Adding Imperative Programming to The Pattern Calculus

by

Quy Tuan Nguyen

a thesis submitted for the degree

Doctor of Philosophy

2005

Certificate of Authorship/Originality

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text. I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate

Production Note:
Signature removed prior to publication.

Acknowledgment

First and foremost, I would like to thank my supervisor Barry Jay whose help, stimulating suggestions and encouragement helped me in all the time of research for and writing of this thesis. Thanks to Murray Cole, Mohammad Hamdan, Ryan Heise, Gabi Keller, Hai Yan Lu, Clara Murdaca and David Skillicorn for valuable advice and suggestions.

Also, I would like to give my thanks to my family and friends for all support and encouragement.

Contents

List of Figures

Abstract

By focusing on data and flow control, imperative languages provide a finely grained and efficient mechanism for directly manipulating state and memory. By focusing on functions, polymorphism increases the modularity and reusability of programs. The pattern calculus gives a new account of polymorphism over arbitrary datatypes which has been used as the foundation for building the functional language FISh2. The power of the new polymorphism is not limited to a functional setting and it can be extended into an imperative setting. The main contribution of this thesis is to expand the pattern calculus with imperative features and implement this within a version of FISh2.

Two approaches are developed in expanding the calculus to imperative programming based on two setting: functional and imperative. Based on a functional setting, updatable locations are given separate location types; while based on an imperative setting, locations and their values share the same types. In both approaches, structured locations can be defined in the same way the calculus defines structured data. Hence, generic functions on locations can be defined by pattern-matching on (location) constructors. In that way, the power of the combination exceeds that of the boundary of functional or imperative alone. In particular, with the generic assignment function, we have a new approach on memory management which performs inplace update whenever it is reasonable to do so.

Similar ideas could be used to extend the power of parametric polymorphism to parallel programming. To illustrate the approach, a key problem is addressed in detail, namely, distributing a data structure over a network of processors.