Lisp and Symbolic Computation, 2, 105–113 (1989)

Arrays in Pure Functional Programming Languages

KLAUS E. GRUE
DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen East, Denmark (grue @ diku. dk)

Abstract. In pure functional programs it is common to represent arrays by association lists. Association lists have the disadvantage that the access time varies linearly both with the size of the array (counted in number of entries) and with the size of the index (counted in cons nodes). This paper presents another simple representation of arrays for which the access time varies linearly in the size of the index but is independent of the size of the array. The paper compares this representation with association lists in functional languages and arrays in imperative languages.

This paper also considers lazy programming and states how to use potentially infinite arrays for time optimization for certain programs.

1. Introduction

Lisp [9, 10] has infinitely many constructors: **cons**, and one for each atom. Hence, the domain of Lisp, i.e. the set of S-expressions, is infinitely generated. Contrarily, the domains for HyperLisp [11] and Formal Language [5] are finitely generated. The generators are {**cons**, **snoc**, **nil**}, and {**cons**, **nil**}, respectively. Furthermore, most implementations of Lisp are finitely generated. As an example, the domain of an implementation of Lisp that offers a function for concatenation of atom names is generated by this concatenation function, the atoms whose names consist of one character, and **cons**.

The representation of arrays presented in this paper works for finitely generated domains, i.e., array indices must be restricted to some finitely generated domain. For languages with finitely generated domains, array indices may vary over the entire domain of the language. Hence, one need not impose restrictions on array indices in HyperLisp, Formal Language, and most implementations of Lisp. The range of values that may be stored in arrays never needs to be restricted.

The representation of arrays works best for finitely generated domains with few constructors, and becomes particularly simple for the domain of Formal Language which is generated by {**cons**, **nil**}. This paper states how to represent arrays for this domain and describes how to treat other cases.

As practical application of languages with few constructors seems quite new [5, 11], and as their use is not widespread, we now list some advantages of such languages.

1. It is possible to make languages with few constructors which have simple and