Towards a core language with row-based effects for optimised compilation

Axel Faes - Advised by: Tom Schrijvers

KULeuven - Computer Science

Abstract

Algebraic effects and handlers are a very active area of research. An important aspect is the development of an optimising compiler. Eff is an ML-style language with support for effects and forms the testbed for the optimising compiler. However, Eff does not offer explicit typing, which makes it easy for type checking bugs to be introduced during the construction of optimised compilation. This work presents a new core language with row-based effects. The core language is explicitly typed in order to reduce bugs in the optimised compilation.

Introduction

Algebraic effect handling is a very active area of research. Implementations of algebraic effect handlers are becoming available. Because of this, improving performance is becoming the focus of research. A lot of research focusses on speeding up the runtime performance. However, a runtime penalty still occurs. This happens since handlers or continuations need to be repeatedly copied on the heap. Due to this, we are looking towards type-directed optimised compilation of algebraic effect handlers. We want to remove the handlers such that no copying is required and thus no runtime penalty occurs.

In our ongoing research towards type-directed optimised compilation, term rewrite rules and purity aware compilation optimise away most handlers. Term rewrite rules use information of the type-&-effect system. Term rewrite rules perform two types of actions. They remove handlers and apply effects such that eventually the program does not contain any more handlers. Term rewrite rules can also change the syntactic structure in order to expose more possibilities for optimisations. Purity aware compilation identifies computations that are effectively pure and purifies them.

Eff, an ML-style language, is being used to develop an optimised compiler for algebraic effect handlers. Eff uses a type system based on subtyping [1]. As explained by Bauer and Pretnar in [2], terms in Eff do not contain any information about computational effects. This information has to be inferred using type inference algorithms. The lack of explicit type information makes source-to-source transformations much more error-prone. Additionally, ensuring that a transformation does not break typeability becomes a time-consuming task, since we need to reconstruct types after each optimisation pass.

The current type system with subtyping becomes impractical since the typing information is not explicitly contained in each term. There are several solutions to make the type system more practical. It is possible to keep subtyping, but use a unification based algorithm [3]. Implicit effect polymorphism can also be used [7]. The option that is explored in this work, is to use a simple type-&-effect system based on row-polymorphism [6, 5, 4].

In this work, we present a simple explicitly-typed language that can serve as an intermediate language during compilation of Eff, and allows for the development of type-preserving core-to-core transformations. Optimisation and term rewriting is done using this core language. This approach will ease the development of an optimised compiler since typechecking becomes linear due to the explicit typing.

Main Objectives

Background

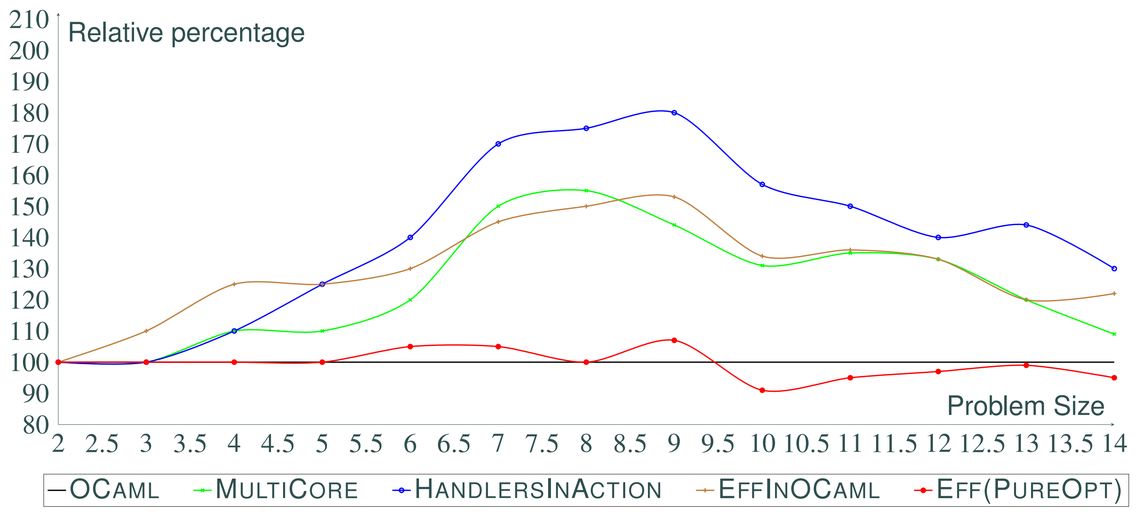
value v	$::=$	x	variable
	$\mid$	$\mathtt{k}$	constant
	$\mid$	$\mathtt{fun}\ x \mapsto c$	function
	$\mid$	$\{$	handler
		$\quad \mathtt{return}\ x \mapsto c_r,$	return case
		$\quad [\mathtt{Op}\ x\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}$	operation cases
		$\}$	
comp c	$::=$	$v_1\ v_2$	application
	$\mid$	$\mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_2$	rec definition
	$\mid$	$\mathtt{return}\ v$	returned val
	$\mid$	$\mathtt{Op}\ v$	operation call
	$\mid$	$\mathtt{do}\ x \leftarrow c_1\ ;\ c_2$	sequencing
	$\mid$	$\mathtt{handle}\ c\ \mathtt{with}\ v$	handling

