Part II

COMPUTER SCIENCE
AN EFFICIENCY COMPARISON OF SOME REPRESENTATIONS OF PURELY FUNCTIONAL ARRAYS *

ANNIKA AASA, SÖREN HOLMSTRÖM and CHRISTINA NILSSON
Programming Methodology Group, Department of Computer Sciences,
Chalmers University of Technology, S-41296 Göteborg, Sweden

Abstract.

We describe an efficient representation of purely functional arrays suggested by Holmström and Hughes, here called version tree arrays or v-arrays. A collection of arrays is represented by a version tree where the root array is stored in contiguous memory as usual and every other array is represented as a node that records the difference to its father. Updating is done by adding a new node to the version tree and takes constant time, while the time for subscripting is proportional to the distance to the root. Essential for efficiency is therefore the possibility to reroot the tree without changing the semantics of the subscript operation, i.e. the values that the arrays represent.

We also show how v-arrays can be implemented in Standard ML and compare the efficiency of v-arrays to other representations, like binary trees, lists and functions, by measuring the execution time of some programs that use sequences. v-arrays turn out to be superior in practice in algorithms that use arrays singlethreadedly and access them randomly.

CR categories: D.1.1., E.1.

1. Introduction.

Functional programs are often slower than their imperative counterparts due to the lack of a storage structure where sequences of data can be selected and replaced in short constant time. Imperative languages have arrays for this purpose, and the good time behaviour of many imperative programs depends on the use of arrays. Introducing efficient arrays in a pure functional language is not straightforward. The naive implementation has to copy the array on updating in order to preserve referential transparency and this is of course costly if the array is large.

An implementation method that is efficient in many cases and also preserves referential transparency was found by Holmström [3] and generalized by Hughes [6]. The same idea has been discussed in [4] and in a Logic Programming context [5]. In this paper we shall, in a slightly different way, describe Hughes'

* This work has been supported by STU, the Swedish Board for Technical Development.
Received June 1987. Revised May 1988.