Semantics of an Effect Analysis for Exceptions

Nick Benton
Microsoft Research
nick@microsoft.com

Peter Buchlovsky
University of Cambridge Computer Laboratory
Peter.Buchlovsky@cl.cam.ac.uk

Abstract

We give a semantics to a polymorphic effect analysis that tracks possibly-thrown exceptions and possible non-termination for a higher-order language. The semantics is defined using partial equivalence relations over a standard monadic, domain-theoretic model of the original language and establishes the correctness of both the analysis itself and of the contextual program transformations that it enables.

Categories and Subject Descriptors F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Denotational semantics, Program analysis; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs—Logics of programs; D.3.4 [*Programming Languages*]: Processors—Compilers, Optimization

General Terms Languages, Theory

Keywords Program analysis, exceptions, optimizing compilation, effect systems, types, denotational semantics, partial equivalence relations

1. Introduction

Proving the correctness of static analyses, and of the optimizing transformations they enable, can be surprisingly difficult. The relational approach, interpreting non-standard/annotated types (equivalently, abstract domain elements) as binary relations, has several advantages over the standard one, using unary predicates. Firstly, it allows even apparently very intensional program analyses, such as ones describing the store effects of commands, to be interpreted in an extensional semantics for the original language. This allows the true meaning of the properties of interest (and their use in justifying transformations) to be decoupled from particular, approximate syntactic analyses for establishing those properties. Secondly, it deals naturally with dependency-based properties, such as those relating to secure information flow, which are not naturally expressible in terms of unary predicates. Finally, using relations builds the notion of equivalence-in-context into the interpretation, which leads fairly directly to correctness proofs for analysis-dependent program transformations.

In earlier work, we have given semantics to both traditional dataflow analyses for first-order imperative programs [4] and an analysis of read and write effects for a higher-order language with global store [7] in this style. In the present paper, we show how an effect analysis [14, 22] that tracks possible divergence and exception-throwing in an impure applied CBV λ-calculus may also be given a natural and useful relational interpretation.

Whilst the analysis considered here is not trivial, neither is it especially novel. The contribution of the paper is in the semantic interpretation and soundness proofs, showing that our general relational methodology for understanding types, analyses and transformations extends easily to the case of exception analysis. There are also a number of small technical extensions to our previous work: we deal with recursion, we include both effect polymorphism and a form of conjunctive types and we make interesting use of the empty value type, allowing the analysis to express 'must throw' and 'must diverge' properties of computations as well as the more usual 'may' properties.

1.1 Overview

Extended type systems that approximate the possible exceptions that may be thrown by a phrase are actually one of the most commonly used effect analyses, since Java [10] (like Modula 3) requires methods to be annotated with `throws` clauses, listing a set of classes $\mathcal{T}$ such that any (non-fatal, 'checked') exception thrown by the method is guaranteed to be a subclass of some element of $\mathcal{T}$. A monomorphic analysis like that used in Java, in which each method is annotated with one finite set of literal bounds covering all possible exceptions that may be thrown, is unduly restrictive for higher-order code, so here we will consider an effect system with general subtyping (rather than just subeffecting) and also effect polymorphism. Parametric polymorphism over effects is useful for capturing generic flows of effects (e.g. the `map` function has whatever effect its argument has) but one can also gain useful accuracy by adding ad hoc effect polymorphism in the form of conjunction on effect-annotated types, which we will do here too.

As an example of the sort of property the analysis computes, consider the following (rather contrived) SML definition of a function `f` of type `(int->bool)->int->int`:

```
fun f g x = (if g x then x else raise E1)
            handle E2 => f g (x+1)
```

The analysis shows that the set of exceptions possibly thrown by an application `f g x` is the set of exceptions that might be thrown by `g` minus `E2` plus `E1`. Furthermore, if `g` is total (does not diverge) *and* cannot throw `E2`, then `f` is total too. Finally, if `g` *always* either throws `E2` or diverges, then `f g x` always diverges.

The analysis will be presented as inference rules defining a non-standard refinement type system, or propositional program logic. This declarative specification separates the definition of *what* the analysis is from algorithmic details, which we will not consider here, of *how* one might actually compute it efficiently. The properties of the function `f` described above are formally captured by the derivability of a judgement that the translation $\mathtt{f}^*$ of `f` into our

TLDI'07 January 16, 2007, Nice, France.

$\boxed{\Gamma \vdash V : A}$

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \vdash E : \mathtt{exn}} \qquad \frac{}{\Gamma \vdash () : \mathtt{unit}}$$

$$\frac{}{\Gamma \vdash n : \mathtt{int}} \qquad \frac{}{\Gamma \vdash b : \mathtt{bool}} \qquad \frac{\Gamma \vdash V_1 : A \quad \Gamma \vdash V_2 : B}{\Gamma \vdash (V_1, V_2) : A \times B}$$

$$\frac{\Gamma, x : A, f : A \to \mathrm{T}B \vdash M : \mathrm{T}B}{\Gamma \vdash \mathtt{rec}\ f(x : A) \Leftarrow M : A \to \mathrm{T}B}$$

$\boxed{\Gamma \vdash H : \mathrm{T}A}$

$$\frac{\Gamma \vdash M : \mathrm{T}A \quad \Gamma \vdash H : \mathrm{T}A}{\Gamma \vdash E \Rightarrow M, H : \mathrm{T}A} \qquad \frac{}{\Gamma \vdash - : \mathrm{T}A}$$

$\boxed{\Gamma \vdash M : \mathrm{T}A}$

$$\frac{\Gamma \vdash V_1 : A \to \mathrm{T}B \quad \Gamma \vdash V_2 : A}{\Gamma \vdash V_1 V_2 : \mathrm{T}B} \qquad \frac{\Gamma \vdash V : A_1 \times A_2}{\Gamma \vdash \pi_i\, V : \mathrm{T}A_i}$$

$$\frac{\Gamma \vdash V_1 : \mathtt{int} \quad \Gamma \vdash V_2 : \mathtt{int}}{\Gamma \vdash V_1 + V_2 : \mathrm{T}\mathtt{int}}$$

$$\frac{\Gamma \vdash V : \mathtt{bool} \quad \Gamma \vdash M : \mathrm{T}A \quad \Gamma \vdash N : \mathrm{T}A}{\Gamma \vdash \mathtt{if}\ V\ \mathtt{then}\ M\ \mathtt{else}\ N : \mathrm{T}A}$$

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathtt{val}\ V : \mathrm{T}A} \qquad \frac{\Gamma \vdash V : \mathtt{exn}}{\Gamma \vdash \mathtt{raise}\ V : \mathrm{T}A}$$

$$\frac{\Gamma \vdash M : \mathrm{T}A \quad \Gamma, x : A \vdash P : \mathrm{T}B \quad \Gamma \vdash H : \mathrm{T}B}{\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H : \mathrm{T}B}$$

