Polymorphic algebraic effects: theoretical properties and implementation

Wiktor Kuchta

Chapter 1

Introduction

A programming language needs to be more than just a lambda calculus, capable only of functional abstraction and evaluation of expressions. Programs need to have an effect on the outside world and, thinking more locally, in our programs we would like to have fragments which do not merely reduce to a value in isolation, but affect the execution of surrounding code in interesting ways. This is what we call computational effects, the typical examples of which are: input/output, mutable state, exceptions, nondeterminism, and coroutines.

Today, we are at the mercy of programming language designers to include the effects we would like to use and make sure that they all interact with each other well. In recent years, however, a new approach, called *algebraic effects*, has emerged. Algebraic effects allow programmers to express all the usual computational effects as library code and use them in the usual direct style (as opposed to monadic style, which is essentially like programming in an embedded language, which cannot which do not always play well with the rest of the language and do not compose easily). These implementations of effects are instances of one concept and interact with each other predictably.

More concretely, algebraic effects may be thought of as resumable exceptions. We can *perform* an *operation*, just as we can raise an exception in a typical high-level programming language. The operation is then handled by the nearest enclosing *handler* for the specific operation. The handler receives the value given at perform-point and also, unlike normal exception handlers, a *continuation*, a first-class functional value representing the rest of the code to execute inside the handler from the perform-point. By calling the continuation with a value we can resume at the perform-point, as if performing the operation evaluated to the value. But more interestingly, we can resume multiple times, or never, or store the continuation for later use.

This access to the continuation, while powerful, might also be a recipe for disaster. That is why languages with algebraic effects typically have *type-and-effect systems*, not only tracking the type of value an expression might evaluate to, but also what kind of effects it may perform along the way.

This work proves termination of evaluation (aka. normalization) of a particular formal language equipped with algebraic effects. We also explain how the proof can carry over to other calculi. Further, we explore the computational content of the proofs – a normalization by evaluation algorithm for the calculus.

Termination of evaluation might be of interest for many reasons. Even though practical programming languages support recursion, loops, and other features allowing nontermination (eg. recursive types), it might be useful that code that does not use these features cannot be blamed for nontermination. Nontermination is sometimes also viewed as a computational effect, which however does not neatly fit the algebraic effect framework. Despite that, it can still be added as a built-in effect to be tracked by the effect system, provided that we have sufficiently identified possible sources of nontermination.

Normalization also implies that the calculus is sound as a logic, looking through the lens of the Curry-Howard isomorphism. However, it is hard to give logical interpretations to effect annotations and it is unclear how useful it is to treat calculi with algebraic effects as logics.

The primary approach to proving normalization (and many other properties) of typed formal calculi is through *logical relations*. Logical relations are type-indexed relations on terms which essentially help carry additional information about *local* behavior of terms to make a type-derivation directed proof of the *global* property (such as normalization) have strong enough induction hypotheses and ultimately go through.

In this work we will define logical predicates [1] to prove termination of evaluation in our language. An interesting point is that the definition of our logical predicates is recursive, in other words it is a solution to a fixed-point equation. This is unlike any normalization proof we know of. The fixed-point construction replaces the use of step-indexing which appears in the logical relation construction we are most directly inspired by.

I would like to thank Dariusz Biernacki, Filip Sieczkowski, and Piotr Polesiuk for their help with this work.

[1]In the context of normalization, logical predicates have many names, including: (Tait's) reducibility predicates, reducibility candidates, and saturated sets.

Chapter 2

The language

We will study the deep handler calculus and type-and-effect system formulated in [2]. It is a refreshingly minimal language – the call-by-value lambda calculus with a few extensions to be able to express the essence of algebraic effects. There is only one unnamed universal operation, performed $\mathsf{do}\ v$. To be able to reach beyond the closest effect handler, the *lift* operator, written $[e]$, is introduced, which makes operations in e skip the nearest handler. In contrast to most work on algebraic effects, the effect-tracking system here is structural, we do not have any concept of predefined or user-defined signatures of effects. Finally, the language features polymorphic expressions and polymorphic operations.

The syntax of the calculus is shown in fig. 2.1. We use metavariables x and r to refer to expression variables. As is standard, we work modulo α-equivalence and distinguish separate instances of metavariables by using subscripts, superscripts, or adding primes.

Evaluation contexts are expressions with a hole ($\square$) and encode a left-to-right evaluation order. There is a concept of *freeness* of evaluation contexts (fig. 2.2), a quantity increased by lifts and decreased by handlers. Intuitively, an evaluation context is n-free if an operation performed inside the context (in place of the hole) would be handled by the n-th handler outside of the context.

Naturally, freeness is used in the operational semantics (fig. 2.3), where we can see that effect handling occurs if there is a 0-free context between the operation and the handler. The resumption v_c wraps the context K with the same handler, which is why the effect handlers are called *deep*. The other rules are the standard β-reduction of the λ-calculus and reductions in cases where an (effect-free) value is wrapped by an effect-related construct. It is straightforward to see that reduction is deterministic and compatible with respect to evaluation contexts.

In fig. 2.4 the type system is introduced. We have three kinds: types, effects, and rows. We use α to range over type variables and Δ to range over type contexts, i.e. mappings from type variables to kinds. A type context is written as a comma-separated sequence of type variables with their assigned kinds ($\alpha :: \kappa$). We furthermore have the concept of *type-level substitutions*: we write $\Delta \vdash \delta :: \Delta'$ if δ maps type variables in Δ' to types of the corresponding kinds, well-formed in Δ. We assume that all types are well-formed (well-kinded) in the sense of fig. 2.5.

