J. Functional Programming 2 (4):505–513, October 1992

FUNCTIONAL PEARLS

A symmetric set of efficient list operations

ROB R. HOOGERWOORD

Department of Mathematics and Computing Science, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands

0 Introduction

In this paper we show that it is possible to implement a symmetric set of finite-list operations efficiently; the set is symmetric in the sense that lists can be manipulated at either end. We derive the definitions of these operations from their specifications by calculation. The operations have O(1) time complexity, provided that we content ourselves with, so-called, *amortized efficiency*, instead of worst-case efficiency.

The idea behind our design is simple and not new (Gries, 1981), but to be effective its elaboration requires some care. The idea is to represent each list by a *pair of lists*: a pair $[x, y]$ is used to represent the list $x +\!\!+ rev \cdot y$. (Here, a dot denotes functional application.) Thus, each list can be represented in many ways, and it is by judicious exploitation of this freedom that we achieve our goal. We elaborate this in section 3.

1 Amortized complexity

Without pretending generality, we introduce the notion of *amortized complexity* in a form suiting our purpose.

In this section V is a set and f and t are functions of types $V \to V$ and $V \to Nat$, respectively. For v, $v \in V$, we interpret $t \cdot v$ as the *cost*, in some meaning of the word, of evaluation of $f \cdot v$. Now suppose that we are interested in a sequence of successive applications of f; i.e. we define a sequence x as follows:

$$x_0 \in V \ (x_0 \text{ is assumed to be known}), \text{ and}$$
$$x_{i+1} = f \cdot x_i, \quad 0 \leqslant i.$$

Computation of the first $n+1$ elements of x then costs $(\Sigma i : 0 \leqslant i < n : t \cdot x_i)$. If the value of this expression, as a function of n, is $O(n)$, then we say that the *amortized cost* of each of f's applications (in sequence x) is O(1). Of course, this is so if t is O(1), but this is not necessary: the requirement that $(\Sigma i : 0 \leqslant i < n : t \cdot x_i)$ is $O(n)$ is weaker. The introduction of amortized cost reflects our decision to be interested only in the cumulative cost of a sequence of successive operations.

For the sake of simplicity, it would be nice if we could discuss the amortized cost of f without introduction of sequence x. This can be done as follows. We introduce

a function s, of type $V \to Nat$, and we interpret $s \cdot v$ as the amortized cost of evaluation of $f \cdot v$. We try to couple s and t in such a way that, for our sequence x, we have:

$$(\Sigma i: 0 \leqslant i < n: t \cdot x_i) \leqslant (\Sigma i: 0 \leqslant i < n: s \cdot x_i), \quad \text{for all } n: 0 \leqslant n. \qquad (*)$$

Consequently, if s is O(1) then the cumulative cost of computing the first $n+1$ elements of x is indeed O(n).

The following idea for a suitable coupling is – as far as we know – due to Tarjan (1985). We design, or invent, a function c, of type $V \to Nat$, and define s as follows:

$$s \cdot v = t \cdot v + c \cdot (f \cdot v) - c \cdot v, \quad \text{for all } v, v \in V.$$

Under the additional assumption $c \cdot x_0 = 0$, this s satisfies $(*)$.

What does this mean in practice? To prove that a function f, with given cost function t, has amortized cost O(1), it suffices to design a natural function c, the so-called *credit function*, satisfying:

$$c \cdot x_0 = 0, \quad \text{and}$$
$$t \cdot v + c \cdot (f \cdot v) - c \cdot v \quad \text{is, as function of } v, \text{ O(1)}.$$

Here x_0 represents the initial argument – or the initial state – of the computation.

The above remains valid when f represents the elements of a whole *class of functions*, each having its own cost function t. In this case, one and the same credit function must satisfy the above requirement for each pair f, t from this class.

