A Haskell to Java Virtual Machine Code Compiler

David Wakeling

Department of Computer Science, University of Exeter,
Exeter, EX4 4PT, United Kingdom.
(web: http://www.dcs.exeter.ac.uk/~david)

Abstract. For some time now, we have been interested in using Haskell to program inexpensive embedded processors, such as those in SUN's new Java family. This paper describes our first attempt to produce a Haskell to Java Virtual Machine code compiler, based on a mapping between the G-machine and the Java Virtual Machine. Although this mapping looks good, it is not perfect, and our first results suggest that the compiled Java Virtual Machine code may be rather larger and slower than one might hope.

1 Introduction

For some time now, we have been interested in the efficient implementation of lazy functional programming languages on very small computers, such as those found in consumer electronics devices. So far, all of our implementations have assumed that next-generation products will be controlled by previous-generation RISC processors [8]. But Java processors, with their compact instruction encoding, are an attractive alternative [7]. This paper investigates whether these processors could successfully run lazy functional programs.

The paper has two parts. The first part points out the similarity between the virtual machine usually used to implement Java [3] and the Chalmers G-machine [5], a virtual machine often used to implement lazy functional languages. Section 2 gives a quick tour of the Java Virtual Machine, Section 3 gives a quick tour of the G-machine, and Section 4 describes a mapping between the two virtual machines that can serve as the basis of a lazy functional language implementation. The second part assesses the effectiveness of the mapping, and suggests how it could be improved. Section 5 presents some benchmark figures, Section 6 discusses these figures, and Section 7 has some ideas for future improvements. Section 8 mentions some related work, and Section 9 concludes.

2 The Java Virtual Machine

In principle, Java could be compiled for any machine, but in practice it is usually compiled for a standard virtual machine. This section gives a quick tour of the Java Virtual Machine; more detail can be found in [3]. Throughout this paper,
Java source code and Java Virtual Machine code will be written in typewriter font.

As Figure 1 shows, the Java Virtual Machine is a stack-based virtual machine which works with methods and constant pools, objects, and two stacks.

2.1 Methods and Constant Pools

A Java program is organised into classes, each of which may have some methods (or functions) for performing computation. For every class, the Java Virtual Machine stores the virtual machine code for each method, and a constant pool of literals, such as numbers and strings, used by the methods. To ensure the binary portability of Java programs, the layout and byte-order of the stored form is precisely specified. Nevertheless, before the Java Virtual Machine runs any untrusted code it verifies it in an attempt to ensure that it is well-behaved.

2.2 Objects

As well as providing methods, classes also describe the structure of objects. An object is a record whose fields may be either scalar values, methods or references to other objects. There are virtual machine instructions for allocating a new object, for setting and getting the value of a field, and for invoking a method. But there is no instruction for disposing of an object. A garbage collector is assumed to run from time-to-time to recover the memory occupied by objects that are no longer in use.