Available online at www.sciencedirect.com
SCIENCE DIRECT®
Science of Computer Programming 58 (2005) 141–178
Science of Computer Programming
www.elsevier.com/locate/scico

Static insertion of safe and effective memory reuse commands into ML-like programs☆

Oukseh Lee[a,*], Hongseok Yang[b], Kwangkeun Yi[b]

[a] *Department of Computer Science & Engineering, Hanyang University, Ansan Gyeonggi 426-791, Republic of Korea*
[b] *School of Computer Science & Engineering, Seoul National University, Sillim-9-dong Gwanak-gu, Seoul 151-742, Republic of Korea*

Received 23 October 2003; received in revised form 31 August 2004; accepted 17 February 2005
Available online 24 May 2005

Abstract

We present a static analysis that estimates reusable memory cells and a source-level transformation that adds explicit memory reuse commands into the program text. For benchmark ML programs, our analysis and transformation system achieves a memory reuse ratio from 5.2% to 91.3% and reduces the memory peak from 0.0% to 71.9%. The small-ratio cases are for programs that have a number of data structures that are shared. For other cases, our experimental results are encouraging in terms of accuracy and cost. Major features of our analysis and transformation are: (1) polyvariant analysis of functions by parameterization for the argument heap cells; (2) use of multiset formulas in expressing the sharings and partitionings of heap cells; (3) deallocations conditioned by dynamic flags that are passed as extra arguments to functions; (4) individual heap cells as the granularity of explicit memory reuse. Our analysis and transformation system is fully automatic.

Keywords: Program analysis; Program transformation; Type system; Compile-time garbage collection

---

☆ Lee and Yi were supported by the Brain Korea 21 (2003–2004) of Korean Ministry of Education and Human Resource Development; Lee was supported by the research fund of Hanyang University (HY-2004-G); Yang was supported by R08-2003-000-10370-0 from the Basic Research Program of the Korea Science and Engineering Foundation; and Yi was supported by Korea Research Foundation KRF-2003-041-D00528.

* Corresponding author.
E-mail addresses: oukseh@hanyang.ac.kr (O. Lee), hyang@ropas.snu.ac.kr (H. Yang), kwang@cse.snu.ac.kr (K. Yi).

0167-6423/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2005.02.007

1. Overview

Our goal is to automatically insert explicit memory reuse commands into ML-like programs so that they do not blindly request memory when constructing data. We present a static analysis and a source-level transformation system that automatically adds explicit memory reuse commands into the program text. The explicit memory reuse is accomplished by inserting explicit memory-free commands right before data-construction expressions. Because the unit for deallocation and allocation is an individual cell, such deallocation and allocation sequences can be implemented as memory reuses.[1]

Example 1. Function call "`insert i l`" returns a new list where integer `i` is inserted into its position in the sorted list `l`.

```
fun insert i l =
  case l of [] => i::[]                                    (1)
          | h::t => if i<h then i::l                       (2)
                    else h::(insert i t)                   (3)
```

Let us assume that the argument list `l` is not used after a call to `insert`. If we program in C, we can destructively add one node for `i` into `l` so that the `insert` procedure should consume only one cons-cell. Meanwhile, the ML program's line `(3)` will allocate as many new cons-cells as that of the recursive calls. Knowing that list `l` is not used any longer, we can reuse the cons-cells from `l`:

```
fun insert i l =
  case l of [] => i::[]
          | h::t => if i<h then i::l
                    else let z = insert i t
                         in (free l; h::z)                 (4)
```

In line `(4)`, "`free l`" will deallocate the single cons-cell pointed to by `l`. The very next expression's data construction "`::`" will reuse the freed cons-cell. □

1.1. Related works

The type systems [25,24,2] based on linear logic fail to achieve the Example 1 case because variable `l` is used twice. Kobayashi [10], and Aspinall and Hofmann [1] overcome this shortcoming by using more fine-grained usage aspects, but their systems still reject Example 1 because variables `l` and `t` are aliased at line `(2)`–`(3)`. They cannot properly handle aliasing: for "`let x=y in` *e*" where `y` points to a list, this list cannot in general be reused at *e* in their systems. Moreover, Aspinall and Hofmann did not consider an automatic transformation for reuse. Kobayashi provides an automatic transformation, but he requires the memory system to manage a reference counter for every heap cell.

Deductive systems like separation logic [9,16,17] and the alias-type system [18,26] are powerful enough to reason about shared mutable data structures, but they cannot be used

---

[1] The drawback of this approach might be that the memory reuse "bandwidth" is limited by the data-construction expressions in the program text. But our experimental results show that such a drawback is imaginary.