2 Specifications

The problem to be solved is to implement an extended set of elementary list operations in such a way that the amortized time complexity of each of these operations is O(1). For this purpose, the set L_* of lists will be represented by a set V, say, such that the representation of lists by elements of V is not unique. The abstraction function mapping V to L_* is denoted by $[\![\cdot]\!]$, i.e. $[\![s]\!]$ is the list represented by s, for s, $s \in V$.

We use lists and their associated functions for two purposes, namely to *specify* the new list operations and to *implement* them. The functions to be implemented are:

$$\vdash (\text{'}\textit{left cons}\text{'}) \quad \text{and} \quad \dashv (\text{'}\textit{right cons}\text{'})$$
$$\textit{lhd}(\text{'}\textit{left head}\text{'}) \quad \text{and} \quad \textit{rhd}(\text{'}\textit{right head}\text{'})$$
$$\textit{ltl}(\text{'}\textit{left tail}\text{'}) \quad \text{and} \quad \textit{rtl}(\text{'}\textit{right tail}\text{'}).$$

Using $[\![\cdot]\!]$ we specify these functions as follows; for any a and for s, $s \in V$:

$$\begin{aligned}
[\![a \vdash s]\!] &= [a] +\!\!+ [\![s]\!] \\
[\![s \dashv a]\!] &= [\![s]\!] +\!\!+ [a] \\
\textit{lhd} \cdot s &= hd \cdot [\![s]\!], \quad [\![s]\!] \neq [\,] \\
\textit{rhd} \cdot s &= hd \cdot (rev \cdot [\![s]\!]), \quad [\![s]\!] \neq [\,] \\
[\![\textit{ltl} \cdot s]\!] &= tl \cdot [\![s]\!], \quad [\![s]\!] \neq [\,] \\
[\![\textit{rtl} \cdot s]\!] &= rev \cdot (tl \cdot (rev \cdot [\![s]\!])), \quad [\![s]\!] \neq [\,].
\end{aligned}$$

https://doi.org/10.1017/S0956796800000526 Published online by Cambridge University Press

Remark. From these specifications, the types of these functions can be derived easily. □

We also need a representation of the empty list, i.e. we must choose a value $[\,]_V$, $[\,]_V \in V$, satisfying:

$$[\![\,[\,]_V]\!] = [\,].$$

Functions $(a \vdash)$ and $(\dashv a)$, for every a, and functions *ltl* and *rtl* have type $V \to V$. They will be implemented in such a way that their amortized time complexity is O(1). Functions *lhd* and *rhd* do not fit into this pattern: they are functions from V to elements. This is no problem: we shall see to it that *lhd* and *rhd* have O(1) (worst-case) time complexity.

3 Representation

Our new lists are represented by pairs of old lists, i.e. we choose $V = L_* \times L_*$. For function $[\![\cdot]\!]$ we choose:

$$[\![\,[x, y]]\!] = x \quad rev \cdot y.$$

This representation leaves us no choice for the definition of $[\,]_V$: the only solution of the equation x, y: $[\,] = x +\!\!+ rev \cdot y$ is $[\,]$, $[\,]$; hence:

$$[\,]_V = [[\,], [\,]].$$

We now derive a definition for *lhd*:

$$\begin{aligned}
& lhd \cdot [x, y] \\
= \quad & \{\text{specification of } lhd\} \\
& hd \cdot [\![\,[x, y]]\!] \\
= \quad & \{\text{definition of } [\![\cdot]\!]\} \\
& hd \cdot (x +\!\!+ rev \cdot y) \\
= \quad & \{\text{definitions of } \quad \text{and } hd\} \\
& \textbf{if } x \neq [\,] \to hd \cdot x \\
& \;[\!]\; x = [\,] \to hd \cdot (rev \cdot y) \\
& \textbf{fi}.
\end{aligned}$$

