A symmetric set of efficient list operations

Rob R. Hoogerwoord
department of mathematics and computing science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven
The Netherlands

abstract

The classical set of list operations, viz. "cons", "head", and "tail", can be implemented efficiently, but it is asymmetric. We derive an implementation of a symmetric set of list operations, in terms of the classical ones. The new operations are efficient too, but only in the amortized sense. We also give a short explanation of amortized complexity. The main purpose of this paper, however, is to illustrate a style of programming by calculation.

keywords: program derivation by calculation, functional programming, list operations, amortized complexity

0 Introduction

In most functional-program notations the only elementary list operations are ; ("cons"), hd ("head") and tl ("tail"). The reason for this is probably historical (LISP). Operationally speaking, by means of these operations lists can be manipulated at their "left ends" only. This restricted choice of operations allows an efficient implementation: they have O(1) time complexity and they allow sharing of storage space.

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. In this respect, this paper also is an exercise in functional programming. 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 [0], but, in order 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 $x \mathbin{+\!\!+} \text{rev}{\cdot}y$. 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 Notational prelude

The set of (finite) lists is L_* . It consists of the *empty list*, which is denoted by $[]$, and the *composite lists*, which are of the form $a;x$, for element a and list x . Throughout this paper, a denotes a value of the (anonymous) element type, whereas x,y,z denote lists. The *singleton list* $a;[]$ is also denoted by $[a]$. For composite list x , the *head* and the *tail* of x are denoted by $\text{hd}{\cdot}x$ and $\text{tl}{\cdot}x$; i.e., we have:

$$\text{hd}{\cdot}(a;x) = a \ \wedge\ \text{tl}{\cdot}(a;x) = x \quad .$$

Catenation of lists is denoted by the infix operator $\mathbin{+\!\!+}$; it is defined recursively by:

$$[] \mathbin{+\!\!+} y = y \ \wedge\ (a;x) \mathbin{+\!\!+} y = a;(x \mathbin{+\!\!+} y) \quad .$$

Catenation has nice algebraic properties: it has $[]$ as a left and right identity element, and it is associative. The *reverse* of list x is denoted by $\text{rev}{\cdot}x$; it has the following algebraic properties:

$$\text{rev}{\cdot}[] = [] \ \wedge\ \text{rev}{\cdot}[a] = [a] \ \wedge\ \text{rev}{\cdot}(x \mathbin{+\!\!+} y) = \text{rev}{\cdot}y \mathbin{+\!\!+} \text{rev}{\cdot}x \quad .$$

Finally, the *length* of list x is denoted by $\#x$.

Many relations between these operators can be derived, such as $\text{hd}{\cdot}([] \mathbin{+\!\!+} y) = \text{hd}{\cdot}y$, and, for composite x , $\text{hd}{\cdot}(x \mathbin{+\!\!+} y) = \text{hd}{\cdot}x$. In this paper we freely use such properties with only vague justifications such as "list calculus" or "definition of $\mathbin{+\!\!+}$ ".

The time complexities of $;$, hd , and tl are assumed to be $O(1)$; the time complexities of $x \mathbin{+\!\!+} y$, $\text{rev}{\cdot}x$, and $\#x$ are assumed to be $O(\#x)$.

2 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 $V \rightarrow V$ and $V \rightarrow Nat$ 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:

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

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 $V \rightarrow Nat$, 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:

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

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

The following idea for a suitable coupling is -- as far as we know -- due to R.E. Tarjan [1]. We design, or invent, a function c , of type $V \rightarrow Nat$, and define s as follows:

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

Under the additional assumption $c \cdot x_0 = 0$, we derive:

$(\Sigma\, i : 0 \leqslant i < n : t \cdot x_i)$

= { "telescope summation" }

$(\Sigma\, i : 0 \leqslant i < n : t \cdot x_i + c \cdot x_{i+1} - c \cdot x_i) \; - \; c \cdot x_n \; + \; c \cdot x_0$

= { definition of x_{i+1} }

$(\Sigma\, i : 0 \leqslant i < n : t \cdot x_i + c \cdot (f \cdot x_i) - c \cdot x_i) \; - \; c \cdot x_n \; + \; c \cdot x_0$

= { definition of s }

$(\Sigma\, i : 0 \leqslant i < n : s \cdot x_i) \; - \; c \cdot x_n \; + \; c \cdot x_0$

$\leqslant$ { $0 \leqslant c \cdot x_n$, $c \cdot x_0 = 0$ }

$(\Sigma\, i : 0 \leqslant i < n : s \cdot x_i)$.

What does this mean in practice? In order 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:

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

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.

3 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, L_* will be represented by a set V , say, such that the representation of lists from L_* 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 L_* and its associated functions for two purposes, namely to *specify* the new list operations and to *implement* them. The functions to be implemented are:

$\vdash$	("left cons")	and	$\dashv$ ("right cons")
lhd	("left head")	and	rhd ("right head")
ltl	("left tail")	and	rtl ("right tail") .

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

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

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

Moreover, we 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→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.

