Algebraic subtyping for algebraic effects and handlers

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

Amr Hany Saleh
daily advisor
Department of Computer Science
KU Leuven
amrhanyshehata.saleh@kuleuven.be

Tom Schrijvers
promotor
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, the type-&-effect system of Eff is unsatisfactory. This is due to the lack of some elegant properties. It is also awkward to implement and use in practice.

Keywords algebraic effect handler, algebraic subtyping, effects, optimised compilation

Contents

List of Figures

List of Tables

1 Introduction

The specification for a type-&-effect system with algebraic subtyping for algebraic effects and handlers is given in this document. The formal properties of this system are studied in order to find which properties are satisfied compared to other type-&-effect systems. The proposed type-&-effect system builds on two very recent developments in the area of programming language theory.

Algebraic subtyping In his December 2016 PhD thesis, Stephen Dolan (University of Cambridge, UK), has presented a novel type system that combines subtyping and parametric polymorphism in a particulary attractive and elegant fashion. A cornerstone of his design are the algebraic properties that the subtyping relation should respect.

Algebraic effects and handlers These are a new formalism for formally modelling side-effects (e.g. mutable state or non-determinism) in programming languages, developed by Matija Pretnar (University of Ljubjana) and Gordon Plotkin (University of Edinburgh). This approach is gaining a lot of traction, not only as a formalism but also as a practical feature in actual programming languages (e.g. the Koka language developed by Microsoft Research). We are collaborating with Matija Pretnar on the efficient implementation of one such language, called Eff. Axel Faes has contributed to this collaboration during a project he did for the Honoursprogramme of the Faculty of Engineering Science.

1.1 Motivation

Algebraic effects and handlers benefit from a custom type-&-effect system, a type system that also tracks which effects can happen in a program. Several such type-&-effect systems have been proposed in the literature, but all are unsatisfactory. We attribute this to the lack of the elegant properties of Dolan's type system. Indeed the existing type-&-effect systems are not only theoretically unsatisfactory, but they are also awkward to implement and use in practice.

Research questions

- How can Dolan's elegant type system be extended with effect information?
- Which properties are preserved and which aren't preserved?
- What advantages are there to an type-&-effect system based on Dolan's elegant type system?

1.2 Goals

The goal of this thesis is to derive a type-&-effect system that extends Dolan's elegant type system with effect information. This type-&-effect system should inherit Dolan's harmonious combination of subtyping (in our case induced by a lattice structure on the effect information) with parametric polymorphism and preserve all of its desirable properties (both low-level algebraic properties and high-level meta-theoretical properties like type soundness and the existence of principal types). Afterwards this type-&-effect system The following approach is taken:

1. Study of the relevant literature and theoretical background.
2. Design of a type-&-effect system derived from Dolan's, that integrates effects.
3. Proving the desirable properties of the proposed type-&-effect system: type soundness, principal typing, ...
4. Time permitting: Design of a type inference algorithm that derives the principal types of programs without type annotations and proving its correctness.
5. Time permitting: Implementation of the algorithm and comparing it to other algorithms (such as row polymorphism based type-&-effect systems).

2 Background

3 Related Work (Eff)

The type-&-effect system that is used in Eff is based on subtyping and dirty types [1].

3.1 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.

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

Figure 1. Terms of Eff

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.

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

Figure 2. Types of Eff

3.2 Type System

3.2.1 Subtyping

The dirty type $A\ !\ \Delta$ is assigned to a computation returning values of type A and potentially calling operations from the set Δ. This set Δ is always an over-approximation of the actually called operations, and may safely be increased, inducing a natural subtyping judgement $A \,!\, \Delta \leq A \,!\, \Delta'$ on dirty types. As dirty types can occur inside pure types, we also get a derived subtyping judgement on pure types. Both judgements are defined in Figure 3. Observe that, as usual, subtyping is contravariant in the argument types of functions and handlers, and covariant in their return types.