Figure 1. Simple type system

intermediate language has type (though this is still not principal):

$$\begin{array}{l}\forall\alpha.(\mathtt{int} \to \mathrm{T}_{\alpha}\mathtt{bool}) \to \mathrm{T}_{\mathtt{f}}(\mathtt{int} \to \mathrm{T}_{(\alpha \setminus E2) \cup \bot \cup E1}\mathtt{int}) \\ \wedge \forall\alpha.(\mathtt{int} \to \mathrm{T}_{(\alpha \setminus \bot) \setminus E2}\mathtt{bool}) \to \mathrm{T}_{\mathtt{f}}(\mathtt{int} \to \mathrm{T}_{\alpha \cup E1}\mathtt{int}) \\ \wedge (\mathtt{int} \to \mathrm{T}_{E2 \cup \bot}\mathbf{0}_{\mathtt{bool}}) \to \mathrm{T}_{\mathtt{f}}(\mathtt{int} \to \mathrm{T}_{\bot}\mathbf{0}_{\mathtt{int}})\end{array}$$

The type above displays most of the features of our refined system: it is a conjunction ('$\wedge$') of effect-polymorphic ('$\forall\alpha.$') monadic types in which the computation type constructor ('T') is indexed by effects and there is an indexed empty type ('$\mathbf{0}_{\mathtt{int}}$', '$\mathbf{0}_{\mathtt{bool}}$'). The effects are built from effect variables ('α'), constants for exceptions ('$E1$', '$E2$') and divergence ('$\bot$') by the operations of union ('$\cup$') and set difference ('$\setminus$').

The soundness of the analysis alone could be expressed by interpreting refined types as subsets of (unary relations on) their underlying, non-refined versions. But we are also interested in proving the correctness of transformations, i.e. in equalities. At higher order, equality at a subtype is not simply the restriction of equality at a supertype to the set of values in the subtype, so we need to augment subsets with equivalence relations defining the subtype-specific notion of equality. A subset equipped with an equivalence relation is just a symmetric and transitive relation, also known as a partial equivalence relation (PER). Thus we will interpret refined types as PERs on the interpretations of their underlying simple types. That a term V has a particular refined type X will be interpreted by the denotation of V being related to itself by (i.e. in the diagonal of) the interpretation of X.

Program transformations are formalized by rules for deriving typed equations in context. For example, we will be able to derive

$$\begin{array}{l}\vdash \mathtt{f}^* = \mathtt{f'}^* \\ \quad : \forall\alpha.(\mathtt{int} \to \mathrm{T}_{(\alpha \setminus \bot) \setminus E2}\mathtt{bool}) \to \mathrm{T}_{\mathtt{f}}(\mathtt{int} \to \mathrm{T}_{\alpha \cup E1}\mathtt{int})\end{array}$$

where `f'` is

```
fun f' g x = if g x then x else raise E1
```

which expresses that one may remove the handler from the definition of `f` (and guarantee termination) in contexts in which the argument `g` is guaranteed not to diverge or throw `E2`. More interesting transformations involve commuting computations and lifting provably pure computations out of lambda abstractions. Such equational judgements are interpreted by the denotations of two different terms being related by the PER interpreting a refined type.

2. Basic Language with Exceptions

We consider a strict functional programming language with a countable set of exceptions $\mathbb{E}$ and a monadic type system. Exceptions are raised by $\mathtt{raise}\ E$ and caught by $\mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H$. Intuitively, this expression evaluates M, binds the result to x and evaluates P. If the evaluation of M raises an exception then the appropriate handler in H (if any) is invoked. The reason for using explicitly monadic types, and the non-standard construct for exception handling [6], is that they simplify both the equational properties of the language and the presentation of the effect system. A more conventional ML-like language can be translated into this one by a variant of Moggi's CBV translation [16], and this extends to the more usual style of presenting effect systems, in which all expressions have both a type and an effect, and function arrows are annotated with a 'latent' effect [23].

Formally, the syntax of our language is as follows:

$$\begin{array}{rcl} E & \in & \mathbb{E} \\ A, B & ::= & \mathtt{unit} \mid \mathtt{int} \mid \mathtt{bool} \mid \mathtt{exn} \mid A \times B \mid A \to \mathrm{T}B \\ \Gamma & ::= & x_1 : A_1, \ldots, x_n : A_n \\ V & ::= & x \mid () \mid n \mid b \mid E \mid (V_1, V_2) \mid \mathtt{rec}\ f(x : A) \Leftarrow M \\ M, N & ::= & V_1 V_2 \mid \pi_i\, V \mid V_1 + V_2 \mid \mathtt{if}\ V\ \mathtt{then}\ M\ \mathtt{else}\ N \mid \\ & & \mathtt{val}\ V \mid \mathtt{raise}\ V \mid \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ N\ \mathtt{unless}\ H \\ H & ::= & - \mid E \Rightarrow M, H \end{array}$$

Occasionally we find it convenient to treat the handlers as a map and write $\mathrm{dom}(H)$ for the set of exceptions handled by H and $H(E)$ for the handler of exception E. We abbreviate $E \Rightarrow M, -$ as $E \Rightarrow M$. In the case of $E \Rightarrow M, H$ we assume that $E \notin \mathrm{dom}(H)$.

We use some syntactic sugar defined as follows:

$$\begin{array}{rcl} \mathtt{let}\ x \Leftarrow M\ \mathtt{in}\ P & \triangleq & \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ - \\ \lambda x : A.M & \triangleq & \mathtt{rec}\ f(x : A) \Leftarrow M \qquad f \notin \mathrm{FV}(M) \\ \Omega & \triangleq & (\mathtt{rec}\ f(x : \mathtt{unit}) \Leftarrow fx)() \end{array}$$

The type system is defined in Fig. 1. There are two forms of typing judgement: $\Gamma \vdash V : A$ for values and $\Gamma \vdash M : \mathrm{T}A$ for computations. An extract from the operational semantics is shown in Fig. 2. The judgement $M \Downarrow V$ means that the closed term M evaluates to the value V, whilst $M \uparrow E$ means that M raises the exception E.

