B. Comp. Dissertation

**Automated Meta-Programming**

**to Support High-Performance OCaml Codes**

By

Arnold Christopher Koroa

Department of Computer Science

School of Computing

National University of Signapore

2014/2015

B. Comp. Dissertation

**Automated Meta-Programming**

**to Support High-Performance OCaml Codes**

By

Arnold Christopher Koroa

Department of Computer Science

School of Computing

National University of Signapore

2014/2015

Project No: H018590

Advisor: Assoc. Prof. Chin Wei Ngan

Deliverables:

Report: 1 Volume

Program: 1 CD

# Abstract

Meta prog is good but hard so automate

Subject Descriptors:

A

Keywords

Programming Languages & Systems, Program Analysis and Optimization

Implementation Software:

OCaml 4.02.1, BER MetaOCaml N102

# Table of Contents

[Abstract ii](#_Toc415059824)

[Table of Contents iii](#_Toc415059825)

[1. Introduction 1](#_Toc415059826)

[2. Tools 2](#_Toc415059827)

[2.1. MetaOCaml 2](#_Toc415059828)

[2.2. OCaml Extension Points 2](#_Toc415059829)

[2.3. OCaml ppx Preprocessor 3](#_Toc415059830)

[3. ppx\_toMeta 4](#_Toc415059831)

[3.1. Goal 4](#_Toc415059832)

[3.2. Translation Scheme 4](#_Toc415059833)

[3.3. Implementation 4](#_Toc415059834)

[3.3.1. Source Code Annotation 4](#_Toc415059835)

[3.3.2. ppx Preprocessor 4](#_Toc415059836)

[3.3.2.1. Hooking to default mapper 4](#_Toc415059837)

[3.3.2.2. Extracting information from the annotations 4](#_Toc415059838)

[3.3.2.3. Building the staged function 4](#_Toc415059839)

[3.3.2.4. Combining the results 4](#_Toc415059840)

[3.3.3. Generating and prettyprinting the staged code 4](#_Toc415059841)

[4. Conclusions 5](#_Toc415059842)

[4.1. Summary 5](#_Toc415059843)

[4.2. Limitations 5](#_Toc415059844)

[4.2.1. Scope of translation scheme 5](#_Toc415059845)

[4.2.2. Scope of preprocessor 5](#_Toc415059846)

[4.3. Recommendations for Further Work 5](#_Toc415059847)

[4.3.1. Removing annotations from helper functions 5](#_Toc415059848)

[4.3.2. Static analysis and optimization 5](#_Toc415059849)

1. Introduction

Meta-programs are programs that manipulates other programs. This allows for powerful optimizations by means of static analysis and transformation (Kiselyov, Swadi, & Taha, 2004; Taha, 2004), code specialization by partial evaluation and parameter removal (Carette, 2006; Frigo, 1999; Jonnalagedda, Coppey, Stucki, Rompf, & Odersky, 2014; Taha, 2004), automatic generation of efficient, potentially parallel, low-level codes (Bourgoin & Chailloux, 2014; Langhammer, 2005; Spampinato & Puschel, 2014), and empirical or machine-learning based generation of platform specific code (Puschel, 2011; Puschel et al., 2005; Whaley, Petitet, & Dongarra, 2001). Meta-programming is also useful for compiler generation and domain-specific language implementation (Czarnecki, O’Donnell, Striegnitz, & Taha, 2004; Futamura, 1999; Herrmann & Langhammer, 2006).

The many opportunities provided by meta-programming motivates us to find a suitable tool with which we can develop meta-programs easily. MetaOCaml, a meta-programming dialect of OCaml (Kiselyov, 2010), is one of such tools, allowing us to create meta-programs using a high-level language, OCaml, as its base. However, MetaOCaml requires its users to manually annotate their source code with staging constructs. This requires the users to have a deep understanding of the meaning and possible usages of the constructs before the user can build an efficient multi-staged program. Furthermore, manual annotation of existing code can be quite tedious, which increases the likelihood of human error and careless mistakes.

This project attempts to automate the staging process, enabling users to reap the benefits of multi-staged meta-programming without having to deal with the complexities of manually staging a source program. This is done by

There are attempts at abstracting over the staging process by objects etc

However this project attempts to go beyond that and

The report will be organized as follows: in section 2, I will introduce the tools; in section 3 I will describe my implementation; in section 4

1. Tools

This section describes the tools used in this project. They include MetaOCaml, the platform used for meta-programming in OCaml, and the ppx preprocessor used to implement the automatic staging of OCaml codes.

* 1. MetaOcaml

One of the main tool used in this project is MetaOCaml, a multi-staged flavor of the OCaml programming language (Kiselyov, 2010). The MetaOcaml version used in this project is BER MetaOCaml N102 which is an update from the original MetaML (Taha, 1999) for OCaml version 4.02.1.

MetaOCaml provides three constructs on top of the OCaml programming language to implement multi-staging

Bracket .< … >.

Delay the computation inside it

let plus2 x = .<x + 2>.;;

# plus2 3;;

: int code = .<3 + 2>.

Escape .~

Runs code pointed by it to produce code to be spliced

# .<.~(plus2 3) + .~(plus2 4)>.;;

: int code = .<(3 + 2) + (4 + 2)>.

