Implementing the Pure Pattern Calculus: Abstract

Matt Roberts (mattr@ics.mq.edu.au)
Department of Computing, Macquarie University

September 11, 2006

1 Introduction

The Pure Pattern Calculus [JK06] is a generalisation of the lambda calculus based on *pattern-matching* rather then beta-reduction. In many ways, it is similar to the lambda calculus, but the differences provide some very interesting capabilities. The Pure Pattern Calculus has a particular advantage over the lambda calculus when describing *polytypic* computations.

To this point, investigations into the Pure Pattern Calculus have been largely theoretical. To fully investigate the properties and potential uses of the Pure Pattern Calculus, we need to build a programming language from it. One with an efficient compiler and a reasonably efficient run-time. This is the goal of this project.

2 Operational Semantics

Our first step has been to create a simpler operational semantics from the Pure Pattern Calculus described in [JK06]. This simpler semantics is designed for easy and efficient evaluation without losing any of the capabilities that made this calculus interesting to us in the first place, these include:

- Data Polymorphism (i.e. Haskell style polymorphism).
- Structure Polymorphism (i.e. Polytypic programming).
- Path Polymorphism (described in [HJS06]).
- Pattern Polymorphism (abstracting functions over the patterns they operate on).

Our particular interest is the applicability of the Pattern Calculus to program transformation, compiler generation and general purpose polytypic programming.

3 Practical Completeness

The resulting semantics is not as general as the pure pattern calculus. There are some expressions that have meaning in the pure pattern calculus, but are meaningless in the semantics we have formulated. It is our claim that the expressions we exclude are not important in the respect that we are not prevented from doing what we want to do. In particular, we can still encode 4 of the 5 types of polymorphism expressible in the Pure Pattern Calculus and we can embed the lambda calculus.

The fifth type of polymorphism is subtyping and is meaningless in the untyped setting here. We expect to be able to include it once types are added.

Importantly, all the previously identified areas of interest are intact. In fact we expect *all* meaningful programs of the Pure Pattern Calculus to be expressible in these semantics, although we are yet to demonstrate this formally. The only ones that will be excluded in this case are degenerate or pathological examples.

4 Future Work

This work is still in its early stages. We have taken the first important steps towards having an efficient compiler for the Pure Pattern Calculus. From here we must consider the following additional issues.

Lazy vs. Strict We have a preference for lazy semantics where this make sense. It will be interesting to see how a lazy pattern calculus behaves, in particular, just how lazy can we make pattern matching?

The Evaluation Mechanism There exists a large body of work on efficiently evaluating functional programming languages. While much of this translates immediately to the current setting, further work is needed to provide a fully optimised implementation, especially in the lazy context.

The Type System Describing untyped polytypic computations is an easy problem. Being able to describe them is a strongly typed language is much harder and much more useful. We are well advanced towards providing a strongly typed version of our language, along lines similar to those presented in [Jay06]. Once we have this, we can begin to explore the space of admissible programs and the characteristics of the type system.

Verifiable Compiler We also have an interest in formally verified compilers and will take the opportunity of implementing a prototype compiler to explore this this aspect of our system as well.

References

[HJS06] F. U. Huang, C. B. Jay, and D. B. Skillicorn. Programming with hetrogenous structures: Manipulating xml data using bondi. In *CRPITS'48: Proceedings of the 48th conference on Computer science 2006*, pages 287–295, Darlinghurst, Australia, 2006. Australian Computer Society, Inc.

[Jay06] Barry Jay. Typing first-class patterns. In *Proceedings of The Third International Workshop on Higher-Order Rewriting. To Appear*, 2006.

[JK06] Barry Jay and Delia Kesner. Pure pattern calculus. In Peter Sestoft, editor, *Programming Languages and Systems: 15th European Symposium on Programming*, volume 3924/2006 of *Lecture Notes in Computer Science*, pages 100–114. Springer, 2006.