Analogously, contexts Γ are mappings from expression variables to types, written

$$
\begin{aligned}
v, u &::= x \mid \lambda x.\, e && \text{(values)}\\
e &::= v \mid e\ e \mid [e] \mid \mathsf{do}\ v \mid \mathsf{handle}\ e\, \{x, r.\, e;\ x.\, e\} && \text{(expressions)}\\
K &::= \square \mid K\ e \mid v\ K \mid [K] \mid \mathsf{handle}\ K\, \{x, r.\, e;\ x.\, e\} && \text{(evaluation contexts)}
\end{aligned}
$$

Figure 2.1: Syntax.

$$
\frac{}{0\text{-free}(\square)} \qquad \frac{n\text{-free}(K)}{n\text{-free}(K\ e)} \qquad \frac{n\text{-free}(K)}{n\text{-free}(v\ K)} \qquad \frac{n\text{-free}(K)}{n+1\text{-free}([K])}
$$

$$
\frac{n+1\text{-free}(K)}{n\text{-free}(\mathsf{handle}\ K\, \{x, r.\, e_h;\ x.\, e_r\})}
$$

Figure 2.2: Evaluation context freeness.

as comma-separated sequences of variables with their assigned types $(x : \tau)$.

Type-level substitutions are used only in the rule for do to instantiate the polymorphic variables Δ' of effect $\Delta'.\, \tau_1 \Rightarrow \tau_2$. Complementarily, the rule for handle states that the effect handler clause e_h has to be essentially polymorphic in Δ'.

We can intuitively understand effect rows as follows: if $e : \tau \,/\, \rho$ and effect ε appears at position n in ρ, then subexpressions performing ε are wrapped by n lifts inside e. Clearly, the order of effects in the row matters. Note that the type system enables *row polymorphism*, since we can have a row ending in a polymorphic type variable. Such rows are usually called *open*.

In fig. 2.6 the subtyping rules are introduced. Their principal purpose is to allow effect rows to be subsumed by rows with more effects at the end. For that to be of use in more places, we also have standard subsumption rules for polymorphic types and function types. In particular, closed rows are subsumed by open rows.

$$
\frac{e_1 \mapsto e_2}{K[e_1] \to K[e_2]} \qquad (\lambda x.\, e)\ v \mapsto e\{v/x\} \qquad [v] \mapsto v
$$

$$
\frac{0\text{-free}(K) \qquad v_c = \lambda z.\, \mathsf{handle}\ K[z]\, \{x, r.\, e_h;\ x.\, e_r\}}{\mathsf{handle}\ K[\mathsf{do}\ v]\, \{x, r.\, e_h;\ x.\, e_r\} \mapsto e_h\{v/x\}\{v_c/r\}}
$$

$$
\mathsf{handle}\ v\, \{x, r.\, e_h;\ x.\, e_r\} \mapsto e_r\{v/x\}
$$

Figure 2.3: Single-step reduction.

$$\kappa ::= \mathsf{T} \mid \mathsf{E} \mid \mathsf{R} \qquad \sigma, \tau, \varepsilon, \rho ::= \alpha \mid \tau \to_\rho \tau \mid \forall \alpha :: \kappa.\, \tau \mid \iota \mid \Delta.\, \tau \Rightarrow \tau \mid \varepsilon \cdot \rho$$

$$\frac{x : \tau \in \Gamma}{\Delta; \Gamma \vdash x : \tau \,/\, \iota} \qquad \frac{\Delta \vdash \tau_1 :: \mathsf{T} \qquad \Delta; \Gamma, x : \tau_1 \vdash e : \tau_2 \,/\, \rho}{\Delta; \Gamma \vdash \lambda x.\, e : \tau_1 \to_\rho \tau_2 \,/\, \iota}$$

$$\frac{\Delta; \Gamma \vdash e_1 : \tau_1 \to_\rho \tau_2 \,/\, \rho \qquad \Delta; \Gamma \vdash e_2 : \tau_1 \,/\, \rho}{\Delta; \Gamma \vdash e_1\, e_2 : \tau_2 \,/\, \rho} \qquad \frac{\Delta \vdash \varepsilon :: \mathsf{E} \qquad \Delta; \Gamma \vdash e : \tau \,/\, \rho}{\Delta; \Gamma \vdash [e] : \tau \,/\, \varepsilon \cdot \rho}$$

$$\frac{\Delta, \alpha :: \kappa; \Gamma \vdash e : \tau \,/\, \iota}{\Delta; \Gamma \vdash e : \forall \alpha :: \kappa.\, \tau \,/\, \iota} \qquad \frac{\Delta \vdash \sigma :: \kappa \qquad \Delta; \Gamma \vdash e : \forall \alpha :: \kappa.\, \tau \,/\, \rho}{\Delta; \Gamma \vdash e : \tau\{\sigma/\alpha\} \,/\, \rho}$$

$$\frac{\Delta \vdash \tau_1 <: \tau_2 \qquad \Delta \vdash \rho_1 <: \rho_2 \qquad \Delta; \Gamma \vdash e : \tau_1 \,/\, \rho_1}{\Delta; \Gamma \vdash e : \tau_2 \,/\, \rho_2}$$

$$\frac{\Delta; \Gamma \vdash v : \delta(\tau_1) \,/\, \iota \qquad \Delta \vdash \delta :: \Delta' \qquad \Delta \vdash \Delta'.\, \tau_1 \Rightarrow \tau_2 :: \mathsf{E}}{\Delta; \Gamma \vdash \mathsf{do}\; v : \delta(\tau_2) \,/\, (\Delta'.\, \tau_1 \Rightarrow \tau_2)}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash e : \tau \,/\, (\Delta'.\, \tau_1 \Rightarrow \tau_2) \cdot \rho \\ \Delta, \Delta'; \Gamma, x : \tau_1, r : \tau_2 \to_\rho \tau_r \vdash e_h : \tau_r \,/\, \rho \qquad \Delta; \Gamma, x : \tau \vdash e_r : \tau_r \,/\, \rho\end{array}}{\Delta; \Gamma \vdash \mathsf{handle}\; e\; \{x, r.\, e_h; x.\, e_r\} : \tau_r \,/\, \rho}$$