Subtyping

Sub-bool
$$\frac{}{\texttt{bool} \leqslant \texttt{bool}}$$

Sub-int
$$\frac{}{\texttt{int} \leqslant \texttt{int}}$$

Sub-$\rightarrow$
$$\frac{A' \leqslant A \qquad \underline{C} \leqslant \underline{C}'}{A \rightarrow \underline{C} \leqslant A' \rightarrow \underline{C}'}$$

Sub-$\Rightarrow$
$$\frac{\underline{C}' \leqslant \underline{C} \qquad \underline{D} \leqslant \underline{D}'}{\underline{C} \Rightarrow \underline{D} \leqslant \underline{C}' \Rightarrow \underline{D}'}$$

Sub-!
$$\frac{A \leqslant A' \qquad \Delta \subseteq \Delta'}{A \,!\, \Delta \leqslant A' \,!\, \Delta'}$$

Figure 3. Subtyping for pure and dirty types of Eff

3.2.2 Typing rules

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

Values The rules for subtyping, variables, and functions are entirely standard. For constants we assume a signature Σ that assigns a type A to each constant k, which we write as $(\mathsf{k} : A) \in \Sigma$.

A handler expression has type $A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta$ iff all branches (both the operation cases and the return case) have dirty type $B \,!\, \Delta$ and the operation cases cover the set of operations O. Note that the intersection $\Delta \cap O$ is not necessarily empty. The handler deals with the operations O, but in the process may re-issue some of them (i.e., $\Delta \cap O$).

When typing operation cases, the given signature for the operation $(\mathsf{Op} : A_{\mathsf{Op}} \rightarrow B_{\mathsf{Op}}) \in \Sigma$ determines the type A_{Op} of the parameter x and the domain B_{Op} of the continuation k. As our handlers are deep, the codomain of k should be the same as the type $B \,!\, \Delta$ of the cases.

Computations With the following exceptions, the typing judgement $\Gamma \vdash c : \underline{C}$ has a straightforward definition. The `return` construct renders a value v as a pure computation, i.e., with empty dirt. An operation invocation $\mathsf{Op}\, v$ is typed according to the operation's signature, with the operation itself as its only operation. Finally, rule With shows that a handler with type $\underline{C} \Rightarrow \underline{D}$ transforms a computation with type $\underline{C}$ into a computation with type $\underline{D}$.

typing contexts $\Gamma \ ::= \ \epsilon \ | \ \Gamma, x : A$

Expressions

SubVal
$$\frac{\Gamma \vdash v : A \qquad A \leqslant A'}{\Gamma \vdash v : A'}$$

Var
$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$$

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

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

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

Computations

SubComp
$$\frac{\Gamma \vdash c : \underline{C} \qquad \underline{C} \leqslant \underline{C}'}{\Gamma \vdash c : \underline{C}'}$$

App
$$\frac{\Gamma \vdash v_1 : A \rightarrow \underline{C} \qquad \Gamma \vdash v_2 : A}{\Gamma \vdash v_1\, v_2 : \underline{C}}$$

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

Ret
$$\frac{\Gamma \vdash v : A}{\Gamma \vdash \texttt{return}\ v : A \,!\, \emptyset}$$

Op
$$\frac{(\mathsf{Op} : A \rightarrow B) \in \Sigma \qquad \Gamma \vdash v : A}{\Gamma \vdash \mathsf{Op}\, v : B \,!\, \{\mathsf{Op}\}}$$

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

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

Figure 4. Typing of Eff

4 Core language (EffCore)

EffCore is a language with row-based effects, intersection and union types and effects and is subtyping based.

4.1 Types and terms

Terms Figure 5 shows the two types of terms in EffCore. There are values v and computations c. Computations are terms that can contain effects. Effects are denoted as operations *Op* which can be called. The function term is explicitly annotated with a type and type abstraction and type application has been added to the language. These terms only work on pure types.

value v	::=	x	variable
	|	k	constant
	|	`fun` $x : A \mapsto c$	function
	|	$\Lambda\alpha.v$	type abstraction
	|	$v\,A$	type application
	|	{	handler
		`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
	|	`if` e `then` c_1 `else` c_2	conditional
	|	`let rec` $f\,x = c_1$ `in` c_2	rec definition
	|	`return` v	returned val
	|	Op v	operation call
	|	do $x \leftarrow c_1$; c_2	sequencing
	|	`handle` c `with` v	handling