Lemma 1. *If* $\vdash M : \mathrm{T}A$ *and* $M \Downarrow V$ *then* $\vdash V : A$. □

$\boxed{M \Downarrow V}$

$$\frac{M \Downarrow V \quad P[V/x] \Downarrow V'}{\texttt{try}\ x \Leftarrow M\ \texttt{in}\ P\ \texttt{unless}\ H \Downarrow V'}$$

$$\frac{M \uparrow E \quad N \Downarrow V}{\texttt{try}\ x \Leftarrow M\ \texttt{in}\ P\ \texttt{unless}\ H \Downarrow V}\ H(E) = N$$

$\boxed{M \uparrow E}$

$$\frac{M \Downarrow V \quad P[V/x] \uparrow V'}{\texttt{try}\ x \Leftarrow M\ \texttt{in}\ P\ \texttt{unless}\ H \uparrow V'}$$

$$\frac{M \uparrow E \quad N \uparrow V}{\texttt{try}\ x \Leftarrow M\ \texttt{in}\ P\ \texttt{unless}\ H \uparrow V}\ H(E) = N$$

$$\frac{M \uparrow E}{\texttt{try}\ x \Leftarrow M\ \texttt{in}\ P\ \texttt{unless}\ H \uparrow E}\ E \notin \mathrm{dom}(H)$$

Figure 2. Natural operational semantics (extract)

We give a denotational semantics of the language in the category of ω-cpos (predomains). The semantics of types is as follows:

$$[\![\texttt{unit}]\!] = \mathbf{1} \qquad [\![\texttt{int}]\!] = \mathbb{Z} \qquad [\![\texttt{bool}]\!] = \mathbb{B} \qquad [\![\texttt{exn}]\!] = \mathbb{E}$$
$$[\![A \times B]\!] = ([\![A]\!] \times [\![B]\!]) \qquad [\![A \to \mathrm{T}B]\!] = ([\![A]\!] \Rightarrow [\![\mathrm{T}B]\!])$$
$$[\![\mathrm{T}A]\!] = ([\![A]\!] + \mathbb{E})_\perp$$

where $D \Rightarrow E$ is the predomain of continuous functions from D to E. We define $\lfloor\cdot\rfloor : D \Rightarrow D_\perp$ to be the constructor of the lift monad where $D_\perp = \{\lfloor d \rfloor \mid d \in D\} \cup \{\perp\}$ and $\lfloor d \rfloor \sqsubseteq \lfloor e \rfloor$ if $d \sqsubseteq e$ and $\forall x. \perp \sqsubseteq x$. The computation type constructor is the usual lifted exceptions monad (separated sum). The semantics of contexts is given by

$$[\![x_1 : A_1, \ldots, x_n : A_n]\!] \quad = \quad [\![A_1]\!] \times \cdots \times [\![A_n]\!]$$

The semantics of judgements for values, computations and handlers

$$\begin{array}{rcl} [\![\Gamma \vdash V : A]\!] & : & [\![\Gamma]\!] \to [\![A]\!] \\ [\![\Gamma \vdash M : \mathrm{T}A]\!] & : & [\![\Gamma]\!] \to [\![\mathrm{T}A]\!] \\ [\![\Gamma \vdash H : \mathrm{T}A]\!] & : & [\![\Gamma]\!] \to \mathbb{E} \to [\![\mathrm{T}A]\!] \end{array}$$

is defined by structural induction on the simple type system. The following extract shows the more interesting cases:
Computations:

$$[\![\Gamma \vdash \texttt{val}\ V : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inl}\, [\![\Gamma \vdash V : A]\!]\rho \rfloor$$
$$[\![\Gamma \vdash \texttt{raise}\ V : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inr}\, [\![\Gamma \vdash V : \texttt{exn}]\!]\rho \rfloor$$

$[\![\Gamma \vdash \texttt{try}\ x \Leftarrow M\ \texttt{in}\ P\ \texttt{unless}\ H : \mathrm{T}B]\!]\rho =$

$$\begin{cases} [\![\Gamma, x : A \vdash P : \mathrm{T}B]\!]\rho[x \mapsto v] & \text{if } [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inl}\, v \rfloor \\ [\![\Gamma \vdash H : \mathrm{T}B]\!]\rho E & \text{if } [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inr}\, E \rfloor \\ \perp & \text{if } [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \perp \end{cases}$$

Handlers:

$$[\![\Gamma \vdash - : \mathrm{T}A]\!]\rho E = \lfloor \mathrm{inr}\, E \rfloor$$

$$[\![\Gamma \vdash E \Rightarrow M, H : \mathrm{T}A]\!]\rho E' = \begin{cases} [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho & \text{if } E' = E \\ [\![\Gamma \vdash H : \mathrm{T}A]\!]\rho E' & \text{else} \end{cases}$$

Definition 1 (Typed Contexts)**.** *We define computation contexts as 'terms with a hole' in the usual way and write*

$$\mathcal{C}[\cdot] : (\Gamma \vdash A) \Rightarrow (\Gamma' \vdash \mathrm{T}B)$$

to mean that if $\Gamma \vdash V : A$ *then* $\Gamma' \vdash \mathcal{C}[V] : \mathrm{T}B$.

Definition 2 (Contextual Equivalence)**.** *If* $\Gamma \vdash V : A$ *and* $\Gamma \vdash V' : A$ *then we write* $\Gamma \vdash V \approx V' : A$ *to mean*

$$\forall \mathcal{C}[\cdot] : (\Gamma \vdash A) \Rightarrow (- \vdash \texttt{Tunit}). \mathcal{C}[V] \Downarrow () \iff \mathcal{C}[V'] \Downarrow ()$$

and similarly for computations.

Theorem 1 (Computational Adequacy)**.**
If $\Gamma \vdash V : A$ *and* $\Gamma \vdash V' : A$ *then*

$$[\![\Gamma \vdash V : A]\!] = [\![\Gamma \vdash V' : A]\!] \implies \Gamma \vdash V \approx V' : A$$

and similarly for computations. □

3. Refined Type System

We now present a extended type system that refines the simple types of the base language by annotating each computation type constructor with an *effect*. The effect specifies which exceptions *may* be raised and whether the computation *may* diverge. The refined system also annotates the type of (first-class) exception values themselves and includes an empty type, intersection types and universal quantification over effects.