Figure 2.4: Type system.

$$\frac{\alpha :: \kappa \in \Delta}{\Delta \vdash \alpha :: \kappa} \qquad \frac{\Delta \vdash \tau_1 :: \mathsf{T} \qquad \Delta \vdash \rho :: \mathsf{R} \qquad \Delta \vdash \tau_2 :: \mathsf{T}}{\Delta \vdash \tau_1 \to_\rho \tau_2 :: \mathsf{T}} \qquad \frac{\Delta, \alpha :: \kappa \vdash \tau :: \mathsf{T}}{\Delta \vdash \forall \alpha :: \kappa.\, \tau :: \mathsf{T}}$$

$$\frac{}{\Delta \vdash \iota :: \mathsf{R}} \qquad \frac{\Delta \vdash \varepsilon :: \mathsf{E} \qquad \Delta \vdash \rho :: \mathsf{R}}{\Delta \vdash \varepsilon \cdot \rho :: \mathsf{R}} \qquad \frac{\Delta, \Delta' \vdash \tau_1 :: \mathsf{T} \qquad \Delta, \Delta' \vdash \tau_2 :: \mathsf{T}}{\Delta \vdash \Delta'.\, \tau_1 \Rightarrow \tau_2 :: \mathsf{E}}$$

Figure 2.5: Well-formedness of types and rows

$$\frac{}{\Delta \vdash \sigma <: \sigma} \qquad \frac{\Delta \vdash \rho :: \mathsf{R}}{\Delta \vdash \iota <: \rho} \qquad \frac{\Delta \vdash \rho_1 <: \rho_2}{\Delta \vdash \varepsilon \cdot \rho_1 <: \varepsilon \cdot \rho_2}$$

$$\frac{\Delta \vdash \tau_2^1 <: \tau_1^1 \qquad \Delta \vdash \rho_1 <: \rho_2 \qquad \Delta \vdash \tau_1^2 <: \tau_2^2}{\Delta \vdash \tau_1^1 \to_{\rho_1} \tau_1^2 <: \tau_2^1 \to_{\rho_2} \tau_2^2} \qquad \frac{\Delta, \alpha :: \kappa \vdash \tau_1 <: \tau_2}{\Delta \vdash \forall \alpha :: \kappa.\, \tau_1 <: \forall \alpha :: \kappa.\, \tau_2}$$

Figure 2.6: Subtyping.

Chapter 3

The logical relation

[todo introduction to logical relations]

The logical relation is inspired by [1]. Some changes are due to language differences: we have only one universal operation which simplifies the treatment of effects, polymorphism does not manifest at the expression level—we do not have type lambdas, and our operations can be polymorphic. Instead of a binary step-indexed relation, our goal is to build a unary relation without step-indexing.

3.1 Definition

We begin with defining the interpretations of kinds. We call them the spaces of *semantic types* or, in the specific cases of E and R, *semantic effects*:

$$\begin{aligned} [\![\mathsf{T}]\!] &= \mathcal{P}(\mathrm{CVal}) = \mathsf{Type} \\ [\![\mathsf{E}]\!] &= \mathcal{P}(\mathrm{CVal} \times \{0\} \times \mathsf{Type}) \\ [\![\mathsf{R}]\!] &= \mathcal{P}(\mathrm{CVal} \times \mathbb{N} \times \mathsf{Type}) \end{aligned}$$

We write CVal for the set of closed values, so elements of Type are simply sets of closed values. Semantic effects are sets of triples, which aim to describe a situation in which an expression being evaluated performs an effect. The components of such a triple are: the argument of the operation, the freeness of the enclosing context beyond which the operation can be handled, and the semantic type of values we can call the resumption with.

We interpret type contexts as mappings from type variables to semantic types:

$$[\![\Delta]\!] = \{\eta \mid \forall \alpha :: \kappa \in \Delta.\, \eta(\alpha) \in [\![\kappa]\!]\}.$$

We define the interpretations of types and effects as well as relations $\mathcal{E}$ on expressions and $\mathcal{S}$ on control-stuck terms by structural induction. We parameterize the definitions by a mapping η from type variables to semantic types.

The interpretations of types are mostly standard, but we additionally have to consider effect annotations, as we can see in the case of the function type.

A polymorphic type is interpreted, as usual, as the intersection of the interpretations for all possible choices of the semantic type for the type variable. Interestingly, we can see that a polymorphic effect is interpreted as the *union* of the interpretations of the effect for all possible choices of the semantic types for the type variables.