Figure 5. Terms of EffCore

(pure) type A, B	::=	`bool` | `int`	basic types
	|	$A \rightarrow \underline{C}$	function type
	|	$\underline{C} \Rightarrow \underline{D}$	handler type
	|	α	type variable
	|	$\forall\alpha.A$	polytype
	|	$\top$	top
	|	$\bot$	bottom
	|	$A \sqcap B$	intersection
	|	$A \sqcup B$	union
dirty type $\underline{C}, \underline{D}$	::=	$A \mathrel{!} \Delta$	
	|	$\underline{C} \sqcap \underline{D}$	intersection
	|	$\underline{C} \sqcup \underline{D}$	union
dirt Δ	::=	{R}	
R	::=	Op ; R	row
	|	δ	row variable
	|	.	closed row
	|	$R_1 \sqcap R_2$	intersection
	|	$R_1 \sqcup R_2$	union
All operations Ω	::=	$\{\mathsf{Op}_i \mid \mathsf{Op}_i \in \Sigma\}$	

Figure 6. Types of EffCore

Types Figure 6 shows the types of EffCore. 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. It can also be an union or intersection of dirty types. In further sections, the relations between dirty intersections or unions and pure intersections or unions are explained. The finite set Δ is an over-approximation of the operations that are actually called. Row variables are introduced as well as intersection and unions. The .(dot) is used to close rows that do not end with a row variable. 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.

4.2 Type system

4.3 Subtyping rules

The subtyping rules are given in Figure 7, Figure 8 and Figure 9. Figure 7 contains all subtyping rules related to types. Figure 8 contains the distributative subtyping rules. Finally Figure 9 contains the subtyping rules that govern the dirts.

The dirty type $A \mathrel{!} \Delta$ is assigned to a computation returning values of type A and potentially calling operations from the set Δ. This set Δ is always an over-approximation of the actually called operations, and may safely be increased, inducing a natural subtyping judgement $A \mathrel{!} \Delta \le A \mathrel{!} \Delta'$ on dirty types. As dirty types can occur inside pure types, we also get a derived subtyping judgement on pure types. Observe that, as usual, subtyping is contravariant in the argument types of functions and handlers, and covariant in their return types.

Dirt intersection and union There are several possible methods to compute the dirt intersection and union. If row variables were to be disregarded, dirt union and intersection could be defined as set union and intersection. This methods allows unions and intersections to be eliminated. This has an advantage, eliminating unions and intersections simplifies the effect system. However, we cannot disregard row variables.

Thus, set union and intersection cannot simply be used. It would be possible to define $\delta_1 \sqcup \delta_2$ and $\delta_1 \sqcap \delta_2$. Using these, it is possible to use a form of set union and intersection. The following union $\{Op_1, ..., Op_n, \delta_1\} \sqcup \{Op_{n+1}, ..., Op_{n+m}, \delta_2\}$ could be defined as $\{Op_1, ..., Op_n, Op_{n+1}, ..., Op_{n+m}, (\delta_1 \sqcup \delta_2)\}$. A similar construction can be used for intersection. This simplifies the subtyping rules since the more complicated aspects are enclosed within the row variables. The equivalence rules are defined in Figure 10.

4.4 Equivalence rules

4.5 Typing rules

Figure 11 defines the typing judgements for values and computations with respect to a standard typing context Γ.

Values The rules for subtyping, variables, type abstraction, type application and functions are entirely standard. For constants we assume a signature Σ that assigns a type A to each constant k, which we write as $(\mathsf{k} : A) \in \Sigma$.

A handler expression has type $A \mathrel{!} \Delta \cup O \Rightarrow B \mathrel{!} \Delta$ iff all branches (both the operation cases and the return case) have dirty type $B \mathrel{!} \Delta$ and the operation cases cover the

Subtyping of pure and dirty types

$$\frac{}{A \leqslant \top}\ \textsc{Sub-Top} \qquad \frac{}{\bot \leqslant A}\ \textsc{Sub-Bottom}$$

$$\frac{}{\texttt{bool} \leqslant \texttt{bool}}\ \textsc{Sub-}\texttt{bool} \qquad \frac{}{\texttt{int} \leqslant \texttt{int}}\ \textsc{Sub-}\texttt{int}$$

$$\frac{}{A \leqslant A}\ \textsc{Sub-Refl} \qquad \frac{A_1 \leqslant A_2 \qquad A_2 \leqslant A_3}{A_1 \leqslant A_3}\ \textsc{Sub-Trans}$$