The syntax of effects, ε, and extended types, X, is as follows, where α ranges over effect variables, $\mathbf{f}$ is the empty set of effects, $\mathbf{t}$ is the top effect (the set of all possible exceptions plus divergence) and φ is the annotation on exception values:

$$\begin{array}{rcl} X, Y & ::= & \texttt{unit} \mid \texttt{int} \mid \texttt{bool} \mid \texttt{exn}_\varphi \mid X \times Y \mid \\ & & X \to \mathrm{T}_\varepsilon Y \mid \mathbf{0}_A \mid X \wedge Y \mid \forall \alpha. X \\ \varphi & \subseteq & \mathbb{E} \\ E & \in & \mathbb{E} \\ \varepsilon & ::= & \alpha \mid \perp \mid E \mid \mathbf{f} \mid \mathbf{t} \mid \varepsilon_1 \cap \varepsilon_2 \mid \varepsilon_1 \oplus \varepsilon_2 \\ \Delta & ::= & \alpha_1, \ldots, \alpha_n \\ \Theta & ::= & x_1 : X_1, \ldots, x_n : X_n \\ \mathrm{Effs} & \triangleq & \mathbb{P}(\mathbb{E} \cup \{\perp\}) \end{array}$$

Effect annotations are built from primitive effect constants and effect variables ('α') by intersection ('$\cap$') and symmetric difference ('$\oplus$') operations. In the absence of effect polymorphism one can define annotations simply to be sets of primitive effects, but the presence of effect variables requires us to make annotations more syntactic, and axiomatize effect equations more explicitly.

3.1 Effect Annotations

The rules defining well-formed effects and effect equations in context are given in Figure 3. The equational rules are a standard presentation of the theory of Boolean rings [15]. In fact the primitives from which effects will be built in the inference system are union ('$\cup$') and set difference ('$\setminus$'), rather than $\cap$ and $\oplus$. In an earlier version of this work we gave a direct axiomatization of entailment in terms of $\cup$ and $\setminus$, but the axioms were a little complex.[1] We instead take the equivalent, slightly more well behaved basis, comprising $\cap$ and $\oplus$ as primitive, and define $\cup$ and $\setminus$ using the following macros:

$$\begin{array}{rcl} \varepsilon_1 \cup \varepsilon_2 & \triangleq & (\varepsilon_1 \oplus \varepsilon_2) \oplus (\varepsilon_1 \cap \varepsilon_2) \\ \varepsilon_1 \setminus \varepsilon_2 & \triangleq & \varepsilon_1 \oplus (\varepsilon_1 \cap \varepsilon_2) \end{array}$$

Write $\varepsilon_1[\varepsilon_2/\alpha]$ for ε_1 with occurrences of α substituted by ε_2, which satisfies the obvious well-formedness property:

Lemma 2. *If* $\Delta, \alpha \vdash \varepsilon_1$ *and* $\Delta \vdash \varepsilon_2$ *then* $\Delta \vdash \varepsilon_1[\varepsilon_2/\alpha]$. □

[1] Nielson, Nielson and Hankin [17] treat '$\setminus$' as a meta-level operation on annotations that use only $\cup$, which loses precision in the interests of simplicity.

$\boxed{\Delta \vdash \varepsilon}$

$$\frac{}{\Delta, \alpha \vdash \alpha} \qquad \frac{}{\Delta \vdash E} \qquad \frac{}{\Delta \vdash \bot} \qquad \frac{}{\Delta \vdash \mathbf{f}} \qquad \frac{}{\Delta \vdash \mathbf{t}}$$

$$\frac{\Delta \vdash \varepsilon_1 \quad \Delta \vdash \varepsilon_2}{\Delta \vdash \varepsilon_1 \cap \varepsilon_2} \qquad \frac{\Delta \vdash \varepsilon_1 \quad \Delta \vdash \varepsilon_2}{\Delta \vdash \varepsilon_1 \oplus \varepsilon_2}$$

$\boxed{\Delta \vdash \varepsilon = \varepsilon'}$ + equivalence and congruence rules

$$\frac{\Delta \vdash \varepsilon_1 \quad \Delta \vdash \varepsilon_2 \quad \Delta \vdash \varepsilon_3}{\Delta \vdash \varepsilon_1 \oplus (\varepsilon_2 \oplus \varepsilon_3) = (\varepsilon_1 \oplus \varepsilon_2) \oplus \varepsilon_3}$$

$$\frac{\Delta \vdash \varepsilon}{\Delta \vdash \varepsilon_1 \oplus \varepsilon_2 = \varepsilon_2 \oplus \varepsilon_1} \qquad \frac{\Delta \vdash \varepsilon}{\Delta \vdash \varepsilon \oplus \mathbf{f} = \varepsilon} \qquad \frac{\Delta \vdash \varepsilon}{\Delta \vdash \varepsilon \oplus \varepsilon = \mathbf{f}}$$

$$\frac{\Delta \vdash \varepsilon_1 \quad \Delta \vdash \varepsilon_2 \quad \Delta \vdash \varepsilon_3}{\Delta \vdash \varepsilon_1 \cap (\varepsilon_2 \cap \varepsilon_3) = (\varepsilon_1 \cap \varepsilon_2) \cap \varepsilon_3}$$

$$\frac{\Delta \vdash \varepsilon}{\Delta \vdash \varepsilon_1 \cap \varepsilon_2 = \varepsilon_2 \cap \varepsilon_1} \qquad \frac{\Delta \vdash \varepsilon}{\Delta \vdash \varepsilon \cap \mathbf{t} = \varepsilon} \qquad \frac{\Delta \vdash \varepsilon}{\Delta \vdash \varepsilon \cap \varepsilon = \varepsilon}$$

$$\frac{\Delta \vdash \varepsilon_1 \quad \Delta \vdash \varepsilon_2 \quad \Delta \vdash \varepsilon_3}{\Delta \vdash \varepsilon_1 \cap (\varepsilon_2 \oplus \varepsilon_3) = (\varepsilon_1 \cap \varepsilon_2) \oplus (\varepsilon_1 \cap \varepsilon_3)}$$

Figure 3. Well-formed effects and effect equations

The meaning of an effect context is an n-tuple of semantic effects:

$$[\![\alpha_1, \ldots, \alpha_n]\!] = \mathrm{Effs}^n$$

The semantics of effects in context

$$[\![\Delta \vdash \varepsilon]\!] \in [\![\Delta]\!] \to \mathrm{Effs}$$

is defined inductively as follows:

$$\begin{array}{rcl}
[\![\Delta \vdash E]\!]\eta & = & \{E\} \\
[\![\Delta \vdash \bot]\!]\eta & = & \{\bot\} \\
[\![\Delta \vdash \mathbf{f}]\!]\eta & = & \emptyset \\
[\![\Delta \vdash \mathbf{t}]\!]\eta & = & \mathbb{E} \cup \{\bot\} \\
[\![\alpha_1, \ldots, \alpha_n \vdash \alpha_j]\!]\eta & = & \pi_j\, \eta \\
[\![\Delta \vdash \varepsilon_1 \cap \varepsilon_2]\!]\eta & = & [\![\Delta \vdash \varepsilon_1]\!]\eta \cap [\![\Delta \vdash \varepsilon_2]\!]\eta \\
[\![\Delta \vdash \varepsilon_1 \oplus \varepsilon_2]\!]\eta & = & [\![\Delta \vdash \varepsilon_1]\!]\eta \oplus [\![\Delta \vdash \varepsilon_2]\!]\eta
\end{array}$$

Lemma 3. *If $\Delta \vdash \varepsilon_1 = \varepsilon_2$ then $[\![\Delta \vdash \varepsilon_1]\!]\eta = [\![\Delta \vdash \varepsilon_2]\!]\eta$.* □

3.2 Refined Types

The rules defining well-formed extended types are given in Figure 4. The judgement $\Delta \vdash X \lhd A$ means that in the effect context Δ, the extended type X refines the simple type A. Note that our intersection types are really just intersections over the effect annotations since both sides of the '$\wedge$' refine the same simple type. We sometimes write $\Delta \vdash X$ if $\Delta \vdash X \lhd A$ for some A.