$$
\begin{aligned}
[\![\alpha]\!]_\eta &= \eta(\alpha) \\
[\![\tau_1 \to_\rho \tau_2]\!]_\eta &= \{\lambda x.\, e \mid \forall v \in [\![\tau_1]\!]_\eta.\, e\{v/x\} \in \mathcal{E}[\![\tau_2/\rho]\!]_\eta\} \\
[\![\forall \alpha :: \kappa.\, \tau]\!]_\eta &= \{v \mid \forall \mu \in [\![\kappa]\!].\, v \in [\![\tau]\!]_{[\alpha \mapsto \mu]\eta}\} \\
[\![\Delta.\, \tau_1 \Rightarrow \tau_2]\!]_\eta &= \{(v, 0, [\![\tau_2]\!]_{\eta\eta'}) \mid \eta' \in [\![\Delta]\!] \wedge v \in [\![\tau_1]\!]_{\eta\eta'}\} \\
[\![\varepsilon \cdot \rho]\!]_\eta &= [\![\varepsilon]\!]_\eta \cup \{(v, n+1, \mu) \mid (v, n, \mu) \in [\![\rho]\!]_\eta\} \\
[\![\iota]\!]_\eta &= \emptyset
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}[\![\tau/\rho]\!]_\eta(X) &= \{e \mid \exists v \in [\![\tau]\!]_\eta.\, e \to^* v \vee \exists e' \in \mathcal{S}[\![\tau/\rho]\!]_\eta(X).\, e \to^* e'\} \\
\mathcal{S}[\![\tau/\rho]\!]_\eta(X) &= \{K[\mathsf{do}\ v] \mid \exists n, \mu.\, (v, n, \mu) \in [\![\rho]\!]_\eta \wedge n\text{-free}(K) \wedge \forall u \in \mu.\, K[u] \in X\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}[\![\tau/\rho]\!]_\eta &= \bigcap \{X \mid \mathcal{E}[\![\tau/\rho]\!]_\eta(X) \subseteq X\} \\
\mathcal{S}[\![\tau/\rho]\!]_\eta &= \mathcal{S}[\![\tau/\rho]\!]_\eta(\mathcal{E}[\![\tau/\rho]\!]_\eta)
\end{aligned}
$$

The definition of the set $\mathcal{E}[\![\tau/\rho]\!]_\eta$ is recursive and we interpret it inductively. The intuition is that an element of $\mathcal{E}[\![\tau/\rho]\!]_\eta$ evaluates to a value of type τ while possibly performing an effect in ρ and getting control-stuck finitely many times along the way. When the term is control-stuck we do not have a handler, but we do know the return type of the operation, so we resume with all possible values in parallel.

In other words, we can associate with an element in $\mathcal{E}[\![\tau/\rho]\!]_\eta$ its "reduction trace" – a tree where edges are the usual reduction ($\to$) relation, except for $\mathcal{S}[\![\tau/\rho]\!]_\eta$ nodes, which have a branch for each element of μ, the value with which we replace the $\mathsf{do}\ v$.[1] Note that different choices of values to resume with could theoretically lead to getting stuck a different number of times, so the depth of the tree is not uniform, and it could even be infinite. These trees are however well-founded thanks to the choice of the inductive interpretation.

To describe the construction in more detail, we temporarily overload notation and define operators $\mathcal{S}[\![\tau/\rho]\!]_\eta$ and $\mathcal{E}[\![\tau/\rho]\!]_\eta$ on sets of expressions (denoted by X). They are clearly monotone, so by the Knaster-Tarski theorem the fixed-point equation $\mathcal{E}[\![\tau/\rho]\!]_\eta(X) = X$ has a least solution. Moreover, it can be characterized as the intersection of all $\mathcal{E}[\![\tau/\rho]\!]_\eta$-closed sets. We immediately obtain an induction principle.

Lemma 1 (Tarski induction principle)**.** *If $\mathcal{E}[\![\tau/\rho]\!]_\eta(X) \subseteq X$, then $\mathcal{E}[\![\tau/\rho]\!]_\eta \subseteq X$.*

By expanding out the definition of the function $\mathcal{E}[\![\tau/\rho]\!]_\eta$ and treating X as a predicate P, we get the more familiar principle of structural induction on $\mathcal{E}[\![\tau/\rho]\!]_\eta$.

Lemma 2 (Induction principle)**.** *Assume P is a predicate on closed expressions and*

- *if e evaluates to a value in $[\![\tau]\!]_\eta$, then $P(e)$ holds; and*

[1] If we formalized the definitions in a programming language (or proof assistant), this is also how we could view the structure of values of the inductive type $\mathcal{E}[\![\tau/\rho]\!]_\eta$.

- *if e reduces to some $K[\mathsf{do}\ v]$ such that there exist $(v, n, \mu) \in [\![\rho]\!]_\eta$ such that n-free(K) and $P(K[u])$ holds for all $u \in \mu$, then $P(e)$ holds.*

Then $P(e)$ holds for all $e \in \mathcal{E}[\![\tau/\rho]\!]_\eta$.

We also note some straightforward but useful properties of the relations.

Lemma 3 (Value inclusion)**.** *For any τ and ρ we have $[\![\tau]\!]_\eta \subseteq \mathcal{E}[\![\tau/\rho]\!]_\eta$.*

Lemma 4 (Control-stuck inclusion)**.** *For any τ and ρ we have $\mathcal{S}[\![\tau/\rho]\!]_\eta \subseteq \mathcal{E}[\![\tau/\rho]\!]_\eta$.*

Lemma 5 (Closedness under antireduction)**.** *If $e \to^* e' \in \mathcal{E}[\![\tau/\rho]\!]_\eta$, then $e \in \mathcal{E}[\![\tau/\rho]\!]_\eta$.*

Lemma 6. *If $[\![\tau_1]\!]_\eta \subseteq [\![\tau_2]\!]_\eta$ and $[\![\rho_1]\!]_\eta \subseteq [\![\rho_2]\!]_\eta$, then $\mathcal{S}[\![\tau_1/\rho_1]\!]_\eta \subseteq \mathcal{S}[\![\tau_2/\rho_2]\!]_\eta$ and $\mathcal{E}[\![\tau_1/\rho_1]\!]_\eta \subseteq \mathcal{E}[\![\tau_2/\rho_2]\!]_\eta$.*

3.2 Compatibility lemmas

We want to establish that $\vdash e : \tau \,/\, \iota$ implies $e \in \mathcal{E}[\![\tau/\iota]\!]$.

