Interpreter for Machine Programs on Arbitrary Models of Computation

Konrad Wienecke

Sometime 2020

Contents

1 Introduction

The aim is to create an interpreter for a meta-language which can be used to write machine programs for any kind of machine, e.g. machines using any arbitrary model of computation to compute programs. While one use of the interpreter is to execute machine programs on a given input, another use is the tracing of program executions. These traces can be used for reasoning about the program, for instance in the context of debugging or when learning to understand an unknown program.

1.1 Models of Computation

Models of computation describe the tools of a machine that can be used by programs written for machines of that type. Fundamentally, a model of computation defines two main interfaces which can be used to interact with the internal state of a machine, which can otherwise not be interacted with: Operations and predicates.

Operations can be used to change the internal state whereas predicates give some insight about the internal state of the machine by returning a

truth value. You can think of these interfaces as mathematical functions. Both operations and predicates are identified by their respective name:

$$operation_{opname}: MachineState \rightarrow MachineState$$

$$predicate_{predname}: MachineState \rightarrow \{true, false\}$$

In practice, we need some way to work with machine states. We choose strings, which can encode many different data structures. In Haskell, encoding and decoding data structures to and from strings is in many cases quite easy using the *read* and *show* functions from the Prelude. Because of this, our model of computation provides one final interface, which is simply a function to check machine states for validity:

$$validstate: String \rightarrow \{true, false\}$$

While the operations and predicates are available and meant for use in programming, this third interface is used before running a program to check that the input is a valid machine state which we can then execute the program on.

1.1.1 A Simple Stack Machine

One concrete example of a model of computation is the stack machine. The stack machine has a given number of registers, or stacks, and a given alphabet that dictates which symbols can exist on the stacks. The stacks are indexed by integers (starting with 0) and the alphabet consists only of alphanumerical symbols, encoded as chars. The model defines the typical push and pop operations for each stack (and in the case of the push operation, for each symbol). In this specific implementation, these operations have the following names:

- Ri+s denotes pushing the symbol s onto the stack with the index i
- Ri- denotes popping the topmost symbol off the stack with the index i

Note that the pop-operation does not "return" the element that was taken off the stack, since operations only return one thing, and that is the changed machine state. To find out information about the stacks, these predicates exist:

- Ri=s is true iff the topmost element on stack i is the symbol s
- Ri=_ is true iff the stack i is empty

1.2 Machine Programs

Machine programs use a model of computation in order to compute their output from a given input. The meta-language allows us to formulate simple programs as a set of program states (not to be confused with machine states) with corresponding operations and binary decision trees. To execute a program, we start at the "Start"-State and evaluates its decision tree: Predicates are evaluated on the current machine state to find the resulting next state. Each state other than the start state also has a corresponding operation, which is ran on the machine state before evaluating its decision tree on the new machine state.