A core language with row-based effects for optimised compilation

Axel Faes
student : undergraduate
ACM Student Member: 2461936
Department of Computer Science
KU Leuven
axel.faes@student.kuleuven.be

Tom Schrijvers
advisor
Department of Computer Science
KU Leuven
tom.schrijvers@kuleuven.be

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.

Keywords algebraic effect handler, row based effect, optimised compilation

1 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 [?]. As explained by Bauer and Pretnar in [?], 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 [?]. Implicit effect polymorphism can also be used [?]. The option that is explored in this work, is to use a simple type-&-effect system based on row-polymorphism [? ? ?].

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.

Todo: showing an example of a bug that can be manifested at an intermediate stage of a compilation and would be caught earlier if only Eff had explicit type annotations.

2 Background (Eff)

The type-&-effect system that is used in Eff is based on subtyping and dirty types [?].

2.1 Types and terms

Terms Figure 1 shows the two types of terms in Eff. There are values v and computations c. Computations are terms that can contain effects. Effects are denoted as operations Op which can be called.

Types Figure 2 shows the types of Eff. There are two main sorts of types. There are (pure) types A, B and dirty types

$\underline{C}, \underline{D}$. A dirty type is a pure type A tagged with a finite set of operations Δ, which we call dirt, that can be called. This finite set Δ is an over-approximation of the operations that are actually called. The type $\underline{C} \Rightarrow \underline{D}$ is used for handlers because a handler takes an input computation $\underline{C}$, handles the effects in this computation and outputs computation $\underline{D}$ as the result.

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

Figure 1. Terms of Eff

2.2 Type System

2.2.1 Subtyping

TODO: subtyping

2.2.2 Typing rules

Figure 4 defines the typing judgements for values and computations with respect to a standard typing context Γ.

TODO: values and terms

3 Core language

The core language with row-based effects is based on the explicitly typed language used in Links [?]. Links uses a row polymorphic type-&-effect system . The design of their calculus is partially based on the type system used by Pretnar which makes it a suitable candidate for our core language

(pure) type A, B	$::=$	$\mathsf{bool} \mid \mathsf{int}$	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	$::=$	$\{\mathsf{Op}_1, \ldots, \mathsf{Op}_n\}$	

Figure 2. Types of Eff

Subtyping

$$\textsc{Sub-bool}\ \frac{}{\mathsf{bool} \leqslant \mathsf{bool}} \qquad \textsc{Sub-int}\ \frac{}{\mathsf{int} \leqslant \mathsf{int}} \qquad \textsc{Sub-}{\to}\ \frac{A' \leqslant A \qquad \underline{C} \leqslant \underline{C}'}{A \to \underline{C} \leqslant A' \to \underline{C}'}$$

$$\textsc{Sub-}{\Rightarrow}\ \frac{\underline{C}' \leqslant \underline{C} \qquad \underline{D} \leqslant \underline{D}'}{\underline{C} \Rightarrow \underline{D} \leqslant \underline{C}' \Rightarrow \underline{D}'} \qquad \textsc{Sub-!}\ \frac{A \leqslant A' \qquad \Delta \subseteq \Delta'}{A \,!\, \Delta \leqslant A' \,!\, \Delta'}$$

Figure 3. Subtyping for pure and dirty types of Eff

[?]. The terms of the core language are seen in Figure 5, the types are seen in the Figure 6.

3.1 Types and terms

TODO: types and terms

3.2 Typing rules

TODO: typing rules

3.3 Big-Step Operational Semantics

TODO: operational semantics

4 Elaboration

TODO: elaboration

5 Proofs

TODO: proofs

6 Conclusion

TODO: conclusion

Acknowledgments

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.

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ \mid \ \Gamma, x : A$$

Expressions

$$\textsc{SubVal}\ \frac{\Gamma \vdash v : A \qquad A \leqslant A'}{\Gamma \vdash v : A'} \qquad \textsc{Var}\ \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{Const}\ \frac{(\mathsf{k} : A) \in \Sigma}{\Gamma \vdash \mathsf{k} : A}$$

$$\textsc{Fun}\ \frac{\Gamma, x : A \vdash c : \underline{C}}{\Gamma \vdash \mathsf{fun}\ x \mapsto c : A \to \underline{C}}$$

$$\textsc{Hand}\ \frac{\Gamma, x : A \vdash c_r : B \,!\, \Delta \qquad \Big[(\mathsf{Op} : A_{\mathsf{Op}} \to B_{\mathsf{Op}}) \in \Sigma \quad \Gamma, x : A_{\mathsf{Op}}, k : B_{\mathsf{Op}} \to B \,!\, \Delta \vdash c_{\mathsf{Op}} : B \,!\, \Delta\Big]_{\mathsf{Op} \in O}}{\Gamma \vdash \{\mathsf{return}\ x \mapsto c_r, [\mathsf{Op}\ y\ k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\} : A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta}$$