(pure) type A, B	$::=$	$\mathtt{bool} \mid \mathtt{int}$	basic types
	$\mid$	$A \to \underline{C}$	function type
	$\mid$	$\underline{C} \Rightarrow \underline{D}$	handler type
dirty type $\underline{C}, \underline{D}$	$::=$	$A\ !\ \Delta$	
dirt Δ	$::=$	$\{\mathtt{Op}_1, \ldots, \mathtt{Op}_n\}$	

value v	$::=$	x	variable
	$\mid$	$\mathtt{k}$	constant
	$\mid$	$\lambda(x : A).c$	**function**
	$\mid$	$\Lambda\alpha.c$	**type abstraction**
	$\mid$	$\{$	handler
		$\quad \mathtt{return}\ x \mapsto c_r,$	return case
		$\quad [\mathtt{Op}\ x\ k \mapsto c_{\mathtt{Op}}]_{\mathtt{Op} \in O}$	operation cases
		$\}$	
comp c	$::=$	$v_1\ v_2$	application
	$\mid$	$v\ A$	**type application**
	$\mid$	$\mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_2$	rec definition
	$\mid$	$\mathtt{return}\ v$	returned val
	$\mid$	$\mathtt{Op}\ v$	operation call
	$\mid$	$\mathtt{do}\ x \leftarrow c_1\ ;\ c_2$	sequencing
	$\mid$	$\mathtt{handle}\ c\ \mathtt{with}\ v$	handling

(pure) type A, B	$::=$	$A \to \underline{C}$	function type
	$\mid$	$\underline{C} \Rightarrow \underline{D}$	handler type
	$\mid$	α	**type variable**
	$\mid$	$\forall\alpha.\underline{C}$	**polytype**
dirty type $\underline{C}, \underline{D}$	$::=$	$A\ !\ \Delta$	
dirt Δ	$::=$	$\{\mathtt{Op}_1, \ldots, \mathtt{Op}_n\}$	

Results



Conclusions

Forthcoming Research

References

[1] Andrej Bauer and Matija Pretnar. An effect system for algebraic effects and handlers. *Logical Methods in Computer Science*, 10(4), 2014.

[2] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, 84(1):108–123, 2015.

[3] Stephen Dolan and Alan Mycroft. Polymorphism, subtyping, and type inference in mlsub. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 60–72, New York, NY, USA, 2017. ACM.

[4] Daniel Hillerström and Sam Lindley. Liberating effects with rows and handlers. In *Proceedings of the 1st International Workshop on Type-Driven Development*, TyDe 2016, pages 15–27, New York, NY, USA, 2016. ACM.

[5] Daan Leijen. Koka: Programming with row polymorphic effect types. *arXiv preprint arXiv:1406.2061*, 2014.

[6] Daan Leijen. Type directed compilation of row-typed algebraic effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 486–499, New York, NY, USA, 2017. ACM.

[7] Sam Lindley, Conor McBride, and Craig McLaughlin. Do be do be do. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 500–514, New York, NY, USA, 2017. ACM.

[8] Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013.

[9] Matija Pretnar. Inferring algebraic effects. *Logical Methods in Computer Science*, 10(3), 2014.

[10] Matija Pretnar. An introduction to algebraic effects and handlers. invited tutorial paper. *Electronic Notes in Theoretical Computer Science*, 319:19–35, 2015.

[11] Didier Rémy. Theoretical aspects of object-oriented programming. chapter Type Inference for Records in Natural Extension of ML, pages 67–95. MIT Press, Cambridge, MA, USA, 1994.

Acknowledgements

I would like to thank Amr Hany Saleh for his continuous guidance and help. I would also like to thank Matija Pretnar for his support during my research.