Evaluation of $hd \cdot (rev \cdot y)$ takes $O(\#y)$ time; hence, the definition thus obtained does not have O(1) time complexity. It does, however, have O(1) time complexity in the special case $x \neq [\,] \vee \#y \leqslant 1$, or equivalently, $1 \leqslant \#x \vee \#y \leqslant 1$. Therefore, we restrict set V to the pairs $[x, y]$ that satisfy $1 \leqslant \#x \vee \#y \leqslant 1$. For y, $\#y \leqslant 1$, we have $rev \cdot y = y$; thus, we obtain the following definition for *lhd*:

$$\begin{aligned}
lhd \cdot [x, y] = \; & \textbf{if } x \neq [\,] \to hd \cdot x \\
& \;[\!]\; x = [\,] \to hd \cdot y \\
& \textbf{fi}.
\end{aligned}$$

By symmetry, we restrict set V to those pairs $[x, y]$ satisfying $1 \leqslant \#y \vee \#x \leqslant 1$ as

https://doi.org/10.1017/S0956796800000526 Published online by Cambridge University Press

well. Together, these two restrictions define set V as a subset of $L_* \times L_*$. The relation defining this subset, also called the *representation invariant*, is Q, with:

$$Q: \quad (1 \leqslant \#x \vee \#y \leqslant 1) \wedge (1 \leqslant \#y \vee \#x \leqslant 1).$$

The definition for *rhd* then becomes (notice the symmetry):

$$\begin{aligned} rhd\cdot[y,x] = \ & \textbf{if}\ x \neq [] \rightarrow hd\cdot x \\ & [\!]\ x = [] \rightarrow hd\cdot y \\ & \textbf{fi}. \end{aligned}$$

4. Left and right cons

The derivation of definitions for $\vdash$ and $\dashv$ is straightforward if we temporarily forget the proof obligation with respect to Q. We perform these derivations in parallel:

$$\begin{array}{ll|ll} & [\![a \vdash [x,y]]\!] & & [\![[y,x] \dashv a]\!] \\ = & \{\text{specification of } \vdash\} & = & \{\text{specification of } \dashv\} \\ & [a] +\!\!+ [\![[x,y]]\!] & & [\![[y,x]]\!] +\!\!+ [a] \\ = & \{\text{definition of } [\![\cdot]\!]\} & = & \{\text{definition of } [\![\cdot]\!]\} \\ & [a] +\!\!+ x +\!\!+ rev\cdot y & & y +\!\!+ rev\cdot x +\!\!+ [a] \\ = & \{\text{list calculus}\} & = & \{\text{list calculus}\} \\ & (a:x) +\!\!+ rev\cdot y & & y +\!\!+ rev\cdot(a:x) \\ = & \{\text{definition of } [\![\cdot]\!]\} & = & \{\text{definition of } [\![\cdot]\!]\} \\ & [\![[a:x,y]]\!]. & & [\![[y,a:x]]\!]. \end{array}$$

Thus, we conclude that the specifications of $\vdash$ and $\dashv$ are satisfied by:

$$a \vdash [x,y] = [a:x,y].$$
$$[y,x] \vdash a = [y,a:x].$$

The expression $[a:x,y]$ will only satisfy Q if $1 \leqslant \#y$. For the special case $y = []$, we redo the above derivation:

$$\begin{aligned} & [\![a \vdash [x,[]]]\!] \\ = & \quad \{\text{as before, with } [] \text{ for } y\} \\ & [a] +\!\!+ x +\!\!+ rev\cdot[] \\ = & \quad \{rev\cdot[] = [],\ [] \text{ is the identity of } +\!\!+\} \\ & [a] +\!\!+ x \\ = & \quad \{[x,[]] \text{ satisfies } Q, \text{ hence } \#x \leqslant 1, \text{ hence } x = rev\cdot x\} \\ & [a] +\!\!+ rev\cdot x \\ = & \quad \{\text{definition of } [\![\cdot]\!]\} \\ & [\![[[a],x]]\!]. \end{aligned}$$

https://doi.org/10.1017/S0956796800000526 Published online by Cambridge University Press