For this purpose we will prove a semantic counterpart of each typing rule. First, we need to define a counterpart to the typing judgment. Unlike typing judgments, our relations are on closed terms only, so we get around that by using substitution. We define semantic entailment as

$$\Delta; \Gamma \models e : \tau \,/\, \rho \iff \forall \eta \in [\![\Delta]\!].\, \forall \gamma \in [\![\Gamma]\!]_\eta.\, \gamma(e) \in \mathcal{E}[\![\tau/\rho]\!]_\eta,$$

where $[\![\Gamma]\!]_\eta = \{\gamma \mid \forall x : \tau \in \Gamma.\, \gamma(x) \in [\![\tau]\!]_\eta\}$ contains expression-level variable substitutions.

Lemma 7 (Variable compatibility)**.**

$$\frac{x : \tau \in \Gamma}{\Delta; \Gamma \models x : \tau \,/\, \iota}$$

Proof. Assume $x : \tau \in \Gamma$. We want to prove $\Delta; \Gamma \models x : \tau \,/\, \iota$. Take any $\eta \in [\![\Delta]\!]$ and $\gamma \in [\![\Gamma]\!]_\eta$. We want to show $\gamma(x) \in \mathcal{E}[\![\tau/\iota]\!]_\eta$. From the definition of $[\![\Gamma]\!]_\eta$ we know that $\gamma(x) \in [\![\tau]\!]_\eta$, so by lemma 3 we have $\gamma(x) \in \mathcal{E}[\![\tau/\iota]\!]_\eta$. □

Lemma 8 (Abstraction compatibility)**.**

$$\frac{\Delta \vdash \tau_1 :: \mathsf{T} \qquad \Delta; \Gamma, x : \tau_1 \models e : \tau_2 \,/\, \rho}{\Delta; \Gamma \models \lambda x.\, e : \tau_1 \to_\rho \tau_2 \,/\, \iota}$$

Proof. Assume $\Delta \vdash \tau_1 :: \mathsf{T}$ and $\Delta; \Gamma, x : \tau_1 \models e : \tau_2 \,/\, \rho$. We want to prove $\Delta; \Gamma \models \lambda x.\, e : \tau_1 \to_\rho \tau_2 \,/\, \iota$. Take any $\eta \in [\![\Delta]\!]$ and $\gamma \in [\![\Gamma]\!]_\eta$. By lemma 3 it suffices to show $\gamma(\lambda x.\, e) = \lambda x.\, \gamma(e) \in [\![\tau_1 \to_\rho \tau_2]\!]_\eta$. So take any $v \in [\![\tau_1]\!]_\eta$. We need to show $\gamma(e)\{v/x\} \in \mathcal{E}[\![\tau_2/\rho]\!]_\eta$. Let $\gamma' = \gamma'[x \mapsto v]$. Then $\gamma' \in [\![\Gamma, x : \tau_1]\!]_\eta$, so $\gamma(e)\{v/x\} = \gamma'(e) \in \mathcal{E}[\![\tau_2/\rho]\!]_\eta$. □

For clarity of presentation, in the following we will assume Γ empty. The lemmas in full generality can be recovered by substituting an interpretation of Γ.

Lemma 9 (Lift compatibility)**.**

$$\frac{\Delta \vdash \varepsilon :: \mathsf{E} \qquad \Delta; \models e : \tau \mathbin{/} \rho}{\Delta; \models [e] : \tau \mathbin{/} \varepsilon \cdot \rho}$$

Proof. Assume $\Delta \vdash \tau :: \mathsf{T}$, $\Delta \vdash \varepsilon :: \mathsf{E}$, and $\Delta \vdash \rho :: \mathsf{R}$. Take any $\eta \in [\![\Delta]\!]$. We will show by induction on $e \in \mathcal{E}[\![\tau/\rho]\!]_\eta$ that $[e] \in \mathcal{E}[\![\tau/\varepsilon \cdot \rho]\!]_\eta$.

If $e \to^* K[\mathsf{do}\ v]$ and there exists $(v, n, \mu) \in [\![\rho]\!]_\eta$ such that n-free(K) and for all $u \in \mu$ the induction hypothesis holds for $K[u]$, then we have $(v, n+1, \mu) \in [\![\varepsilon \cdot \rho]\!]_\eta$, $n+1$-free$([K])$, and $\forall u \in \mu.\, [K[u]] \in \mathcal{E}[\![\tau/\varepsilon \cdot \rho]\!]_\eta$. So $[K[\mathsf{do}\ v]] \in \mathcal{S}[\![\tau/\varepsilon \cdot \rho]\!]_\eta$ and $[e] \in \mathcal{E}[\![\tau/\varepsilon \cdot \rho]\!]_\eta$ by antireduction.

If $e \to^* v \in [\![\tau]\!]_\eta$, then $[e] \to^* [v] \to v$, so $[e] \in \mathcal{E}[\![\tau/\varepsilon \cdot \rho]\!]_\eta$. □

Lemma 10 (Application compatibility)**.**

$$\frac{\Delta; \models e_1 : \tau_1 \to_\rho \tau_2 \mathbin{/} \rho \qquad \Delta; \models e_2 : \tau_1 \mathbin{/} \rho}{\Delta; \models e_1\, e_2 : \tau_2 \mathbin{/} \rho}$$

Proof. Fix any well-formed Δ and $\tau_1 \to_\rho \tau_2$. Take any $\eta \in [\![\Delta]\!]$ and $e_2 \in \mathcal{E}[\![\tau_1/\rho]\!]_\eta$. We will show by induction on $e_1 \in \mathcal{E}[\![\tau_1 \to_\rho \tau_2/\rho]\!]_\eta$ that $e_1\ e_2 \in \mathcal{E}[\![\tau_2/\rho]\!]_\eta$.

