Towards a Semantics of Emerald Expressed in Map Theory

Klaus Grue

December 5, 1994

Abstract

This paper presents a semantics for a subset of an object oriented, concurrent and distributed language with Emerald as the chosen example. The concurrent and distributed aspects posses particular problems, and the paper focuses on these aspects. In particular the paper covers parallelism in which a created process can continue to run after the creating process halts. The semantics is presented in a continuation passing style in map theory.

Map theory is a foundation of mathematics with the same expressive power (w.r.t. consistency) as set theory, but it builds upon λ-calculus and functions instead of logic and sets. It allows unlimited recursion which, combined with its expressive power, makes it ideal for expressing semantics of programming languages.

The paper serves three purposes: First, it gives a semantics for a subset of Emerald, which can be scaled up to cover all of Emerald. Second, it shows how map theory can be used in definitions of semantics. Third, it shows how distribution and parallelism can be modelled.

Contents

1 Introduction

Emerald [6,7,16,17,15,18,19,20,25] is a novel object oriented, concurrent and distributed language. It has been adopted for teaching of first year students at the University of Copenhagen. This paper presents a semantics of a subset of Emerald. The concurrency and distribution aspects of Emerald are difficult to express in traditional frameworks, so the present paper focuses on these aspects.

The reader is assumed to know Emerald as described in [15], but Section 5 will be a sufficient introduction to many readers.

The semantics of programming languages are traditionally defined axiomatically [10,11,9], in lambda calculus [3,5,21], in set theory [1,2,4,23] or in other suitable theories [14,24]. The axiomatic approach has the benefit that it highlights a number of basic properties of the language from which all other properties can be deduced. A drawback of this method is that consistency of the basic properties has to be established somehow. Another drawback is that it is necessary to develop a new style of proof and a new collection of metatheorems for each new axiomatisation.

In the denotational approach, the semantics is defined within some, previously established theory. That theory can be a general purpose one like set theory, it can be tailored to computer science like lambda calculus, or it can be even further tailored to semantics like CCS and CSP [14,24]. This paper uses the denotational approach, but expresses the semantics in map theory [8] which is a general purpose competitor to set theory. The semantics makes heavy use of the unlimited recursion available in map theory.

A semantics of a programming language is by nature a definition. For that reason, the present paper contains a definition and no theorems or proofs. For some of the above mentioned approaches, it is necessary or at least convenient to accompany each semantics by proofs such as consistency, soundness, completeness or congruence proofs. This is not necessary or relevant in the present