Expression $[[a], x]$ satisfies Q. Thus, we obtain the following definition for $\vdash$ and, similarly, for $\dashv$:

$$\begin{aligned} a \vdash [x, y] = \ & \textbf{if } y \neq [] \rightarrow [a : x, y] \\ & [\!] \ y = [] \rightarrow [[a], x] \\ & \textbf{fi}, \end{aligned}$$

$$\begin{aligned} [y, x] \dashv a = \ & \textbf{if } y \neq [] \rightarrow [y, a : x] \\ & [\!] \ y = [] \rightarrow [x, [a]] \\ & \textbf{fi}. \end{aligned}$$

The (normal) time complexity of these definitions is O(1). In order that their amortized time complexity is O(1) as well, the credit function c must be chosen such that its value increases by a bounded amount under these operations; that is, $c \cdot (a \vdash [x, y]) - c \cdot [x, y]$ must be bounded from above by a constant.

5 Left and right tail

We now derive definitions for *ltl* and *rtl*. These derivations do not yield efficient definitions, but they do provide information on how the credit function can be chosen such that these definitions have O(1) amortized time complexity.

For $ltl \cdot [x, y]$, with precondition $[\![[x, y]]\!] \neq []$, we derive:

$$\begin{aligned} & [\![ltl \cdot [x, y]]\!] \\ = \ & \{\text{specification of } ltl\} \\ & tl \cdot [\![[x, y]]\!] \\ = \ & \{\text{definition of } [\![\cdot]\!]\} \\ & tl \cdot (x +\!\!+ rev \cdot y). \end{aligned}$$

Further manipulation of this formula requires distinction of the cases $x \neq []$ and $x = []$.

For the case $x = []$ we have:

$$\begin{aligned} & tl \cdot (x +\!\!+ rev \cdot y) \\ = \ & \{x = []\} \\ & tl \cdot (rev \cdot y) \\ = \ & \{Q \wedge [\![[x, y]]\!] \neq [] \wedge x = [] \Rightarrow \#y = 1, \text{hence } tl \cdot (rev \cdot y) = []\} \\ & [] \\ = \ & \{\text{specification of } []_v\} \\ & [\![[]_v]\!] \\ = \ & \{\text{definition of } []_v\} \\ & [\![[[], []]]\!]. \end{aligned}$$

Hence, for the case $x = []$ we must choose – no freedom – $ltl \cdot [x, y] = [[], []]$.

https://doi.org/10.1017/S0956796800000526 Published online by Cambridge University Press

For the case $x \neq [\,]$ we derive:

$$\begin{array}{ll} & tl\cdot(x \quad rev\cdot y) \\ = & \quad \{x \neq [\,], \text{definitions of } +\!\!+ \text{ and } tl\} \\ & tl\cdot x +\!\!+ rev\cdot y \\ = & \quad \{\text{definition of } [\![\cdot]\!]\} \\ & [\![[tl\cdot x, y]]\!]. \end{array}$$

So, for the case $x \neq [\,]$, we may choose $ltl\cdot[x,y] = [tl\cdot x, y]$, provided that this expression satisfies Q, i.e. we must prove $Q \Rightarrow Q(x, y \leftarrow tl\cdot x, y)$, where $\leftarrow$ denotes *substitution*. Assuming Q we derive:

$$\begin{array}{ll} & Q(x, y \leftarrow tl\cdot x, y) \\ = & \quad \{\text{definition of } Q\} \\ & (1 \leqslant \#(tl\cdot x) \vee \#y \leqslant 1) \wedge (1 \leqslant \#y \vee \#(tl\cdot x) \leqslant 1) \\ = & \quad \{\text{definitions of } tl \text{ and } \#\} \\ & (2 \leqslant \#x \vee \#y \leqslant 1) \wedge (1 \leqslant \#y \vee \#x \leqslant 2) \\ = & \quad \{Q \Rightarrow 1 \leqslant \#y \vee \#x \leqslant 2\} \\ & 2 \leqslant \#x \vee \#y \leqslant 1 \\ \Leftarrow & \quad \{\text{predicate calculus}\} \\ & 2 \leqslant \#x. \end{array}$$