4 Representation

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

$$[\![[x,y]]\!] = x +\!\!+ \mathrm{rev}\cdot y \ .$$

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 :

$$
\begin{array}{ll}
 & lhd{\cdot}[x,y] \\
= & \quad \{ \text{ specification of lhd } \} \\
 & hd{\cdot}[\![\, [x,y] \,]\!] \\
= & \quad \{ \text{ definition of } [\![\cdot]\!] \ \} \\
 & hd{\cdot}(x +\!\!+ rev{\cdot}y) \\
= & \quad \{ \text{ definition of } +\!\!+ \quad \} \\
 & \textbf{if}\ x \neq [] \rightarrow hd{\cdot}x \\
 & [\!]\ x = [] \rightarrow hd{\cdot}(rev{\cdot}y) \\
 & \textbf{fi}\ \ .
\end{array}
$$

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 $x{\neq}[] \vee y{=}[]$. We could, therefore, restrict set V to the pairs $[x,y]$ that satisfy $x{\neq}[] \vee y{=}[]$. The conjunction of this restriction and its symmetric counterpart for rhd , $y{\neq}[] \vee x{=}[]$, amounts to $x{=}[] \equiv y{=}[]$, which excludes all possible representations of the singleton lists. Hence, the restriction $x{\neq}[] \vee y{=}[]$ is too strong. We weaken it to $x{\neq}[] \vee \#y{\leqslant}1$, or, equivalently, $1{\leqslant}\#x \vee \#y{\leqslant}1$. For y , $\#y{\leqslant}1$, we have $rev{\cdot}y = y$; thus, we obtain the following definition for lhd :

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

By symmetry, we restrict set V to those pairs $[x,y]$ satisfying $1{\leqslant}\#y \vee \#x{\leqslant}1$ as well. Together, these two restrictions define set V : V now is a subset of $\mathcal{L}_* \times \mathcal{L}_*$. The relation defining this subset, also called the *representation invariant*, is Q , with:

Q: $(1 \leq \#x \vee \#y \leq 1) \;\wedge\; (1 \leq \#y \vee \#x \leq 1)$.

The definition for rhd then becomes --notice the symmetry-- :

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

For the derivations of definitions for the other functions we shall use the following simple lemma.

lemma 0: $Q \;\Leftarrow\; \#x = 1 \vee \#y = 1$

□

5 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:

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

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

$$
\begin{array}{lll}
a \vdash [x,y] & = & [\,a\,;x\,,y\,] \;, \\
[y,x] \vdash a & = & [\,y\,,a\,;x\,] \;.
\end{array}
$$

The expression $[a;x,y]$, however, need not satisfy Q : Q's second conjunct may be false, but it certainly is true if $1 \leqslant \#y$. For the special case $y=[]$, we redo the above derivation:

$\llbracket a \vdash [x,[]] \rrbracket$

= { as before , with [] for y }

$[a] +\!\!+ x +\!\!+ rev{\cdot}[]$

= { $rev{\cdot}[]=[]$, [] is the identity of $+\!\!+$ }

$[a] +\!\!+ x$

= { $[x,[]]$ satisfies Q , hence $\#x \leqslant 1$, hence $x = rev{\cdot}x$ }

$[a] +\!\!+ rev{\cdot}x$

= { definition of $\llbracket \cdot \rrbracket$ }

$\llbracket [[a],x] \rrbracket$.

Expression $[[a],x]$ satisfies Q because of lemma 0. Thus, we obtain the following definition for $\vdash$ and, similarly, for $\dashv$:

```
a⊢[x,y]  =  if y≠[] → [a;x,y]
            ▯ y=[] → [[a],x]
            fi ,

[y,x]⊣a  =  if y≠[] → [y,a;x]
            ▯ y=[] → [x,[a]]
            fi .
```

The (normal) time complexity of these definitions is $O(1)$; in order that their amortized time complexity is $O(1)$ too, the credit function must be chosen such that its value increases by a bounded amount under these operations.

6 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 , we derive:

$[\![\, ltl \cdot [x,y] \,]\!]$

= { specification of ltl }

$tl \cdot [\![\, [x,y] \,]\!]$

= { definition of $[\![\cdot]\!]$ }

$tl \cdot (x +\!\!+ rev \cdot y)$.

Further manipulation of this formula requires distinction of the cases $x \neq [\,]$ and $x = [\,]$. For the case $x = [\,]$ we have:

$tl \cdot (x +\!\!+ rev \cdot y)$

= { $x = [\,]$ }

$tl \cdot (rev \cdot y)$

= { $\#y = 1$ (note 0, see below) , hence $tl \cdot (rev \cdot y) = [\,]$ }

$[\,]$

= { specification of $[\,]_V$ }

$[\![\, [\,]_V \,]\!]$

= { definition of $[\,]_V$ }

$[\![\, [\,[\,],[\,]\,] \,]\!]$.

Hence, for the case $x = [\,]$ we choose $ltl \cdot [x,y] = [\,[\,],[\,]\,]$.

note 0: From $Q \wedge x = [\,]$ it follows that $\#y \leqslant 1$. The precondition of $ltl \cdot [x,y]$ is $[\![\, [x,y] \,]\!] \neq [\,]$, which equivales $x \neq [\,] \vee y \neq [\,]$. So, because $x = [\,]$, we have $y \neq [\,]$; hence: $\#y = 1$.

