Polytypic Programming
— abstract —

Johan Jeuring
Chalmers University of Technology and University of Göteborg
S-412 96 Göteborg, Sweden
email: johanj@cs.chalmers.se

There are many programming problems that can be formulated independent of a specific datatype. For example, the pattern matching problem can be informally specified as follows: given a pattern and a text, find the first occurrence of the pattern in the text. The pattern and the text may be both lists, or they may be both trees, or they may be both multidimensional arrays, etc. Many algorithms solving a programming problem which is originally formulated on the datatype *List* have later been extended to solve 'the same' problem on other datatypes.

I call a function that is defined on arbitrary datatypes a polytypic function, i.e., a polytypic function is a function that is defined on lists, binary trees, and all other 'tree-like' datatypes. A familiar example of a polytypic function is the function `map`, which takes a function `f :: a -> b`, and a value of some datatype `D a` (for example lists over the type `a` or binary trees over the type `a`), and applies function `f` to all elements of type `a` in the value, obtaining a value of type `D b`. Often a polytypic function is also polymorphic, but it need not be. An example of a polytypic function that is not polymorphic is the function that generalises summing a list to arbitrary datatypes.

In this talk I discuss polytypic functions. I combine the polytypic functions that have been defined in the Bird-Meertens calculus, a calculus for transformational programming developed over the last decade, with inductive definitions of natural transformations to build new polytypic functions. Examples of such new polytypic functions are the function `size`, which generalises the function `length` defined on lists to arbitrary datatypes, and the function `pattern_match`, which generalises a version of the pattern-matching algorithm of Knuth, Morris, and Pratt to arbitrary datatypes. Furthermore, I discuss Hollum, an experimental system that supports polytypic programming. Using Hollum it is possible to use functions like `map_Data`, and `pattern_match_Data` for every `data Data` declared in the program. Hollum takes a program written in the functional programming language Gofer, and returns the same program to which instances of polytypic functions are added for each data declared in the original program.

This is joint work with Graham Hutton (Utrecht University) and Oege de Moor (Oxford University).