$$\frac{A_1 \leqslant A_2' \qquad \Delta_1 \leqslant \Delta_2'}{A_1 ~!~ \Delta_1 \leqslant A_2 ~!~ \Delta_2}\ \textsc{Sub-!} \qquad \frac{A \leqslant B}{\forall \alpha.A \leqslant \forall \alpha.B}\ \textsc{Sub-All}$$

$$\frac{A_2 \leqslant A_1 \qquad \underline{C}_1 \leqslant \underline{C}_2}{A_1 \to \underline{C}_1 \leqslant A_2 \to \underline{C}_2}\ \textsc{Sub-}\!\to \qquad \frac{\underline{C}_2 \leqslant \underline{C}_1 \qquad \underline{D}_1 \leqslant \underline{D}_2}{\underline{C}_1 \Rightarrow \underline{D}_1 \leqslant \underline{C}_2 \Rightarrow \underline{D}_2}\ \textsc{Sub-}\!\Rightarrow$$

$$\frac{A_1 \leqslant A_2 \qquad B_1 \leqslant B_2}{A_1 \sqcap B_1 \leqslant A_2 \sqcap B_2}\ \textsc{Sub-Inter} \qquad \frac{A_1 \leqslant A_2 \qquad B_1 \leqslant B_2}{A_1 \sqcup B_1 \leqslant A_2 \sqcup B_2}\ \textsc{Sub-Union}$$

$$\frac{A \leqslant B_1 \qquad A \leqslant B_2}{A \leqslant B_1 \sqcap B_2}\ \textsc{Sub-Inter-Greatest-LB} \qquad \frac{i \in \{1;2\}}{A_1 \sqcap A_2 \leqslant A_i}\ \textsc{Sub-Inter-LB}$$

$$\frac{B_1 \leqslant A \qquad B_2 \leqslant A}{B_1 \sqcup B_2 \leqslant A}\ \textsc{Sub-Union-Least-UB} \qquad \frac{i \in \{1;2\}}{A_i \leqslant A_1 \sqcup A_2}\ \textsc{Sub-Inter-UB}$$

Figure 7. Subtyping for pure and dirty types of EffCore

set of operations O. Note that the intersection $\Delta \cap O$ is not necessarily empty (with $\cap$ being the intersection of the operations, not to be confused with the $\sqcap$ type). The handler deals with the operations O, but in the process may re-issue some of them (i.e., $\Delta \cap O$).

When typing operation cases, the given signature for the operation $(\mathsf{Op} : A_{\mathsf{Op}} \to B_{\mathsf{Op}}) \in \Sigma$ determines the type A_{Op} of the parameter x and the domain B_{Op} of the continuation k. As our handlers are deep, the codomain of k should be the same as the type $B ~!~ \Delta$ of the cases.

Computations With the following exceptions, the typing judgement $\Gamma \vdash c : \underline{C}$ has a straightforward definition. The `return` construct renders a value v as a pure computation, i.e., with empty dirt. In this case, this is defined as a set with the .(DOT) as the only element. An operation invocation $\mathsf{Op}\, v$ is typed according to the operation's signature, with the operation itself as its only operation. Finally, rule With shows

Distributivity Subtyping of pure and dirty types

$$\frac{}{(A \to \underline{C}) \sqcap (A \to \underline{D}) \leqslant A \to (\underline{C} \sqcap \underline{D})}\ \textsc{Sub-Dist-Func-Inter}$$

$$\frac{}{(A \to \underline{C}) \sqcap (B \to \underline{C}) \leqslant (A \sqcup B) \to \underline{C}}\ \textsc{Sub-Dist-Func-Union}$$

$$\frac{}{(A \Rightarrow \underline{C}) \sqcap (A \Rightarrow \underline{D}) \leqslant A \Rightarrow (\underline{C} \sqcap \underline{D})}\ \textsc{Sub-Dist-Hand-Inter}$$

$$\frac{}{(A \Rightarrow \underline{C}) \sqcap (B \Rightarrow \underline{C}) \leqslant (A \sqcup B) \Rightarrow \underline{C}}\ \textsc{Sub-Dist-Hand-Union}$$

$$\frac{}{\forall \alpha.A \sqcap \forall \alpha.B \leqslant \forall \alpha.A \sqcap B}\ \textsc{Sub-Dist-All}$$

Figure 8. Distributivity Subtyping for pure and dirty types of EffCore

that a handler with type $\underline{C} \Rightarrow \underline{D}$ transforms a computation with type $\underline{C}$ into a computation with type $\underline{D}$.