So, for the special case that x has at least two elements, the above definition for ltl is correct. The remaining case is that x is a singleton list:

$$\begin{array}{ll} & tl\cdot x +\!\!+ rev\cdot y \\ = & \quad \{\#x = 1\} \\ & [\,] +\!\!+ rev\cdot y \\ = & \quad \{[\,] \text{ is the identity of } +\!\!+\} \\ & rev\cdot y \\ = & \quad \{\text{introduce } u \text{ and } v \text{ such that } y = u +\!\!+ v \quad \text{(note 1, see below)}\} \\ & rev\cdot(u +\!\!+ v) \\ = & \quad \{\text{list calculus}\} \\ & rev\cdot v +\!\!+ rev\cdot u \\ = & \quad \{\text{definition of } [\![\cdot]\!]\} \\ & [\![[rev\cdot v, u]]\!]. \end{array}$$

So, for the case $\#x = 1$ we may choose $ltl\cdot[x,y] = [rev\cdot v, u]$, where $u +\!\!+ v = y$.

Note 1: The decision to split y into parts u and v is inspired by the desire to transform $rev\cdot y$ into a pair of values. By not further specifying u and v we retain the freedom to choose the most efficient representation.

□

https://doi.org/10.1017/S0956796800000526 Published online by Cambridge University Press

Evaluation of $[rev \cdot v, u]$ takes $O(\#y)$ time, independently of how u and v have been chosen. To obtain O(1) amortized time complexity, the credit function c must satisfy:

$$t \cdot y + c \cdot [rev \cdot v, u] - c \cdot [x, y] \leqslant K,$$

where $t \cdot y$ denotes the time needed to evaluate $[rev \cdot v, u]$ and for some constant K. Because $t \cdot y$ is $O(\#y)$ we may safely assume that $t \cdot y \leqslant \#y + 1$; then we have:

$$\begin{array}{ll} & t \cdot y + c \cdot [rev \cdot v, u] - c \cdot [x, y] \leqslant K \\ \equiv & \{\text{calculus}\} \\ & c \cdot [x, y] - c \cdot [rev \cdot v, u] \geqslant t \cdot y - K \\ \Leftarrow & \{t \cdot y \leqslant \#y + 1, \text{transitivity of} \leqslant\} \\ & c \cdot [x, y] - c \cdot [rev \cdot v, u] \geqslant \#y + 1 - K. \end{array}$$

6 The credit function

So as not to destroy the symmetry we require c to be symmetric in x and y; so, $c \cdot [x, y] = c \cdot [y, x]$, for all x and y. One of the simplest such functions is given by:

$$c \cdot [x, y] = \#x + \#y.$$

By a simple calculation it can be shown that this definition is equivalent to:

$$c \cdot [x, y] = \# [\![[x, y]]\!].$$

This function is not useful, for two reasons. First, the length of the represented list increases or decreases by just 1 under each of the list operations. So, amortized and normal complexity coincide. Phrased differentially, the idea of amortized complexity amounts to choosing a function c that allows, every now and then, more substantial decreases of its value. Second, the second definition shows that c is invariant under changes of representation; so, this c gives no heuristic guidance when we exploit the freedom to apply changes of representation.

A function c that does satisfy these requirements is:

$$c \cdot [x, y] = |\#x - \#y|.$$

We leave the proof that the value of c increases by at most 1 under the left and right cons operations as an exercise to the reader.

