Dependently Typed Programming in Agda

Ulf Norell[1] and James Chapman[2] *

[1] Chalmers University, Gothenburg
ulfn@chalmers.se
[2] Institute of Cybernetics, Tallinn
james@cs.ioc.ee

1 Introduction

In Hindley-Milner style languages, such as Haskell and ML, there is a clear separation between types and values. In a dependently typed language the line is more blurry – types can contain (*depend on*) arbitrary values and appear as arguments and results of ordinary functions.

The standard example of a dependent type is the type of lists of a given length: `Vec A n`. Here `A` is the type of the elements and `n` is the length of the list. Many languages allow you to define lists (or arrays) of a given size, but what makes `Vec` a true dependent type is that the length of the list can be an arbitrary term, which need not be known at compile time.

Since dependent types allows types to talk about values, we can encode properties of values as types whose elements are proofs that the property is true. This means that a dependently typed programming language can be used as a logic. In order for this logic to be consistent we need to require programs to be total, i.e. they are not allowed to crash or non-terminate.

The rest of these notes are structured as follows: Section 2 introduces the dependently typed language Agda and its basic features, and Section 3 explains a couple of programming techniques made possible by the introduction of dependent types.

2 Agda Basics

Agda is a dependently typed language based on intuitionistic type theory[4]. Its current version (Agda 2) is a complete rewrite instigated by Ulf Norell during his PhD[6] at Chalmers University in Gothenburg. This section introduces the basic features of Agda and how they can be employed in the construction of dependently typed programs. Information on how

* Please send bug reports for this tutorial to James.