Computations

$$\textsc{SubComp}\ \frac{\Gamma \vdash c : \underline{C} \qquad \underline{C} \leqslant \underline{C}'}{\Gamma \vdash c : \underline{C}'} \qquad \textsc{App}\ \frac{\Gamma \vdash v_1 : A \to \underline{C} \qquad \Gamma \vdash v_2 : A}{\Gamma \vdash v_1\, v_2 : \underline{C}}$$

$$\textsc{LetRec}\ \frac{\Gamma, f : A \to \underline{C}, x : A \vdash c_1 : \underline{C} \qquad \Gamma, f : A \to \underline{C} \vdash c_2 : \underline{D}}{\Gamma \vdash \mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_2 : \underline{D}}$$

$$\textsc{Ret}\ \frac{\Gamma \vdash v : A}{\Gamma \vdash \mathsf{return}\ v : A \,!\, \emptyset} \qquad \textsc{Op}\ \frac{(\mathsf{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A}{\Gamma \vdash \mathsf{Op}\, v : B \,!\, \{\mathsf{Op}\}}$$

$$\textsc{Do}\ \frac{\Gamma \vdash c_1 : A \,!\, \Delta \qquad \Gamma, x : A \vdash c_2 : B \,!\, \Delta}{\Gamma \vdash \mathsf{do}\ x \leftarrow c_1 \,;\, c_2 : B \,!\, \Delta}$$

$$\textsc{With}\ \frac{\Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \qquad \Gamma \vdash c : \underline{C}}{\Gamma \vdash \mathsf{handle}\ c\ \mathtt{with}\ v : \underline{D}}$$

Figure 4. Typing of Eff

value v	$::=$	x	variable
	$\mid$	k	constant
	$\mid$	$\lambda(x : A).c$	**function**
	$\mid$	$\Lambda\alpha.c$	**type abstraction**
	$\mid$	$\{$	handler
		$\mathtt{return}\ x \mapsto c_r,$	return case
		$[\mathsf{Op}\ y\ k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}$	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
	$\mid$	$\mathsf{Op}\, v$	operation call
	$\mid$	$\mathsf{do}\ x \leftarrow c_1 \,;\, c_2$	sequencing
	$\mid$	$\mathtt{handle}\ c\ \mathtt{with}\ v$	handling

Figure 5. Terms of the explicitly typed core language

(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	$::=$	$\{\mathsf{R}\}$	
R	$::=$	$\mathsf{Op} \,;\, R$	row
	$\mid$	δ	row variable
	$\mid$	$.$	end of row

Figure 6. Types of the explicitly type core language

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ \mid \ \Gamma, x : A$$

Expressions

$$\textsc{Val}\quad \frac{}{\Gamma \vdash v : A} \qquad \textsc{Var}\quad \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{Const}\quad \frac{(\mathsf{k} : A) \in \Sigma}{\Gamma \vdash \mathsf{k} : A}$$

$$\textsc{Fun}\quad \frac{\Gamma, x : A \vdash c : \underline{C}}{\Gamma \vdash \lambda(x : A).c : A \to \underline{C}} \qquad \textsc{Type Abstraction}\quad \frac{\Gamma, \alpha \vdash c : \underline{C}}{\Gamma \vdash \Lambda\alpha.c : \forall\alpha.\underline{C}}$$

$$\textsc{Hand}\quad \frac{\begin{array}{c} \underline{C} = A \,!\, \{Op_i \,;\, R\} \\ \underline{D} = B \,!\, \{Op_i \,;\, R\} \qquad (\mathsf{Op}_i : A_{\mathsf{Op}} \to B_{\mathsf{Op}}) \in \Sigma \\ h = \{\mathtt{return}\ x \mapsto c_r, [\mathsf{Op}\, y\, k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\} \\ \Gamma, x : A_{\mathsf{Op}} \vdash c_r : \underline{D} \\ \Gamma, y : A_{\mathsf{Op}}, k : B_{\mathsf{Op}} \to \underline{D} \vdash c_{op} : \underline{D} \end{array}}{\Gamma \vdash h : \underline{C} \Rightarrow \underline{D}}$$