Lemma 4. *If $\Delta \vdash X \lhd A$ and $\Delta \vdash X \lhd B$ then $A = B$.* □

We write $X[\varepsilon/\alpha]$ for X with unbound occurrences of α in effect annotations substituted by ε.

$\boxed{\Delta \vdash X \lhd A}$

$$\frac{}{\Delta \vdash \mathtt{unit} \lhd \mathtt{unit}} \qquad \frac{}{\Delta \vdash \mathtt{int} \lhd \mathtt{int}} \qquad \frac{}{\Delta \vdash \mathtt{bool} \lhd \mathtt{bool}}$$

$$\frac{}{\Delta \vdash \mathtt{exn}_\varphi \lhd \mathtt{exn}} \qquad \frac{}{\Delta \vdash \mathbf{0}_A \lhd A} \qquad \frac{\Delta \vdash X \lhd A}{\Delta, \alpha \vdash X \lhd A}$$

$$\frac{\Delta \vdash X \lhd A \quad \Delta \vdash Y \lhd B}{\Delta \vdash X \times Y \lhd A \times B} \qquad \frac{\Delta \vdash X \lhd A \quad \Delta \vdash \mathrm{T}_\varepsilon Y \lhd \mathrm{T}B}{\Delta \vdash X \to \mathrm{T}_\varepsilon Y \lhd A \to \mathrm{T}B}$$

$$\frac{\Delta \vdash \varepsilon \quad \Delta \vdash X \lhd A}{\Delta \vdash \mathrm{T}_\varepsilon X \lhd \mathrm{T}A} \qquad \frac{\Delta \vdash X \lhd A \quad \Delta \vdash Y \lhd A}{\Delta \vdash X \wedge Y \lhd A}$$

$$\frac{\Delta, \alpha \vdash X \lhd A}{\Delta \vdash \forall \alpha. X \lhd A}$$

$\boxed{\Delta \vdash \Theta \lhd \Gamma}$

$$\frac{\{\Delta \vdash X_i \lhd A_i\}_{i=1..n}}{\Delta \vdash x_1 : X_1, \ldots, x_n : X_n \lhd x_1 : A_1, \ldots, x_n : A_n}$$

Figure 4. Well-formed types

Definition 3. *Substitution on extended types*

$$\begin{array}{rcl}
\mathtt{unit}[\varepsilon/\alpha] & \triangleq & \mathtt{unit} \qquad \mathtt{int}[\varepsilon/\alpha] \triangleq \mathtt{int} \\
\mathtt{bool}[\varepsilon/\alpha] & \triangleq & \mathtt{bool} \qquad \mathtt{exn}_\varphi[\varepsilon/\alpha] \triangleq \mathtt{exn}_\varphi \\
(X \times Y)[\varepsilon/\alpha] & \triangleq & X[\varepsilon/\alpha] \times Y[\varepsilon/\alpha] \\
(X \to \mathrm{T}_{\varepsilon'} Y)[\varepsilon/\alpha] & \triangleq & X[\varepsilon/\alpha] \to \mathrm{T}_{\varepsilon'} Y[\varepsilon/\alpha] \\
(\mathrm{T}_{\varepsilon'} X)[\varepsilon/\alpha] & \triangleq & \mathrm{T}_{\varepsilon'[\varepsilon/\alpha]} X[\varepsilon/\alpha] \qquad \mathbf{0}_A[\varepsilon/\alpha] \triangleq \mathbf{0}_A \\
(X \wedge Y)[\varepsilon/\alpha] & \triangleq & X[\varepsilon/\alpha] \wedge Y[\varepsilon/\alpha] \\
(\forall \alpha. X)[\varepsilon/\alpha] & \triangleq & \forall \alpha. X \\
(\forall \beta. X)[\varepsilon/\alpha] & \triangleq & \forall \beta. X[\varepsilon/\alpha] \quad \textit{where } \alpha \neq \beta
\end{array}$$

Lemma 5. *If $\Delta, \alpha \vdash X \lhd A$ and $\Delta \vdash \varepsilon$ then $\Delta \vdash X[\varepsilon/\alpha] \lhd A$.* □

We define a subtyping relation on extended types in Figure 5. The judgement $\Delta \vdash X \leq Y \lhd A$ means that in the effect context Δ, the type X is a subtype of Y and they both refine the simple type A. We write $\Delta \vdash X \leq Y$ if $\Delta \vdash X \leq Y \lhd A$ for some A.

The hypothesis $\varepsilon \cap \varepsilon' = \varepsilon$ in the rule for subtyping computation types is just the encoding in terms of our equational laws for Boolean rings of the subeffect condition $\varepsilon \leq \varepsilon'$.

The semantics of each extended type is given by a relation on the semantics of the simple type that it refines, defined in Figure 6, where I_D is the diagonal relation $\{(d, d) \mid d \in D\}$, and if $R_1 \subseteq D_1 \times D_1$ and $R_2 \subseteq D_2 \times D_2$ then

$$\begin{array}{l}
R_1 \times R_2 \;\subseteq\; (D_1 \times D_2) \times (D_1 \times D_2) \\
\quad \triangleq \{((d_1, d_2), (d_1', d_2')) \mid (d_1, d_1') \in R_1,\ (d_2, d_2') \in R_2\}
\end{array}$$

and

$$\begin{array}{l}
R_1 \Rightarrow R_2 \;\subseteq\; (D_1 \Rightarrow D_2) \times (D_1 \Rightarrow D_2) \\
\quad \triangleq \{(f, f') \mid \forall (d, d') \in R_1, (f\, d, f'\, d') \in R_2\}.
\end{array}$$

Notice that the semantics in Figure 6 is just a familiar-looking logical relation [19]. We are essentially giving a Church-style interpretation of the basic type system and a Curry-style interpretation