Subtyping of dirts

$$\frac{}{\{.\} \leqslant \Delta}\ \textsc{Sub-!-Empty} \qquad \frac{}{\Delta \leqslant \Omega}\ \textsc{Sub-!-Top} \qquad \frac{\Delta_1 \leqslant \Delta_2 \qquad \Delta_2 \leqslant \Delta_3}{\Delta_1 \leqslant \Delta_3}\ \textsc{Sub-!-Trans}$$

$$\frac{}{\{Op_1, ..., Op_n, .\} \leqslant \{Op_1, ..., Op_n, \delta\}}\ \textsc{Sub-!-Row} \qquad \frac{}{\Delta \leqslant \Delta}\ \textsc{Sub-!-Refl}$$

$$\frac{n \geq 0 \qquad m \geq 0 \qquad p \geq 0 \qquad \{Op_1, ..., Op_n, Op_{n+m+1}, ..., Op_{n+m+p}, \delta_1\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, ..., Op_{n+m}, \delta_2\}}{\{\delta_1\} \leqslant \{Op_{n+1}, ..., Op_{n+m}, \delta_2\} \qquad \{\delta_2\} = \{Op_{n+m}, ..., Op_{n+m+p}, \delta_3\}}\ \textsc{Sub-!-Row-Row}$$

$$\frac{n \geq 0 \qquad m \geq 0 \qquad p \geq 0 \qquad \{Op_1, ..., Op_n, Op_{n+m+1}, ..., Op_{n+m+p}, .\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, ..., Op_{n+m}, \delta_2\}}{\{.\} \leqslant \{Op_{n+1}, ..., Op_{n+m}, \delta_2\} \qquad \{\delta_2\} = \{Op_{n+m}, ..., Op_{n+m+p}, \delta_3\}}\ \textsc{Sub-!-Dot-Row}$$

$$\frac{n \geq 0 \qquad m \geq 0 \qquad \{Op_1, ..., Op_n, \delta_1\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, Op_{n+m}, .\}}{\{\delta_1\} \leqslant \{Op_{n+1}, Op_{n+m}, .\}}\ \textsc{Sub-!-Row-Dot}$$

$$\frac{n \geq 0 \qquad m \geq 0 \qquad \{Op_1, ..., Op_n, .\} \leqslant \{Op_1, ..., Op_n, Op_{n+1}, ..., Op_{n+m}, .\}}{\{.\} \leqslant \{Op_{n+1}, Op_{n+m}, .\}}\ \textsc{Sub-!-Dot-Dot}$$

$$\frac{\Delta_1 \leqslant \Delta_2 \qquad \Delta_3 \leqslant \Delta_4 \qquad \Delta_1 \neq \Delta_3}{\Delta_1 \sqcap \Delta_3 \leqslant \Delta_2 \sqcap \Delta_4}\ \textsc{Sub-Inter-!}$$

$$\frac{\Delta_1 \leqslant \Delta_2 \qquad \Delta_3 \leqslant \Delta_4 \qquad \Delta_1 \neq \Delta_3}{\Delta_1 \sqcup \Delta_3 \leqslant \Delta_2 \sqcup \Delta_4}\ \textsc{Sub-Union-!}$$

$$\frac{\Delta_1 \leqslant \Delta_2 \qquad \Delta_1 \leqslant \Delta_3}{\Delta_1 \leqslant (\Delta_2 \sqcap \Delta_3)}\ \textsc{Sub-Inter-Greatest-LB-!} \qquad \frac{i \in \{1; 2\}}{(\Delta_1 \sqcap \Delta_2) \leqslant \Delta_i}\ \textsc{Sub-Inter-LB-!}$$

$$\frac{\Delta_2 \leqslant \Delta_1 \qquad \Delta_3 \leqslant \Delta_1}{(\Delta_2 \sqcup \Delta_3) \leqslant \Delta_1}\ \textsc{Sub-Union-Least-UB-!} \qquad \frac{i \in \{1; 2\}}{\Delta_i \leqslant (\Delta_1 \sqcup \Delta_2)}\ \textsc{Sub-Union-UB-!}$$

