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

Author: Axel Faes
Advisor: Tom Schrijvers

KU LEUVEN

Introduction

Algebraic effect handlers
A feature for side effects and exception handlers on steroids [1, 2]

Implementations have runtime penalty
- ➔ handlers or continuations need to be repeatedly copied [3]
- ➔ Evaluation of effects

Elaborate into representation without algebraic effect handlers

Work in **Eff programming language**

```
effect Decide : unit -> bool;;

let choose_all = handler
  | #Decide () k -> k true @ k false
  | val x -> [x];;

with choose_all handle
  let x =
    (if #Decide () then 10 else 20)
  in
  let y =
    (if #Decide () then 0 else 5)
  in
  x - y

(* Output: [10; 5; 20; 15] *)
```

Background: Optimisations

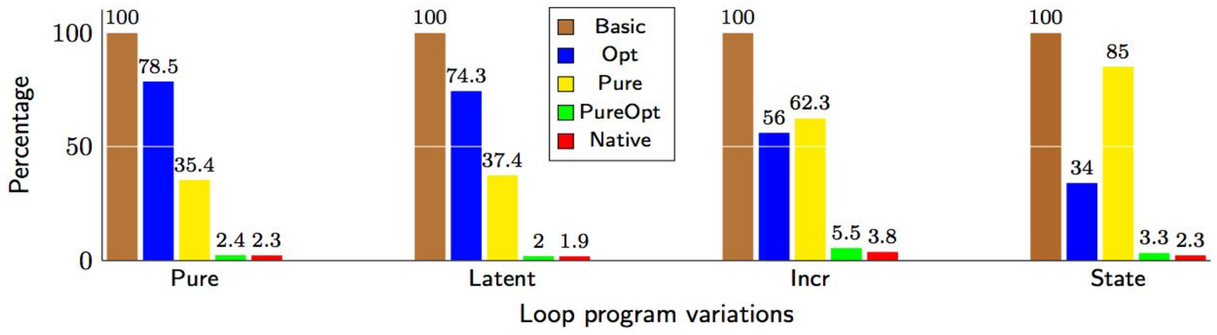
Term rewrite rules
1. remove handlers / apply effects
2. expose optimisations

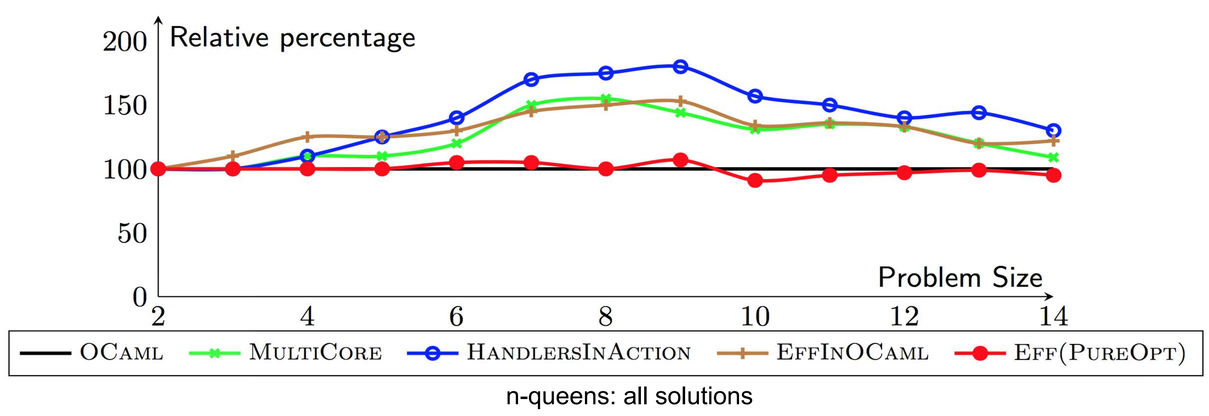
Purity aware compilation
Identify computations that are pure