$\boxed{\Delta \vdash X \leq Y \lhd A}$

$$\frac{\Delta \vdash X \lhd A}{\Delta \vdash X \leq X \lhd A} \qquad \frac{\Delta \vdash X \leq Y \lhd A \quad \Delta \vdash Y \leq Z \lhd A}{\Delta \vdash X \leq Z \lhd A} \qquad \frac{\varphi \subseteq \varphi'}{\Delta \vdash \mathtt{exn}_{\varphi} \leq \mathtt{exn}_{\varphi'} \lhd \mathtt{exn}} \qquad \frac{\Delta \vdash X \lhd A}{\Delta \vdash \mathbf{0}_A \leq X \lhd A}$$

$$\frac{\Delta \vdash X \lhd A}{\Delta, \alpha \vdash X \leq X \lhd A} \qquad \frac{\Delta, \alpha \vdash X \lhd A \quad \Delta \vdash \varepsilon}{\Delta \vdash \forall \alpha. X \leq X[\varepsilon/\alpha] \lhd A} \qquad \frac{\Delta \vdash X \lhd A \quad \Delta, \alpha \vdash X \leq Y \lhd A}{\Delta \vdash X \leq \forall \alpha. Y \lhd A} \qquad \frac{\Delta \vdash X \leq X' \lhd A \quad \Delta \vdash Y \leq Y' \lhd B}{\Delta \vdash X \times Y \leq X' \times Y' \lhd A \times B}$$

$$\frac{\Delta \vdash X' \leq X \lhd A \quad \Delta \vdash \mathrm{T}_{\varepsilon} Y \leq \mathrm{T}_{\varepsilon'} Y' \lhd \mathrm{T} B}{\Delta \vdash (X \to \mathrm{T}_{\varepsilon} Y) \leq (X' \to \mathrm{T}_{\varepsilon'} Y') \lhd A \to \mathrm{T} B} \qquad \frac{\Delta \vdash \varepsilon \cap \varepsilon' = \varepsilon \quad \Delta \vdash X \leq X' \lhd A}{\Delta \vdash \mathrm{T}_{\varepsilon} X \leq \mathrm{T}_{\varepsilon'} X' \lhd \mathrm{T} A} \qquad \frac{\Delta \vdash X \leq Y \lhd A \quad \Delta \vdash X \leq Z \lhd A}{\Delta \vdash X \leq Y \wedge Z \lhd A}$$

$$\frac{\Delta \vdash X \lhd A \quad \Delta \vdash Y \lhd A}{\Delta \vdash X \wedge Y \leq X \lhd A} \qquad \frac{\Delta \vdash X \lhd A \quad \Delta \vdash Y \lhd A}{\Delta \vdash X \wedge Y \leq Y \lhd A} \qquad \frac{\Delta \vdash X \lhd A \quad \Delta \vdash Y \lhd B \quad \Delta \vdash Z \lhd B \quad \Delta \vdash \varepsilon}{\Delta \vdash (X \to \mathrm{T}_{\varepsilon} Y) \wedge (X \to \mathrm{T}_{\varepsilon} Z) \leq X \to \mathrm{T}_{\varepsilon}(Y \wedge Z) \lhd A \to \mathrm{T} B}$$

$\boxed{\Delta \vdash \Theta \leq \Theta' \lhd \Gamma}$

$$\frac{\{\Delta \vdash X_i \leq Y_i \lhd A_i\}_{i=1..n}}{\Delta \vdash x_1 : X_1, \ldots, x_n : X_n \leq x_1 : Y_1, \ldots, x_n : Y_n \lhd x_1 : A_1, \ldots, x_n : A_n}$$

Figure 5. Subtyping extended types

of their refinements, but the fact that we are using standard semantic machinery makes it straightforward to get the 'right' shape for the definitions and theorems, even though the consequences of the definitions at higher types can be surprisingly subtle.

The interpretations of extended types are admissible PERs:

Lemma 6 (Admissibility)**.**

(i) If $\bot \in [\![\Delta \vdash \varepsilon]\!]\eta$ *and* $\eta \in [\![\Delta]\!]$ *then* $(\bot, \bot) \in [\![\Delta \vdash \mathrm{T}_{\varepsilon} X]\!]\eta$

(ii) For all chains $v_0 \sqsubseteq v_1 \sqsubseteq \ldots$ *and* $v_0' \sqsubseteq v_1' \sqsubseteq \ldots$ *in* $[\![A]\!]$*, for all* $\eta \in [\![\Delta]\!]$ *and* $i \geq 0$*,*

$$(v_i, v_i') \in [\![\Delta \vdash X \lhd A]\!]\eta \Rightarrow (\bigsqcup_{i \geq 0} v_i, \bigsqcup_{i \geq 0} v_i') \in [\![\Delta \vdash X \lhd A]\!]\eta$$

and similarly for computations. □

Lemma 7. *For any* Θ*,* X *and* ε *and* $\eta \in [\![\Delta]\!]$*, all of* $[\![\Delta \vdash \Theta]\!]\eta$*,* $[\![\Delta \vdash X]\!]\eta$ *and* $[\![\Delta \vdash \mathrm{T}_{\varepsilon} X]\!]\eta$ *are partial equivalence relations.*

Proof. By induction on extended types. The base cases are trivial, since the identity relation on a set is a PER as is the empty relation. For function, product and computation types we can use the facts that PERs are closed under $\Rightarrow$, $\times$ and $+$. □

One should think of $[\![\Delta \vdash X \lhd A]\!]\eta$ as picking out a subset of $[\![A]\!]$ *together* with a coarser notion of equality. The following sanity check shows that the semantics of the refined type carrying no information coincides with the unrefined semantics, in the sense that the interpretation of the refined type with a top effect everywhere is just equality on the interpretation of its erasure. If we write $G(A)$ for the obvious map from simple types to refined types with the top effect everywhere then:

Lemma 8. *For all* A*, and* $\eta \in [\![\Delta]\!]$*,* $[\![\Delta \vdash G(A) \lhd A]\!]\eta = I_{[\![A]\!]}$*.* □

The subtype relation is sound in the semantics of refined types:

Lemma 9. *If* $\Delta \vdash X \leq Y \lhd A$ *and* $\eta \in [\![\Delta]\!]$ *then*

$$[\![\Delta \vdash X \lhd A]\!]\eta \subseteq [\![\Delta \vdash Y \lhd A]\!]\eta$$

and similarly for computations. □

4. Effect Analysis

The actual effect analysis is shown in Figure 7, where the notation $\ulcorner \cdot \urcorner$ used in the rule for `try` stands for the natural map from finite sets of exceptions to effects:

$$\ulcorner \{E_1, \ldots, E_n\} \urcorner \quad \triangleq \quad E_1 \cup \cdots \cup E_n.$$

The inference rules defining the analysis are not entirely standard, as rather than present single-subject judgements of the form $\Delta\,;\Theta \vdash V : X$ (and similarly for computations), we use equational judgements of the form $\Delta\,;\Theta \vdash V = V' : X$ from the start and define the more conventional unary judgement $\Delta\,;\Theta \vdash V : X$ as a mere abbreviation for $\Delta\,;\Theta \vdash V = V : X$. This is because we will need the more general congruence rules for equality to perform transformations in context anyway, and their soundness strictly subsumes the usual 'fundamental theorem' showing the soundness of the more conventional unary judgements. Furthermore, this presentation allows the addition of further sound equational rules to also add new extended typing judgements – extended types are purely descriptive ('extrinsic', in Reynolds's terminology [20]), unlike the prescriptive ('intrinsic') basic type system.