We now use this c to complete the design of the definitions for *ltl* and *rtl*. In the previous section we have derived that, for the special case $\#x = 1$, values u and v must be chosen such that $c \cdot [x, y] - c \cdot [rev \cdot v, u] \geqslant \#y + 1 - K$, for some constant K. We have:

$$\begin{array}{ll} & c \cdot [x, y] \\ = & \{\text{definition of } c\} \\ & |\#x - \#y| \\ = & \{\#x = 1\} \\ & |\#y - 1| \\ \geqslant & \{\text{definition of } |\cdot|\} \\ & \#y - 1. \end{array}$$

https://doi.org/10.1017/S0956796800000526 Published online by Cambridge University Press

Furthermore, we must see to it that $c\cdot[rev\cdot v, u]$ is not too large:

$$\begin{aligned} & c\cdot[rev\cdot v, u] \\ = & \quad \{\text{definition of } c\} \\ & |\#(rev\cdot v) - \#u| \\ = & \quad \{\#(rev\cdot v) = \#v\} \\ & |\#v - \#u|. \end{aligned}$$

By choosing the lengths of u and v as equal as possible we achieve $c\cdot[rev\cdot v, u] \leqslant 1$. So, we have:

$$c\cdot[x, y] - c\cdot[rev\cdot v, u] \geqslant \#y - 2.$$

Therefore, we choose u and v such that:

$$u +\!\!+ v = y \wedge \#u \leqslant \#v \leqslant \#u + 1.$$

The pair $[rev\cdot v, u]$ thus specified satisfies Q; this follows from a simple calculation. From this specification it also follows that $\#u = \#y \,\mathbf{div}\, 2$. For any k, $0 \leqslant k \leqslant \#y$, the equation $u, v : u +\!\!+ v = y \wedge \#u = k$ has exactly one solution which we denote by $y\uparrow k$, $y\downarrow k$. This solution can be computed, in $O(k)$ time. Therefore, we define u and v by $u = y\uparrow k$ and $v = y\downarrow k$, with $k = \#y \,\mathbf{div}\, 2$.

Putting all the pieces together we obtain the following definitions for *ltl* and its symmetric counterpart *rtl*:

$$\begin{aligned} ltl\cdot[x, y] = \ & \mathbf{if}\ \#x = 0 \to [[\,],[\,]] \\ & \ \mathbb{[]}\ \#x = 1 \to [rev\cdot(y\downarrow k), y\uparrow k] \quad \mathbf{where}\ k = \#y \,\mathbf{div}\, 2\ \mathbf{end} \\ & \ \mathbb{[]}\ \#x \geqslant 2 \to [tl\cdot x, y] \\ & \mathbf{fi}. \end{aligned}$$

$$\begin{aligned} rtl\cdot[y, x] = \ & \mathbf{if}\ \#x = 0 \to [[\,],[\,]] \\ & \ \mathbb{[]}\ \#x = 1 \to [y\uparrow k, rev\cdot(y\downarrow k)] \quad \mathbf{where}\ k = \#y \,\mathbf{div}\, 2\ \mathbf{end} \\ & \ \mathbb{[]}\ \#x \geqslant 2 \to [y, tl\cdot x] \\ & \mathbf{fi}. \end{aligned}$$

8 Epilogue

A formalized notion of amortized complexity turns out to be of heuristic value for the derivation of efficient programs. In our example, we have chosen function c with no more justification than an appeal to a few general criteria. Once c has been chosen, the definitions for *ltl* and *rtl* can be completed in a rather straightforward way.

In terms of the list representation used in this paper, reversal of a list is a trivial operation: we have $rev\cdot[\![[x, y]]\!] = [\![[y, x]]\!]$, for all x and y. Hence, *rev* can be implemented efficiently by a function *Rev*, defined by $Rev\cdot[x, y] = [y, x]$. Thus, list reversal becomes an $O(1)$ operation.

https://doi.org/10.1017/S0956796800000526 Published online by Cambridge University Press

Acknowledgements

To Berry Schoenmakers for drawing my attention to the above definition of *Rev*, and to David Gries for his comments on an earlier version of this paper.

References

Gries, D. 1981. *The Science of Programming*. Springer-Verlag.

Tarjan, R. E. 1985. *Amortized computational complexity. SIAM J. on Algebraic and Discrete Methods*, **6**: 306–318.