Computations

$$\textsc{Comp}\quad \frac{}{\Gamma \vdash c : \underline{C}} \qquad \textsc{App}\quad \frac{\Gamma \vdash v_1 : A \to \underline{C} \qquad \Gamma \vdash v_2 : A}{\Gamma \vdash v_1\, v_2 : \underline{C}}$$

$$\textsc{Type App}\quad \frac{\Gamma \vdash v : \forall\alpha.\underline{C}}{\Gamma \vdash v\, A : \underline{C}[A/\alpha]}$$

$$\textsc{LetRec}\quad \frac{\Gamma, f : A \to \underline{C}, x : A \vdash c_1 : \underline{C} \qquad \Gamma, f : A \to \underline{C} \vdash c_2 : \underline{D}}{\Gamma \vdash \mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_2 : \underline{D}}$$

$$\textsc{Ret}\quad \frac{\Gamma \vdash v : A}{\Gamma \vdash \mathtt{return}\ v : A \,!\, \emptyset}$$

$$\textsc{Op}\quad \frac{(\mathsf{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A \qquad \underline{C} : B \,!\, \{\mathsf{Op} \,;\, R\}}{\Gamma \vdash \mathsf{Op}\, v : \underline{C}}$$

$$\textsc{Do}\quad \frac{\Gamma \vdash c_1 : A \,!\, \Delta \qquad \Gamma, x : A \vdash c_2 : B \,!\, \Delta}{\Gamma \vdash \mathtt{do}\ x \leftarrow c_1 \,;\, c_2 : B \,!\, \Delta}$$

$$\textsc{With}\quad \frac{\Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \qquad \Gamma \vdash c : \underline{C}}{\Gamma \vdash \mathtt{handle}\ c\ \mathtt{with}\ v : \underline{D}}$$

Figure 7. Typing of the explicitly typed language

$$\text{result } r \ ::= \ \mathtt{return}\ v \ \mid \ \mathsf{Op}\, v\, (y.\, c)$$

Evaluation

$$\textsc{Eval-App}\quad \frac{c[v/x] \Downarrow r}{(\mathtt{fun}\ x \mapsto c)\, v \Downarrow r}$$

$$\textsc{Eval-LetRec}\quad \frac{c_2[(\mathtt{fun}\ x \mapsto \mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_1)/f] \Downarrow r}{\mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_2 \Downarrow r}$$

$$\textsc{Eval-Ret}\quad \frac{}{\mathtt{return}\ v \Downarrow \mathtt{return}\ v} \qquad \textsc{Eval-Op}\quad \frac{}{\mathsf{Op}\, v \Downarrow \mathsf{Op}\, v\, (y.\, \mathtt{return}\ y)}$$

$$\textsc{Eval-Do-Ret}\quad \frac{c_1 \Downarrow \mathtt{return}\ v \qquad c_2[v/x] \Downarrow r}{\mathtt{do}\ x \leftarrow c_1 \,;\, c_2 \Downarrow r}$$

$$\textsc{Eval-Do-Op}\quad \frac{c_1 \Downarrow \mathsf{Op}\, v\, (y.\, c_1')}{\mathtt{do}\ x \leftarrow c_1 \,;\, c_2 \Downarrow \mathsf{Op}\, v\, (y.\, \mathtt{do}\ x \leftarrow c_1' \,;\, c_2)}$$

$$\textsc{Eval-With-Ret}\quad \frac{c \Downarrow \mathtt{return}\ v \qquad c_r[v/x] \Downarrow r}{\mathtt{handle}\ c\ \mathtt{with}\ h \Downarrow r}$$

$$\textsc{Eval-With-Handled-Op}\quad \frac{\begin{array}{c} c \Downarrow \mathsf{Op}\, v\, (y.\, c') \\ c_{\mathsf{Op}}[v/x, (\mathtt{fun}\ y \mapsto \mathtt{handle}\ c'\ \mathtt{with}\ h)/k] \Downarrow r \end{array}}{\mathtt{handle}\ c\ \mathtt{with}\ h \Downarrow r}$$

$$\textsc{Eval-With-Unhandled-Op}\quad \frac{c \Downarrow \mathsf{Op}'\, v\, (y.\, c') \qquad \mathsf{Op}' \notin O}{\mathtt{handle}\ c\ \mathtt{with}\ h \Downarrow \mathsf{Op}'\, v\, (y.\, \mathtt{handle}\ c'\ \mathtt{with}\ h)}$$

Figure 8. Operational semantics (in the last three rules, $h = \{\mathtt{return}\ x \mapsto c_r, [\mathsf{Op}\, y\, k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\}$)

$$\text{typing contexts } \Gamma \ ::= \ \ \epsilon \ \mid \ \Gamma, x : A$$

Expressions

$$\textsc{Val}\quad \frac{}{\Gamma \vdash v : A \rightsquigarrow v'} \qquad \textsc{Var}\quad \frac{(x : S) \in \Gamma \qquad S = \forall \bar{\alpha}.A}{\Gamma \vdash x : A[\bar{S}/\bar{\alpha}] \rightsquigarrow x\,\bar{S}'}$$