The 'diagonal' judgements of the form $\Delta\,;\Theta \vdash V = V : X$ (and similarly for computations) that are derivable constitute a fairly unsurprising effect analysis.[2] For example, the rule for `raise` V says that it is a computation that may raise any exception that V might be and never returns a value. The `try` construct has the possible effects of the scrutinee M, with handled exceptions subtracted, unioned with the possible effects of the continuations. There are many possible variations in the details of the presentation; we could, for example, let the handlers and the success continuation each have different effects and union them all explicitly, or only included the effects of handlers covering exceptions appearing in the effect of the scrutinee. In the presence of subtyping and the

[2] Although we are not really concerned with algorithms in this paper, one of the referees did wonder about decidability. It seems intuitively clear that one could recast the diagonal fragment of the rules in Figure 7 as an abstract interpretation in which the abstract domain associated with each simple type is the set of refinements of that type quotiented by $\leq \cap \geq$, ordered by $\leq$. Since the set of equivalence classes at each type is clearly finite (and $\leq$ is decidable), we have little doubt that the analysis is decidable.

$$
\begin{array}{rcl}
[\![\Delta \vdash X \lhd A]\!] & \in & [\![\Delta]\!] \to \mathbb{P}([\![A]\!] \times [\![A]\!]) \\
[\![\Delta \vdash \mathtt{int} \lhd \mathtt{int}]\!]\eta & = & I_{\mathbb{Z}} \\
[\![\Delta \vdash \mathtt{bool} \lhd \mathtt{bool}]\!]\eta & = & I_{\mathbb{B}} \\
[\![\Delta \vdash \mathtt{unit} \lhd \mathtt{unit}]\!]\eta & = & I_{\mathbf{1}} \\
[\![\Delta \vdash \mathtt{exn}_\varphi \lhd \mathtt{exn}]\!]\eta & = & \{(E, E) \mid E \in \varphi\} \\
[\![\Delta \vdash X \times Y \lhd A \times B]\!]\eta & = & [\![\Delta \vdash X \lhd A]\!]\eta \times [\![\Delta \vdash Y \lhd B]\!]\eta \\
[\![\Delta \vdash X \to \mathrm{T}_\varepsilon Y \lhd A \to \mathrm{T}B]\!]\eta & = & [\![\Delta \vdash X \lhd A]\!]\eta \Rightarrow [\![\Delta \vdash \mathrm{T}_\varepsilon Y \lhd \mathrm{T}B]\!]\eta \\
[\![\Delta \vdash \mathrm{T}_\varepsilon X \lhd \mathrm{T}A]\!]\eta & = & \{(\lfloor \mathrm{inl}\, m \rfloor, \lfloor \mathrm{inl}\, m' \rfloor) \mid (m, m') \in [\![\Delta \vdash X \lhd A]\!]\eta\} \\
 & & \cup \{(\lfloor \mathrm{inr}\, E \rfloor, \lfloor \mathrm{inr}\, E \rfloor) \mid E \in [\![\Delta \vdash \varepsilon]\!]\eta\} \\
 & & \cup \{(\bot, \bot) \mid \bot \in [\![\Delta \vdash \varepsilon]\!]\eta\} \\
[\![\Delta \vdash \mathbf{0}_A \lhd A]\!]\eta & = & \emptyset \\
[\![\Delta \vdash X \wedge Y \lhd A]\!]\eta & = & [\![\Delta \vdash X \lhd A]\!]\eta \cap [\![\Delta \vdash Y \lhd A]\!]\eta \\
[\![\Delta \vdash \forall \alpha.X \lhd A]\!]\eta & = & \bigcap_{s \in \mathrm{Effs}} [\![\Delta, \alpha \vdash X \lhd A]\!]\eta[\alpha \mapsto s] \\
 & & \\
[\![\Delta \vdash \Theta \lhd \Gamma]\!] & \in & [\![\Delta]\!] \to \mathbb{P}([\![\Gamma]\!] \times [\![\Gamma]\!]) \\
[\![\Delta \vdash x_1 : X_1, \ldots, x_n : X_n \lhd x_1 : A_1, \ldots, x_n : A_n]\!]\eta & = & \{(\rho, \rho') \mid \forall 1 \le i \le n.(\pi_i(\rho), \pi_i(\rho')) \in [\![\Delta \vdash X_i \lhd A]\!]\eta\}
\end{array}
$$

Figure 6. Semantics of extended types

equational rules we present later (e.g. allowing dead handlers to be removed), most sound variants are derivable.

The treatment of possible divergence in the rule for recursive functions constitutes possibly the weakest termination tester there is, but it does at least allow the '$\bot$' to appear in the right place in the types of curried recursive functions, and gives the right non-divergent type to lambda abstractions encoded as trivial recursive functions.

The following shows that the refined type system does not type any more terms than the original one:

Lemma 10. *If $\Delta \vdash \Theta \lhd \Gamma$, $\Delta \vdash X \lhd A$ and $\Delta\,;\Theta \vdash V : X$ then $\Gamma \vdash V : A$ and similarly for computations.* □

Similarly the following shows that we have not ruled out any typable terms from the original system:

Lemma 11. *If $\Gamma \vdash V : A$ then $-\,; G(\Gamma) \vdash V : G(A)$ and similarly for computations.* □

5. Equivalences

We will now present the remaining equivalences of our system and then prove them sound in the semantics. Figure 8 shows β and η laws for products, function spaces, booleans and the computation type constructor. Figure 9 shows other equivalences that do not explicitly depend on particular effects.

More interesting equivalences are those that depend on particular effect annotations. These are shown in Figure 10.

The **Dead Computation** transformation allows the removal of a computation whose value is not used and which does not have any side-effects (including divergence).

The **Must Raise** transformation allows the replacement of any computation which always raises a particular exception E with $\mathtt{raise}\ E$.

The **Diverging Computation** transformation allows the replacement of any computation which always diverges with Ω (and hence with any other always divergent computation).