If $e_1 \to^* K_1[\mathsf{do}\ v]$ and there exists $(v, n, \mu) \in [\![\rho]\!]_\eta$ such that n-free(K_1) and for all $u \in \mu$ the inductive hypothesis holds for $K_1[u]$, then $K_1[\mathsf{do}\ v]\ e_2 \in \mathcal{S}[\![\tau_2/\rho]\!]_\eta$, since n-free$(K_1\ e_2)$. By antireduction $e_1\ e_2 \in \mathcal{E}[\![\tau_2/\rho]\!]_\eta$.

Now assume $e_1 \to^* (\lambda x.\, e) \in [\![\tau_1 \to_\rho \tau_2/\rho]\!]_\eta$. We will show by induction on $e_2 \in \mathcal{E}[\![\tau_1/\rho]\!]_\eta$ that $(\lambda x.\, e)\ e_2 \in \mathcal{E}[\![t_2/\rho]\!]_\eta$ and the claim will follow by antireduction.

If $e_2 \to^* K_2[\mathsf{do}\ v]$ and there exists $(v, n, \mu) \in [\![\rho]\!]_\eta$ such that n-free(K_2) and for all $u \in \mu$ the inductive hypothesis holds for $K_2[u]$, then $(\lambda x.\, e)\ K_2[\mathsf{do}\ v] \in \mathcal{S}[\![\tau_2/\rho]\!]_\eta$, since n-free$((\lambda x.\, e)\ K_2)$. By antireduction $(\lambda x.\, e)\ e_2 \in \mathcal{E}[\![\tau_2/\rho]\!]_\eta$.

If $e_2 \to^* v \in [\![\tau_1]\!]_\eta$, then $(\lambda x.\, e)\ e_2 \to^* (\lambda x.\, e)\ v \to e\{v/x\} \in \mathcal{E}[\![\tau_2/\rho]\!]_\eta$. □

Lemma 11 (Handle compatibility)**.**

$$\frac{\begin{array}{c}\Delta; \models e : \tau \mathbin{/} (\Delta'.\, \tau_1 \Rightarrow \tau_2) \cdot \rho \\ \Delta, \Delta'; x : \tau_1, r : \tau_2 \to_\rho \tau_r \models e_h : \tau_r \mathbin{/} \rho \qquad \Delta; x : \tau \models e_r : \tau_r \mathbin{/} \rho\end{array}}{\Delta; \models \mathsf{handle}\ e\ \{x, r.\, e_h; x.\, e_r\} : \tau_r \mathbin{/} \rho}$$

Proof. Assume $\Delta, \Delta'; x : \tau_1, r : \tau_2 \to_\rho \tau_r \models e_h : \tau_r \mathbin{/} \rho$ and $\Delta; x : \tau \models e_r : \tau_r \mathbin{/} \rho$. Let h stand for $\{x, r.\, e_h;\ x.\, e_r\}$. Take any $\eta \in [\![\Delta]\!]$. We will show by induction on $e \in \mathcal{E}[\![\tau/(\Delta'.\, \tau_1 \Rightarrow \tau_2) \cdot \rho]\!]_\eta$ that $\mathsf{handle}\ e\, h \in \mathcal{E}[\![\tau_r/\rho]\!]_\eta$. Note that only τ_1 and τ_2 require Δ' to be in context.

If $e \to^* v \in [\![\tau]\!]_\eta$, then $\mathsf{handle}\ e\, h \to^* e_r\{v/x\} \in \mathcal{E}[\![\tau_r/\rho]\!]_\eta$, so the claim follows by antireduction.

Now assume $e \to^* K[\mathsf{do}\ v]$ and we have $(v, n, \mu) \in [\![(\Delta'.\, \tau_1 \Rightarrow \tau_2) \cdot \rho]\!]_\eta$ such that n-free(K) and for all $u \in \mu$ the induction hypothesis holds for $K[u]$.

If $n = 0$, then $(v, n, \mu) \in [\![\tau_1 \Rightarrow \tau_2]\!]_{\eta\eta'}$ for some $\eta' \in [\![\Delta']\!]$. More specifically, $v \in [\![\tau_1]\!]_{\eta\eta'}$ and $\mu = [\![\tau_2]\!]_{\eta\eta'}$. We have $\mathsf{handle}\ e\, h \to^* \mathsf{handle}\ K[\mathsf{do}\ v]\, h \to e_h\{v/x\}\{v_c/r\}$, where $v_c = \lambda z.\,\mathsf{handle}\ K[z]\,h$. To show $v_c \in [\![\tau_2 \to_\rho \tau_r]\!]_{\eta\eta'}$, take any $u \in [\![\tau_2]\!]_{\eta\eta'}$ and show $\mathsf{handle}\ K[u]\,h \in \mathcal{E}[\![\tau_r/\rho]\!]_{\eta\eta'} = \mathcal{E}[\![\tau_r/\rho]\!]_{\eta}$. Which holds by induction hypothesis. Therefore, $e_h\{v/x\}\{v_c/r\}$ is in $\mathcal{E}[\![\tau_r/\rho]\!]_\eta$ and so is $\mathsf{handle}\ e\ h$.