$$\textsc{Const}\quad \frac{(\mathsf{k} : A) \in \Sigma}{\Gamma \vdash \mathsf{k} : A \rightsquigarrow \mathsf{k}'}$$

$$\textsc{Fun}\quad \frac{\Gamma, x : A \vdash c : \underline{C} \rightsquigarrow c'}{\Gamma \vdash \mathsf{fun}\ x \mapsto c : A \to \underline{C} \rightsquigarrow \lambda(x : A).c' : A \to \underline{C}}$$

$$\textsc{Hand}\quad \frac{\Gamma, x : A \vdash c_r : B \,!\, \Delta \qquad \Big[(\mathsf{Op} : A_{\mathsf{Op}} \to B_{\mathsf{Op}}) \in \Sigma \quad \Gamma, x : A_{\mathsf{Op}}, k : B_{\mathsf{Op}} \to B \,!\, \Delta \vdash c_{\mathsf{Op}} : B \,!\, \Delta\Big]_{\mathsf{Op} \in O}}{\Gamma \vdash \{\mathsf{return}\ x \mapsto c_r, [\mathsf{Op}\, y\, k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\} : A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta}$$

$$\textsc{Hand}\quad \frac{\begin{array}{c} \underline{C} = A \,!\, \{Op_i \,;\, R\} \\ \underline{D} = B \,!\, \{Op_i \,;\, R\} \qquad (\mathsf{Op}_i : A_{\mathsf{Op}} \to B_{\mathsf{Op}}) \in \Sigma \\ h = \{\mathsf{return}\ x \mapsto c_r, [\mathsf{Op}\, y\, k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\} \\ \rightsquigarrow h' = \{\mathsf{return}\ x \mapsto c_r', [\mathsf{Op}\, y\, k \mapsto c_{\mathsf{Op}}']_{\mathsf{Op} \in O}\} \\ \Gamma, x : A_{\mathsf{Op}} \vdash c_r : \underline{D} \rightsquigarrow c_r' : \underline{D} \\ \Gamma, y : A_{\mathsf{Op}}, k : B_{\mathsf{Op}} \to \underline{D} \vdash c_{op} : \underline{D} \rightsquigarrow c_{op}' : \underline{D} \end{array}}{\Gamma \vdash h : \underline{C} \Rightarrow \underline{D} \rightsquigarrow h' : \underline{C} \Rightarrow \underline{D}}$$

Computations

$$\textsc{Comp}\quad \frac{}{\Gamma \vdash c : \underline{C}' \rightsquigarrow c'}$$

$$\textsc{App}\quad \frac{\Gamma \vdash v_1 : A \to \underline{C} \rightsquigarrow v_1' \qquad \Gamma \vdash v_2 : A \rightsquigarrow v_2'}{\Gamma \vdash v_1\, v_2 : \underline{C} \rightsquigarrow v_1'\, v_2' : \underline{C}}$$

$$\textsc{LetRec}\quad \frac{\begin{array}{c} \Gamma, f : A \to \underline{C}, x : A \vdash c_1 : \underline{C} \rightsquigarrow c_1's \\ \Gamma, f : A \to \underline{C} \vdash c_2 : \underline{D} \rightsquigarrow c_2' \end{array}}{\begin{array}{c} \Gamma \vdash \mathtt{let\ rec}\ f\, x = c_1\ \mathtt{in}\ c_2 : \underline{D} \\ \rightsquigarrow \mathtt{let\ rec}\ f\, x = c_1'\ \mathtt{in}\ c_2' : \underline{D} \end{array}}$$

$$\textsc{Ret}\quad \frac{\Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathsf{return}\ v : A \,!\, \emptyset \rightsquigarrow \mathsf{return}\ v' : A \,!\, \emptyset}$$

$$\textsc{Op}\quad \frac{\begin{array}{c} (\mathsf{Op} : A \to B) \in \Sigma \\ \underline{C} = B \,!\, \{\mathsf{Op} \,;\, R\} \qquad \Gamma \vdash v : A \rightsquigarrow v' \end{array}}{\Gamma \vdash \mathsf{Op}\, v : \underline{C} \rightsquigarrow \mathsf{Op}\, v' : \underline{C}}$$

$$\textsc{Do}\quad \frac{\begin{array}{c} \Gamma \vdash c_1 : \underline{C} \rightsquigarrow c_1' \qquad S = \forall \bar{\alpha}.A \\ \bar{\alpha} = FTV(A) - TV(\Gamma) \qquad \Gamma, x : S \vdash \underline{D} \rightsquigarrow c_2' \end{array}}{\Gamma \vdash \mathsf{do}\ x \leftarrow c_1 \,;\, c_2 : \underline{D} \rightsquigarrow (\lambda(x : A).c_2')(\Lambda \bar{\alpha}.c_1')}$$

$\textsc{With}$