Progress Report

Nguyen Huu Duc
School of Information Science
Japan Advanced Institute of Science and Technology
duchuu@jaist.ac.jp

March 1, 2005

1 Aims of the Research

The goal of the present work is to develop a compilation method for ML polymorphism that supports interoperable heap management so that a common heap space can be shared by ML and other languages including Java and C, while keeping the full benefits of ML-style polymorphic languages.

2 Method

Most of the current implementations of functional languages adopt the so-called "tagged data representations," where each word contains a one bit tag indicating whether it is a pointer or not. This yields particularly simple memory management: a compiler only needs to set one bit when emitting code, and a tracing garbage collector can locate all the pointers in a heap block by simply scanning the tag bit in each word in the block. A well-known drawback to this approach is the lack of interoperability with other languages. Since integers and other atomic data do not have their natural runtime representation, they must be converted each time when they are passed to other languages.

In order to solve this problem, we consider the following two features to be prerequisites to achieve high-degree of interoperability.

1. Integers and other atomic data have their natural representations.
2. Each heap block includes its layout information for garbage collection.

Those features can be achieved by compiling ML so that each runtime object (a heap block or a stack frame) has a bitmap that describes the pointer positions in the block. For example, a nested record $(1,(2,3))$ is implemented as $([0,1];1,([0,0];2,3))$ where $[0,1]$ is a bitmap indicating the two word block of an atom and a pointer to a block, which in this case is another record $([0,0];2,3)$. However, for a polymorphic language such as ML, generating a correct bitmap for each block allocation expressions constitutes a challenge. The bitmap of a polymorphic record can not be directly generated due to the polymorphic types of some record elements.

The general idea is to encode the necessary information for composing bitmaps in polymorphic types. Those information are statically computed at the time of type instantiation, passed through the code as arguments and dynamically composed to generate necessary bitmaps.

The main result of this research is to realize this idea by

- defining a bitmap-passing calculus for representing bitmap information and bitmap computation
- developing a type-directed bitmap-passing compilation algorithm which transforms typed lambda expressions into bitmap-passing expressions.

3 Progress

Following the proposed method, several works have been done in order to fulfill the requirement of an interoperable memory management system.

- Defining a typed bitmap-passing calculus with rank-1 polymorphism and proving the type system is sound with respect to an operational semantics that models bitmap-inspecting garbage collection.
- Develop a type-directed bitmap-passing compilation algorithm and showing it preserves typing.
- Solving the problem of mutual dependency among bitmap-passing compilation, closure conversion and A-normalization.
- Realizing above theoretical results in the IML project.

4 Future Direction

In order to achieve a desirable interoperable feature of ML, some future works may involve in this research.

- Adopting bitmap-passing compilation algorithm to represent unboxed multi-word objects such as floating points.
- Investigating the problems of separate compilation and linking of ML modules and libraries implemented by other languages.

5 Result

- A paper, namely "Type-directed compilation of ML supporting interoperable garbage collection", is to be submitted to SCP Special Issue on Memory Management.
- An working version of bitmap compiler for IML project.