If $n > 0$, then $\mathsf{handle}\ K[\mathsf{do}\ v]\,h \in \mathcal{S}[\![\tau_r/\rho]\!]_\eta$, since $n-1$-free($\mathsf{handle}\ K\,h$), $(v, n-1, \mu) \in [\![\rho]\!]_\eta$, and $\forall u \in \mu.\,\mathsf{handle}\ K[u]\,h \in \mathcal{E}[\![\tau_r/\rho]\!]_\eta$. Again, the claim follows by antireduction.

□

Lemma 12. *If Δ and Δ' disjoint, $\Delta \vdash \delta :: \Delta'$, and $\Delta, \Delta' \vdash \tau :: \kappa$, $\eta \in [\![\Delta]\!]$, and η' extends η by mappings $\alpha \mapsto [\![\delta(\alpha)]\!]_\eta$, then $[\![\tau]\!]_{\eta'} = [\![\delta(\tau)]\!]_\eta$.*

Proof. By induction on the kinding rules.

If $\tau = \alpha \in \Delta$, then both sides are equal to $\eta(\alpha)$.

If $\tau = \alpha \in \Delta'$, then equality follows from the definition of η'.

If $\tau = \iota$, then both sides are empty.

If $\tau = \forall\alpha :: \kappa.\,\tau'$, then $[\![\tau]\!]_{\eta'} = \bigcap\{[\![\tau']\!]_{\eta'[\alpha\mapsto\mu]} \mid \mu \in [\![\kappa]\!]\}$ and $[\![\delta(\tau)]\!]_\eta = \bigcap\{[\![\delta(\tau')]\!]_{\eta[\alpha\mapsto\mu]} \mid \mu \in [\![\kappa]\!]\}$, which are equal by the inductive hypothesis (taking $\Delta, \alpha :: \kappa$ as Δ in the statement).

If $\tau = \Delta''.\,\tau_1 \Rightarrow \tau_2$, then $[\![\tau]\!]_{\eta'} = \{(v, 0, [\![\tau_2]\!]_{\eta'\eta''}) \mid \eta'' \in [\![\Delta'']\!] \wedge v \in [\![\tau_1]\!]_{\eta'\eta''}\}$ and $[\![\delta(\tau)]\!]_\eta = \{(v, 0, [\![\delta(\tau_2)]\!]_{\eta\eta''}) \mid \eta'' \in [\![\Delta'']\!] \wedge v \in [\![\delta(\tau_1)]\!]_{\eta\eta''}\}$, which are equal q'by induction (taking Δ, Δ'' as Δ in the statement).

If $\tau = \varepsilon \cdot \rho$, then $[\![\tau]\!]_{\eta'} = [\![\varepsilon]\!]_{\eta'} \cup \{(v, n+1, \mu) \mid (v, n, \mu) \in [\![\rho]\!]_{\eta'}\}$ and $[\![\delta(\tau)]\!]_\eta = [\![\delta(\varepsilon)]\!]_\eta \cup \{(v, n+1, \mu) \mid (v, n, \mu) \in [\![\delta(\rho)]\!]_\eta\}$, which are equal by the inductive hypothesis. □

Lemma 13 (Do compatibility)**.**

$$\frac{\Delta; \models v : \delta(\tau_1) \,/\, \iota \qquad \Delta \vdash \delta :: \Delta' \qquad \Delta \vdash \Delta'.\,\tau_1 \Rightarrow \tau_2 :: \mathsf{E}}{\Delta; \models \mathsf{do}\ v : \delta(\tau_2) \,/\, (\Delta'.\,\tau_1 \Rightarrow \tau_2)}$$

Proof. Assume $\Delta \vdash \delta :: \Delta'$, $\Delta \vdash \Delta'.\,\tau_1 \to \tau_2 :: \mathsf{E}$. Take any $\eta \in [\![\Delta]\!]$. Assume $v \in [\![\delta(\tau_1)]\!]_\eta$. We want to show $\mathsf{do}\ v \in \mathcal{E}[\![\delta(\tau_2)/(\Delta'.\,\tau_1 \Rightarrow \tau_2)]\!]_\eta$.

By lemma 4 it suffices to show $\mathsf{do}\ v \in \mathcal{S}[\![\delta(\tau_2)/(\Delta'.\,\tau_1 \Rightarrow \tau_2)]\!]_\eta$. By taking the empty context in the definition of $\mathcal{S}$ and lemma 3, it suffices to show $(v, 0, [\![\delta(\tau_2)]\!]_\eta) \in [\![\Delta'.\,\tau_1 \Rightarrow \tau_2]\!]_\eta$. From the interpretation of polymorphic effects, it would be enough to show $(v, 0, [\![\delta(\tau_2)]\!]_\eta) \in [\![\tau_1 \Rightarrow \tau_2]\!]_{\eta'}$, where η' extends η by mappings $\alpha \mapsto [\![\delta(\alpha)]\!]_\eta$ for all $\alpha \in \Delta'$. By the interpretation of operations, it remains to show $v \in [\![\tau_1]\!]_{\eta'}$ and $[\![\delta(\tau_2)]\!]_\eta = [\![\tau_2]\!]_{\eta'}$. Which follows immediately from lemma 12. □

Lemma 14 (Subtyping compatibility)**.**

$$\frac{\Delta \vdash \tau_1 <: \tau_2 \qquad \Delta \vdash \rho_1 <: \rho_2 \qquad \Delta; \models e : \tau_1 \,/\, \rho_1}{\Delta; \models e : \tau_2 \,/\, \rho_2}$$

Proof. By induction on subtyping rules we will show that if $\Delta \vdash \sigma_1 <: \sigma_2$, then $[\![\sigma_1]\!]_\eta \subseteq [\![\sigma_2]\!]_\eta$ for all $\eta \in [\![\Delta]\!]$. Then, by lemma 6, from $\Delta \vdash \tau_1 <: \tau_2$, $\Delta \vdash \rho_1 <: \rho_2$ and $\Delta; \Gamma \models e : \tau_1 \,/\, \rho_1$ we will be able to conclude $\Delta; \Gamma \models e : \tau_2 \,/\, \rho_2$.