Run !.

Executes a delayed computation

# !. .<.~(plus2 3) + .~(plus2 4)>.;;

- : int = 11

* 1. OCaml Extension Points

OCaml extension points (Frisch, 2013; Zotov, 2014) are generic structures embedded in the OCaml syntax (available from OCaml 4.02.1 onwards). These structures can then be

Camlp4 is an alternative but (de Rauglaudre, 2003)

Attributes

Extension Nodes

Quoted Strings

In this project, only attributes are used as what we need is only annotations on the original source code as will be explained in Section 3

* 1. ppx Preprocessor

As mentioned above, the generic extention point structures need to be preprocessed by a preprocessor to produce vanilla OCaml AST

To do this we need to access and modify the parsed AST

Compiler-libs

Asttypes Parsetree, the OCaml AST data structure

Ast\_mapper hook into it

Ast\_helper help write AST

1. ppx\_toMeta

The preproc inpml

* 1. Goal

Use a simple annotation scheme to cue the preprocessor to automatically stage functions

This is done by adding attributes to function definition and function calls to let the preprocessor know what to stage and how

* 1. Translation Scheme

A standard way to stage functions

Currently covers simple functions

Non recursive plus

Recursive pow (if then, match)

Also functions that uses previously staged function

* 1. Implementation
     1. Source Code Annotation
     2. ppx Preprocessor
        1. Hooking to default mapper
        2. Extracting information from the annotations
        3. Building the staged function
        4. Combining the results
     3. Generating and prettyprinting the staged code

1. Conclusions
   1. Summary
   2. Limitations
      1. Scope of translation scheme
      2. Scope of preprocessor
   3. Recommendations for Further Work
      1. Removing annotations from helper functions
      2. Static analysis and optimization

References

Bourgoin, M., & Chailloux, E. (2014). *GPGPU Composition with OCaml*. Paper presented at the Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming, Edinburgh, United Kingdom.

Carette, J. (2006). Gaussian elimination: a case study in efficient genericity with MetaOCaml. *Sci. Comput. Program., 62*(1), 3-24.

Czarnecki, K., O’Donnell, J., Striegnitz, J., & Taha, W. (2004). DSL Implementation in MetaOCaml, Template Haskell, and C++. In C. Lengauer, D. Batory, C. Consel, & M. Odersky (Eds.), *Domain-Specific Program Generation* (Vol. 3016, pp. 51-72): Springer Berlin Heidelberg.

de Rauglaudre, D. (2003). Camlp4 - Reference Manual.

Frigo, M. (1999). *A Fast Fourier Transform Compiler*. Paper presented at the Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation, Atlanta, Georgia, USA.

Frisch, A. (2013, 16 April 2014). extension\_points.txt. from <http://caml.inria.fr/cgi-bin/viewvc.cgi/ocaml/trunk/experimental/frisch/extension_points.txt>

Futamura, Y. (1999). Partial Evaluation of Computation Process - An Approach to a Compiler-Compiler. *Higher Order Symbol. Comput., 12*(4), 381-391.

Herrmann, C. A., & Langhammer, T. (2006). Combining partial evaluation and staged interpretation in the implementation of domain-specific languages. *Science of Computer Programming, 62*(1), 47-65.

Jonnalagedda, M., Coppey, T., Stucki, S., Rompf, T., & Odersky, M. (2014). *Staged parser combinators for efficient data processing*. Paper presented at the Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, Portland, Oregon, USA.

Kiselyov, O. (2010, 10 January 2015). MetaOCaml -- an OCaml dialect for multi-stage programming. from <http://okmij.org/ftp/ML/MetaOCaml.html>

Kiselyov, O., Swadi, K. N., & Taha, W. (2004). *A methodology for generating verified combinatorial circuits*. Paper presented at the Proceedings of the 4th ACM international conference on Embedded software, Pisa, Italy.

Langhammer, T. (2005). *Tuning MetaOCaml programs for high performance.* Retrieved from <http://www.infosun.fmi.uni-passau.de/cl/arbeiten/Langhammer.pdf>

Puschel, M. (2011). *Automatic Performance Programming?* Paper presented at the ACM international conference companion on Object oriented programming systems languages and applications companion, Portland, Oregon, USA.

Puschel, M., Moura, J. M. F., Johnson, J. R., Padua, D., Veloso, M. M., Singer, B. W., . . . Rizzolo, N. (2005). SPIRAL: Code Generation for DSP Transforms. *Proceedings of the IEEE, 93*(2), 232-275.

Spampinato, D. G., & Puschel, M. (2014). *A Basic Linear Algebra Compiler*. Paper presented at the Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization, Orlando, FL, USA.

Taha, W. (1999). *Multi-stage programming: Its theory and applications.* Oregon Graduate Institute of Science and Technology.

Taha, W. (2004). A Gentle Introduction to Multi-stage Programming. from <http://www.cs.rice.edu/~taha/publications/journal/dspg04a.pdf>

Whaley, R. C., Petitet, A., & Dongarra, J. J. (2001). Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing, 27*(1--2), 3-35. doi: citeulike-article-id:6904486

Zotov, P. (2014). A Guide to Extension Points in OCaml. from <http://whitequark.org/blog/2014/04/16/a-guide-to-extension-points-in-ocaml/>