- ➔ Free monad representation
- ➔ Regular OCaml code

Preliminary results
Eff compared to regular OCaml, Multicore Ocaml, HandlersInAction [4] and EffDirectlyInOCaml [5]

Need to add optimisations for **edge cases**





Research Problem

Compilation of effect handlers
Terms in Eff do not contain explicit type information [3]

Source-to-source transformations are error prone

Ensuring transformations do not break typability is time consuming

Example
function specialisation of handle (let rec)

x specialisation => expose optimisations
- ➔ Making copy of *x* and bring handler inside body

The optimisation needs to correctly handle types of the copy of *x*

Source language

```
effect Op : unit -> int;;

let rec x () = #Op ();;

let result =
  handle (x ()) with
    | #Op () k -> k 1
```

After function specialisation

```
effect Op : unit -> int;;

let rec x () = #Op ();;

let result =
  let rec x_spec () =
    handle (#Op ()) with
      | #Op () k -> k 1
  in x_spec ()
```

New Core Language

Based on calculus from Links [7]
- ➔ row polymorphic type-&-effect system

Other row-based systems
- ➔ Koka [6, 8]
- ➔ PureScript (with Eff monad) [9]

The **row-based effects** are based on row polymorphism and natural fit for effects.

Explicitly typed core calculus and **row-based effects** make the source-to-source transformations less error-prone.

Terms of core calculus

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

Types of core calculus

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

Ongoing Work

Implementation
Integration into the Eff programming language.

Metatheory
The metatheory is under development.

Other solutions
An unification based algorithm for subtyping based type-&-effect system. [10]

Summary

Algebraic effects and handlers are a very active area of research. An important aspect is the development of an optimising compiler.

Without a type-&-effect system with explicit typing, it is easy for *type checking bugs* to be introduced during the construction of optimised compilation.

A core language with *row-based effects* was introduced. The core language is *explicitly typed* in order to reduce bugs in the *optimised compilation*.

References

[1] Andrej Bauer and Matija Pretnar. 2015. Programming with algebraic effects and handlers. Journal of Logical and Algebraic Methods in Programming. 84, 1 (2015), 108–123. https://doi.org/10.1016/j.jlamp.2014.02.00

[2] Andrej Bauer and Matija Pretnar. 2014. An Effect System for Algebraic Effects and Handlers. Logical Methods in Computer Science 10, 4 (2014). https://doi.org/10.2168/LMCS-10(4:9)2014

[3] Gordon D. Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. Logical Methods in Computer Science 9, 4 (2013). https://doi.org/10.2168/LMCS-9(4:23)2013

[4] Ohad Kammar, Sam Lindley, and Nicolas Oury. 2013. Handlers in Action. SIGPLAN Not. 48.9 (Sept. 2013), pp. 145–158. issn: 0362-1340. doi: 10.1145/2544174.2500590. url: http://doi.acm.org/10.1145/2544174.2500590.

[5] Kiselyov, Oleg, and K. C. Sivaramakrishnan. 2016. Eff directly in OCaml. ML Workshop.

[6] Daan Leijen. 2014. Koka: Programming with row polymorphic effect types. arXiv preprint arXiv:1406.2061 (2014).

[7] Daniel Hillerström and Sam Lindley. 2016. Liberating Effects with Rows and Handlers. In Proceedings of the 1st International Workshop on Type-Driven Development (TyDe 2016). ACM, New York, NY, USA, 15–27. https://doi.org/10.1145/2976022.2976033

[8] Daan Leijen. 2017. Type Directed Compilation of Row-typed Algebraic Effects. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017). ACM, New York, NY, USA, 486–499. https://doi.org/10.1145/3009837.3009872

[9] Phil Freeman. 2017. PureScript: Handling Native Effects with the Eff Monad. https://github.com/purescript/documentation/blob/master/guides/Eff.md

[10] .Stephen Dolan and Alan Mycroft . 2017. Polymorphism, Subtyping, and Type Inference in MLsub. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017). ACM, New York, NY, USA, 60–72. https://doi.org/10.1145/3009837.3009882

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.