For the case of the reflexivity rule, we obviously have $[\![\sigma]\!]_\eta \subseteq [\![\sigma]\!]_\eta$.

For the case of the function type rule, assume $[\![\tau_2^1]\!]_\eta \subseteq [\![\tau_1^1]\!]_\eta$, $[\![\rho_1]\!]_\eta \subseteq [\![\rho_2]\!]_\eta$, and $[\![\tau_1^2]\!]_\eta \subseteq [\![\tau_2^2]\!]_\eta$. We want to show $[\![\tau_1^1 \to_{\rho_1} \tau_1^2]\!]_\eta \subseteq [\![\tau_2^1 \to_{\rho_2} \tau_2^2]\!]_\eta$. So take any $(\lambda x.\, e)$ in the former and any $v \in [\![\tau_2^1]\!]$. Since $v \in [\![\tau_1^1]\!]_\eta$, we have $e\{v/x\} \in \mathcal{E}[\![\tau_1^2/\rho_1]\!]_\eta$. By lemma 6 we obtain $e\{v/x\} \in \mathcal{E}[\![\tau_2^2/\rho_2]\!]_\eta$ as desired.

For the case of the universal quantifier rule, assume $[\![\tau_1]\!]_{\eta[\alpha\mapsto\mu]} \subseteq [\![\tau_2]\!]_{\eta[\alpha\mapsto\mu]}$ for all $\mu \in [\![\kappa]\!]$. The claim holds since

$$\begin{aligned}[\![\forall\alpha :: \kappa.\, \tau_1]\!]_\eta &= \{v \mid \forall\mu \in [\![\kappa]\!].\, v \in [\![\tau_1]\!]_{[\alpha\mapsto\mu]\eta}\} \\ &\subseteq \{v \mid \forall\mu \in [\![\kappa]\!].\, v \in [\![\tau_2]\!]_{[\alpha\mapsto\mu]\eta}\} = [\![\forall\alpha :: \kappa.\, \tau_2]\!]_\eta.\end{aligned}$$

The case of the empty row rule holds trivially, since $[\![\iota]\!]_\eta = \emptyset$.

For the case of the row extension rule, assume $[\![\rho_1]\!]_\eta \subseteq [\![\rho_2]\!]_\eta$. We clearly have

$$\{(v, n+1, \mu) \mid (v, n, \mu) \in [\![\rho_1]\!]_\eta\} \subseteq \{(v, n+1, \mu) \mid (v, n, \mu) \in [\![\rho_2]\!]_\eta\},$$

so $[\![\varepsilon \cdot \rho_1]\!]_\eta \subseteq [\![\varepsilon \cdot \rho_2]\!]_\eta$ as well.

□

Lemma 15 (Polymorphism introduction compatibility).

$$\frac{\Delta, \alpha :: \kappa; \models e : \tau \;/\; \iota}{\Delta; \models e : \forall\alpha :: \kappa.\, \tau \;/\; \iota}$$

Proof. Assume $\Delta, \alpha :: \kappa; \models e : \tau \;/\; \iota$. Take $\eta \in [\![\Delta]\!]$. We know e evaluates to a value in $[\![\tau]\!]_{\eta[\alpha\mapsto\mu]}$ for any $\mu \in [\![\kappa]\!]$. Therefore this value is in $[\![\forall\alpha :: \kappa.\, \tau]\!]_\eta$, and by antireduction $e \in \mathcal{E}[\![\forall\alpha :: \kappa.\, \tau/\iota]\!]_\eta$. □

Lemma 16 (Polymorphism elimination compatibility).

$$\frac{\Delta \vdash \sigma :: \kappa \qquad \Delta; \models e : \forall\alpha :: \kappa.\, \tau \;/\; \rho}{\Delta; \models e : \tau\{\sigma/\alpha\} \;/\; \rho}$$

Proof. Assume $\Delta \vdash \sigma :: \kappa$ and $\Delta; \models e : \forall\alpha :: \kappa.\, \tau \;/\; \rho$. By lemma 12 we have $[\![\tau\{\sigma/\alpha\}]\!]_\eta = [\![\tau]\!]_{\eta[\alpha\mapsto[\![\sigma]\!]_\eta]}$, which is a superset of $[\![\forall\alpha :: \kappa.\, \tau]\!]$. So we have $\Delta; \models e : \tau\{\sigma/\alpha\} \;/\; \rho$ by lemma 6. □

Theorem 1 (Termination of evaluation).

Proof. If we have a derivation of $\vdash e : \tau \;/\; \iota$, then after we replace each rule by the corresponding compatibility lemma, we obtain $\models e : \tau \;/\; \iota$, so $e \in \mathcal{E}[\![\tau/\iota]\!]$. Therefore e has to terminate to a value, since $[\![\iota]\!]$ is empty and hence $\mathcal{S}[\![\tau/\iota]\!]$ is empty. □

This is also an alternative proof of soundness ("well-typed programs do not go wrong"), which was already shown by the technique of progress and preservation [2].

3.3 Applicability to other settings

Bibliography

[1] Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Handle with care: relational interpretation of algebraic effects and handlers. *Proc. ACM Program. Lang.*, 2(POPL):8:1–8:30, 2018.

[2] Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Typed equivalence of effect handlers and delimited control. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.