E. and Madge T. Griswold. *The Icon Programming Language*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1983.

2. Wampler, Stephen B. *Control Mechanisms for Generators in Icon*, Ph.D. Dissertation, Department of Computer Science, The University of Arizona. 1981.

3. Wampler, Stephen B. and Ralph E. Griswold. "Result Sequences", *Journal of Computer Languages*, Vol. 8, No. 1 (1983), pp. 1-14.

4. Burge, William H. *ISWIM Programming Manual.* IBM Computer Science Research Report RA 129. 1981.

5. Ghezzi, Carlo and Mehdi Jazayeri. *Programming Language Concepts.* John Wiley & Sons, Inc., New York. 1982. pp. 227-254.

6. Griswold, Ralph E. *Seque: An Experimental Language for Manipulating Sequences.* Technical report TR 83-16, Department of Computer Science, The University of Arizona. November 1983.

7. Knuth, Donald E. *The Art of Computer Programming*, Vol. 1. Addison-Wesley Publishing Company, Reading, Massachusetts. 1968. p. 85.

8. Hofstadter, Douglas R. *Gödel, Escher, Bach: An Eternal Golden Braid.* Basic Books, New York. 1979. pp. 137-138.

9. Downey, Peter J. and Ralph E. Griswold. *On a Family of Nested Recurrences*, Technical Report TR 82-18, Department of Computer Science, The University of Arizona. 1982.

10. Friedman, Daniel P. and David S. Wise. "CONS Should Not Evaluate its Arguments", in *Automata, Languages, and Programming*, S. Michaelson and R. Miker, eds., Edinburgh University Press. 1976. pp. 257-284.