Figure 9. Subtyping for dirts of EffCore

Equivalance rules

Union-!-Row-Row

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \vee Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, (\delta_1 \sqcup \delta_2)\} = \{Op_1, ..., Op_n, \delta_1\} \sqcup \{Op_{n+1}, ..., Op_{n+m}, \delta_2\}$$

Union-!-Row-Dot

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \vee Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, \delta_1\} = \{Op_1, ..., Op_n, \delta_1\} \sqcup \{Op_{n+1}, ..., Op_{n+m}, .\}$$

Union-!-Dot-Row

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \vee Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, \delta_2\} = \{Op_1, ..., Op_n, .\} \sqcup \{Op_{n+1}, ..., Op_{n+m}, \delta_2\}$$

Union-!-Dot-Dot

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \vee Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, .\} = \{Op_1, ..., Op_n, .\} \sqcup \{Op_{n+1}, ..., Op_{n+m}, .\}$$

Intersection-!-Row-Row

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \wedge Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, (\delta_1 \sqcap \delta_2)\} = \{Op_1, ..., Op_n, \delta_1\} \sqcap \{Op_{n+1}, ..., Op_{n+m}, \delta_2\}$$

Intersection-!-Row-Dot

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \wedge Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, \delta_1\} = \{Op_1, ..., Op_n, \delta_1\} \sqcap \{Op_{n+1}, ..., Op_{n+m}, .\}$$

Intersection-!-Dot-Row

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \wedge Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, \delta_2\} = \{Op_1, ..., Op_n, .\} \sqcap \{Op_{n+1}, ..., Op_{n+m}, \delta_2\}$$

Intersection-!-Dot-Dot

$$\{Op_i | Op_i \in \{Op_1, ..., Op_n\} \wedge Op_i \in \{Op_{n+1}, ..., Op_{n+m}\}, .\} = \{Op_1, ..., Op_n, .\} \sqcap \{Op_{n+1}, ..., Op_{n+m}, .\}$$

Figure 10. Equivalance rules

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ \mid \ \Gamma, x : A, x : \forall\alpha.B$$

Expressions

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

$$\textsc{Type Abs}\ \frac{\Gamma, \alpha \vdash v : A}{\Gamma \vdash \Lambda\alpha.v : \forall\alpha.A} \qquad \textsc{Type App}\ \frac{\Gamma \vdash v : \forall\alpha.B}{\Gamma \vdash v\, A : B[A/\alpha]}$$

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

$$\textsc{Hand}\ \frac{\Gamma, x : A \vdash c_r : B \mathbin{!} \Delta \qquad \left[\begin{array}{l} (\mathsf{Op} : A_{\mathsf{Op}} \to B_{\mathsf{Op}}) \in \Sigma \\ \Gamma, x : A_{\mathsf{Op}}, k : B_{\mathsf{Op}} \to B \mathbin{!} \Delta \vdash c_{\mathsf{Op}} : B \mathbin{!} \Delta \end{array} \right]_{\mathsf{Op} \in O}}{\begin{array}{c} \Gamma \vdash \{\mathtt{return}\ x \mapsto c_r, [\mathsf{Op}\, y\, k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\} : \\ A \mathbin{!} \Delta \cup O \Rightarrow B \mathbin{!} \Delta \end{array}}$$

Computations

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

$$\textsc{Cond}\ \frac{\Gamma \vdash v : A \qquad \Gamma \vdash c_1 : \underline{C} \qquad \Gamma \vdash c_2 : \underline{D}}{\Gamma \vdash \mathtt{if}\ v\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 : (\underline{C} \sqcup \underline{D})}$$

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

$$\textsc{Ret}\ \frac{\Gamma \vdash v : A}{\Gamma \vdash \mathtt{return}\ v : A \mathbin{!} \{.\}}$$

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

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

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

Figure 11. Typing of EffCore

5 Semantics

6 Elaboration

The elaboration show how the source language can be transformed into EffCore.

