On Some Semantic Issues in the Reflective Tower

Karoline Malmkjær*
DIKU - Computer Science Department, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen Ø, DENMARK
e-mail: karoline@diku.dk

Abstract

Introducing meta-level access in a programming language is a delicate task, surrounded by threats of vicious circularity, infinite regress and unresolvable paradoxes on the one hand and triviality on the other hand. The reflective tower is one of the attempts to structure computational reflection, that is, access from a running process to its computational state.

This paper gives the framework of a denotational semantics of a reflective tower, isolating the features which we have found to be sufficient to describe this model. The framework is then used to analyze some concrete problems in a reflective tower based on the programming language Scheme with continuations as applicable objects.

The paper first gives a brief description of the reflective tower and discusses the unstable points in the design. Then the denotational description is outlined and used to analyze these points and to clarify the consequences of the necessary design decisions.

1 Introduction

The reflective tower [Smith 82, Wand & Friedman 88, Danvy & Malmkjær 88] is a design introducing new structured programming facilities. This design still holds ambiguities, and for some facilities it is not clear whether they are desirable or not and whether they are optional or intrinsic. A description of a reflective tower in denotational semantics provides the means for identifying a number of the arbitrary choices and their consequences and it gives a clear description of why and how this conceptually infinite design can be implemented finitely.

The reflective tower is designed to structure facilities of meta-level access (sometimes called reflective or introspective facilities). The aim is to be able to access, test, modify, and reinstall the state of any program from the program itself while maintaining a proper distance from the program when it is being modified.

To fulfill the criteria of proper distance, access is only given to special procedures (called reifiers) which, when they are applied, run in the interpreter of the program.
rather than in the program itself. In order to be able to write the reifiers in the same language as the rest of the program, the interpreter is assumed to be meta-circular [Reynolds 72]. Since there are no restrictions on the actions that can be performed in the reifiers (e.g., applying another reifier) this leads to the picture of an infinite tower of interpreters, each interpreting the interpreter below it, until the user program.

This can then be simulated using some circular structure to represent the infinity of interpreter states — it is not necessary to represent the infinity of interpreter programs since they are identical.

With the facilities of the reflective tower it is possible — in the language itself — to define environment and control manipulating procedures such as Scheme's call-with-current-continuation (abbreviated call/cc) [Rees & Clinger 86]. Call/cc, however, may also serve as an example of the problems of the reflective tower. [Wand & Friedman 88] attempts two definitions of call/cc, and neither of them corresponds to the Scheme procedure. In order to see why, we first have to take a closer look at the access that is provided in the reflective tower and the movements in the tower that accompany the use of this access.

The next section presents the reflective tower. Section 3 sketches a denotational description of a reflective tower. This is then used in section 4 and section 5 to clarify some problems of the access to control and of the concept of scope in a multi-level setting and to outline solutions. Section 6 discusses the particular problems of reinstalling expressions.

2 The Reflective Tower

The tower is pictured as an infinite tower of levels, where each level represents a running program. The user program runs at the bottom of the tower and each level above it is an interpreter interpreting the level below itself (if it was not interpreting, there would not be any levels below).

```
            +-----------+
            |  0  |
            |      |
            +------+
            |  1  |
            |      |
            +------+
            |  2  |
            |      |
            +------+
            |      |
            +-----------+
```

The languages of the reflective towers designed so far are mainly Scheme-like expression languages.

The interpreter is a continuation-passing, tail-recursive program manipulating three state variables representing the current expression, the environment in which to evaluate the expression, and the continuation (or control) that requests the result of