□

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

$\quad \mathrm{tl}\cdot(x+\!\!+\mathrm{rev}\cdot y)$

$=\quad \{\ x\neq[\,]\ ,\ \text{definition of } +\!\!+\ \}$

$\quad \mathrm{tl}\cdot x +\!\!+ \mathrm{rev}\cdot y$

$=\quad \{\ \text{definition of } [\![\cdot]\!]\ \}$

$\quad [\![\,[\mathrm{tl}\cdot x\,,\,y\,]\,]\!]$.

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

$\quad Q(x,y\leftarrow \mathrm{tl}\cdot x,y)$

$=\quad \{\ \text{definition of } Q\ \}$

$\quad (1\leqslant \#(\mathrm{tl}\cdot x)\ \vee\ \#y\leqslant 1)\ \wedge\ (1\leqslant \#y\ \vee\ \#(\mathrm{tl}\cdot x)\leqslant 1)$

$=\quad \{\ \text{definition of } \#\ \}$

$\quad (2\leqslant \#x\ \vee\ \#y\leqslant 1)\ \wedge\ (1\leqslant \#y\ \vee\ \#x\leqslant 2)$

$=\quad \{\ Q \Rightarrow 1\leqslant \#y\ \vee\ \#x\leqslant 2\ \}$

$\quad 2\leqslant \#x\ \vee\ \#y\leqslant 1$

$\Leftarrow\quad \{\ \text{predicate calculus}\ \}$

$\quad 2\leqslant \#x$.

So, for the special case that x has at least 2 elements the above definition for ltl is correct. Remains the case that x is a singleton list:

$\quad \mathrm{tl}\cdot x +\!\!+ \mathrm{rev}\cdot y$

$=\quad \{\ \#x=1\ \}$

$\quad [\,] +\!\!+ \mathrm{rev}\cdot y$

$=\quad \{\ [\,] \text{ is the identity of } +\!\!+\ \}$

$\quad \mathrm{rev}\cdot y$

$=\quad \{\ \text{introduce } u \text{ and } v \text{ such that } y=u+\!\!+v \text{ (note 1, see below)}\ \}$

$\quad \mathrm{rev}\cdot(u+\!\!+v)$

$=\quad \{\ \text{list calculus}\ \}$

$$\begin{aligned} & \mathrm{rev}\cdot v +\!\!+ \mathrm{rev}\cdot u \\ = \quad & \{ \text{ definition of } [\![\cdot]\!] \} \\ & [\![[\mathrm{rev}\cdot v , u]]\!] \ . \end{aligned}$$

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

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

□

Evaluation of $[\mathrm{rev}\cdot v , u]$ takes $O(\#y)$ time, independently of how u and v have been chosen. In order to obtain $O(1)$ amortized time complexity, the value of the credit function must decrease by an amount that is at least linear in $\#y$. I.e., $c\cdot[x,y] - c\cdot[\mathrm{rev}\cdot v,u]$ must be linear in $\#y$, where c denotes the credit function.

7 The credit function

In order 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 differently, 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:

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

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 $c\cdot[x,y] - c\cdot[rev\cdot v,u]$ is linear in $\#y$. We have:

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

This formula is linear in $\#y$. Hence, the decrease of c is linear in $\#y$, provided that we see to it that $c\cdot[rev\cdot v,u]$ is not too large:

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

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

$$u +\!\!\!+\, v = y \;\wedge\; \#u \leqslant \#v \leqslant \#u{+}1 \quad .$$

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 $u,v : u \mathbin{+\!\!+} v = y \wedge \#u = k$ 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 pieces together we obtain the following definitions for ltl and its symmetric counterpart rtl :

```
ltl·[x,y]  =  if #x = 0 → [ [] , [] ]
              [] #x = 1 → [ rev·(y↓k) , y↑k ]  where k = #y div 2 end
              [] #x ≥ 2 → [ tl·x , y ]
              fi  ,

rtl·[y,x]  =  if #x = 0 → [ [] , [] ]
              [] #x = 1 → [ y↑k , rev·(y↓k) ]  where k = #y div 2 end
              [] #x ≥ 2 → [ y , tl·x ]
              fi  .
```

8 Epilogue

A formalised 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 $\text{rev}{\cdot}[\![\, [x,y] \,]\!] = [\![\, [y,x] \,]\!]$, for all x and y . Hence, rev can be implemented efficiently by a function Rev , defined by $\text{Rev}{\cdot}[x,y] = [y,x]$. Thus, list reversal becomes an $O(1)$ operation.

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

[0] D. Gries
The Science of Programming
(section 19.3: Efficient queues in LISP)
Springer-Verlag, New York, 1981.

[1] R.E. Tarjan
Amortized Computational Complexity
SIAM Journal on Algebraic and Discrete Methods 6, 1985, pp. 306-318.

(Eindhoven, 24 september 1991)