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ | \ \Gamma, x : A, x : \forall\alpha.B$$

Expressions

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

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

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

$$\textsc{Hand}\ \frac{\Gamma, x : A \vdash c_r : B \,!\, \Delta \qquad \left[\begin{array}{l}(\mathsf{Op} : A_{\mathsf{Op}} \to B_{\mathsf{Op}}) \in \Sigma \\ \Gamma, x : A_{\mathsf{Op}}, k : B_{\mathsf{Op}} \to B \,!\, \Delta \vdash c_{\mathsf{Op}} : B \,!\, \Delta\end{array}\right]_{\mathsf{Op} \in O}}{\begin{array}{c}\Gamma \vdash \{\mathtt{return}\ x \mapsto c_r, [\mathsf{Op}\ y\ k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\} : \\ A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta \\ \rightsquigarrow \{\mathtt{return}\ x \mapsto c_r, [\mathsf{Op}\ y\ k \mapsto c_{\mathsf{Op}}]_{\mathsf{Op} \in O}\} : \\ A \,!\, \Delta \cup O \Rightarrow B \,!\, \Delta\end{array}}$$

Figure 12. Elaboration of source to core language: expressions

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ | \ \Gamma, x : A, x : \forall\alpha.B$$

Computations

$$\textsc{Comp}\ \frac{\Gamma \vdash c : \underline{C} \rightsquigarrow c' \qquad \underline{C} \leqslant \underline{D}}{\Gamma \vdash c : \underline{D} \rightsquigarrow c'}$$

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

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

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

$$\textsc{Op}\ \frac{(\mathsf{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathsf{Op}\ v : B \,!\, \{\mathsf{Op}\} \rightsquigarrow \mathsf{Op}\ v' : B \,!\, \{\mathsf{Op}, .\}}$$

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

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

Figure 13. Elaboration of source to core language: computations

7 Constraint Generation

Figure 14 and Figure 15 show the constraint generation algorithm for EffCore.

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ \mid \ \Gamma, x : A, x : \forall \alpha.B$$

Expressions

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

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

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

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

Figure 14. Constraint generation within expressions

$$\text{typing contexts } \Gamma \ ::= \ \epsilon \ \mid \ \Gamma, x : A, x : \forall \alpha.B$$

Computations

$$\textsc{Comp}\ \frac{\Gamma \vdash c : \underline{C} \rightsquigarrow c' \qquad \underline{C} \leqslant \underline{D}}{\Gamma \vdash c : \underline{D} \rightsquigarrow c'}$$

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

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

$$\textsc{Ret}\ \frac{\Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathtt{return}\ v : A \mathbin{!} \emptyset \rightsquigarrow \mathtt{return}\ v' : A \mathbin{!} \{.\}}$$

$$\textsc{Op}\ \frac{(\mathsf{Op} : A \to B) \in \Sigma \qquad \Gamma \vdash v : A \rightsquigarrow v'}{\Gamma \vdash \mathsf{Op}\,v : B \mathbin{!} \{\mathsf{Op}\} \rightsquigarrow \mathsf{Op}\,v' : B \mathbin{!} \{\mathsf{Op}, .\}}$$

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

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

Figure 15. Constraint generation within computations

8 Proofs

9 Implementation

10 Evaluation

11 Conclusion

A Appendix A

Acknowledgments

I would like to thank Amr Hany Saleh for his continuous guidance and help.

References

[1] 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

[2] Andrej Bauer and Matija Pretnar. 2015. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.* 84, 1 (2015), 108–123. https://doi.org/10.1016/j.jlamp.2014.02.001

[3] 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

[4] 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

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

[6] 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

[7] Sam Lindley, Conor McBride, and Craig McLaughlin. 2017. Do Be Do Be Do. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. ACM, New York, NY, USA, 500–514. https://doi.org/10.1145/3009837.3009897

[8] 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

[9] Matija Pretnar. 2014. Inferring Algebraic Effects. *Logical Methods in Computer Science* 10, 3 (2014). https://doi.org/10.2168/LMCS-10(3:21)2014

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

[11] Didier Rémy. 1994. Theoretical Aspects of Object-oriented Programming. MIT Press, Cambridge, MA, USA, Chapter Type Inference for Records in Natural Extension of ML, 67–95. http://dl.acm.org/citation.cfm?id=186677.186689