The **Commuting Computations** transformation allows the order of two value-independent computations to be swapped provided that either at least one of them has no side-effects, or they both at most allow the possibility of diverging or they both at most raise the same exception (which must be known).

The **Pure Lambda Hoist** transformation allows a computation to be hoisted out of a lambda abstraction provided that it is pure and does not depend on the function's arguments.

The **Dead Handler Elimination** transformation allows the removal of a handler for some exception from a try expression provided that the computation it guards cannot possibly raise that particular exception.

Theorem 2. *All of the equations shown in Figures 7, 8, 9 and 10 are soundly modeled in the semantics:*

(i) If $\Delta\,;\Theta \vdash V = V' : X$, $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$ then

$$([\![\Gamma \vdash V : A]\!]\rho, [\![\Gamma \vdash V' : A]\!]\rho') \in [\![\Delta \vdash X \lhd A]\!]\eta$$

(ii) If $\Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X$, $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$ then

$$([\![\Gamma \vdash M : \mathrm{T}A]\!]\rho, [\![\Gamma \vdash M' : \mathrm{T}A]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_\varepsilon X \lhd \mathrm{T}A]\!]\eta$$

(iii) If $\Delta\,;\Theta \vdash H = H' : \mathrm{T}_\varepsilon X$, $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$ then

$$([\![\Gamma \vdash H : \mathrm{T}A]\!]\rho E, [\![\Gamma \vdash H' : \mathrm{T}A]\!]\rho' E) \in [\![\Delta \vdash \mathrm{T}_\varepsilon X \lhd \mathrm{T}A]\!]\eta$$

Proof. We just present some of the cases involving effects:

Identity Handler Elimination.
Let $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$. We have to show

$$([\![\Gamma \vdash E \Rightarrow \mathtt{raise}\ E, H : \mathrm{T}A]\!]\rho E, [\![\Gamma \vdash H : \mathrm{T}A]\!]\rho' E) \in [\![\Delta \vdash \mathrm{T}_\varepsilon X \lhd \mathrm{T}A]\!]\eta$$

We assumed that each exception occurs at most once inside a handler hence $E \notin \mathrm{dom}(H)$. By definition of the semantics

$$[\![\Gamma \vdash E \Rightarrow \mathtt{raise}\ E, H : \mathrm{T}A]\!]\rho E = [\![\Gamma \vdash H : \mathrm{T}A]\!]\rho E$$

and by assumption

$$([\![\Gamma \vdash H : \mathrm{T}A]\!]\rho E, [\![\Gamma \vdash H : \mathrm{T}A]\!]\rho' E) \in [\![\Delta \vdash \mathrm{T}_\varepsilon X \lhd \mathrm{T}A]\!]\eta$$

so we are done.

$\boxed{\Delta\,;\Theta \vdash V = V' : X}$

$$\frac{\Delta\,;\Theta \vdash V = V' : X}{\Delta\,;\Theta \vdash V' = V : X} \qquad \frac{\Delta\,;\Theta \vdash V = V' : X \quad \Delta\,;\Theta \vdash V' = V'' : X}{\Delta\,;\Theta \vdash V = V'' : X} \qquad \frac{\Delta \vdash \Theta \quad \Delta \vdash X}{\Delta\,;\Theta, x : X \vdash x = x : X}$$

$$\frac{\Delta \vdash \Theta}{\Delta\,;\Theta \vdash () = () : \mathtt{unit}} \qquad \frac{\Delta \vdash \Theta}{\Delta\,;\Theta \vdash n = n : \mathtt{int}} \qquad \frac{\Delta \vdash \Theta}{\Delta\,;\Theta \vdash b = b : \mathtt{bool}} \qquad \frac{\Delta \vdash \Theta}{\Delta\,;\Theta \vdash E = E : \mathtt{exn}_{\{E\}}}$$

$$\frac{\Delta\,;\Theta \vdash V_1 = V_1' : X \quad \Delta\,;\Theta \vdash V_2 = V_2' : Y}{\Delta\,;\Theta \vdash (V_1, V_2) = (V_1', V_2') : X \times Y} \qquad \frac{\Delta\,;\Theta, x : X, f : X \to \mathrm{T}_{\varepsilon \cup \bot} Y \vdash M = M' : \mathrm{T}_\varepsilon Y \quad \Delta \vdash X \lhd A}{\Delta\,;\Theta \vdash (\mathtt{rec}\ f(x : A) \Leftarrow M) = (\mathtt{rec}\ f(x : A) \Leftarrow M') : X \to \mathrm{T}_\varepsilon Y}$$

$$\frac{\Delta\,;\Theta \vdash V = V' : X \quad \Delta \vdash X \leq X'}{\Delta\,;\Theta \vdash V = V' : X'} \qquad \frac{\Delta \vdash \Theta' \leq \Theta \quad \Delta\,;\Theta \vdash V = V' : X}{\Delta\,;\Theta' \vdash V = V' : X} \qquad \frac{\begin{array}{c}\Delta \vdash X \lhd A \qquad \Delta \vdash X' \lhd A \\ \Delta\,;\Theta \vdash V = V' : X \quad \Delta\,;\Theta \vdash V = V' : X'\end{array}}{\Delta\,;\Theta \vdash V = V' : X \wedge X'}$$

$$\frac{\Delta, \alpha\,;\Theta \vdash V = V' : X \quad \Delta \vdash \Theta}{\Delta\,;\Theta \vdash V = V' : \forall \alpha . X} \qquad \frac{\Delta\,;\Theta \vdash V = V' : \forall \alpha . X \quad \Delta \vdash \varepsilon}{\Delta\,;\Theta \vdash V = V' : X[\varepsilon/\alpha]}$$

$\boxed{\Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X}$

$$\frac{\Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X}{\Delta\,;\Theta \vdash M' = M : \mathrm{T}_\varepsilon X} \qquad \frac{\Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta \vdash M' = M'' : \mathrm{T}_\varepsilon X}{\Delta\,;\Theta \vdash M = M'' : \mathrm{T}_\varepsilon X} \qquad \frac{\Delta\,;\Theta \vdash V = V' : X_1 \times X_2}{\Delta\,;\Theta \vdash \pi_i\, V = \pi_i\, V' : \mathrm{T}_{\mathbf{f}} X_i}$$

$$\frac{\Delta\,;\Theta \vdash V_1 = V_1' : \mathtt{int} \quad \Delta\,;\Theta \vdash V_2 = V_2' : \mathtt{int}}{\Delta\,;\Theta \vdash (V_1 + V_2) = (V_1' + V_2') : \mathrm{T}_{\mathbf{f}} \mathtt{int}} \qquad \frac{\Delta\,;\Theta \vdash V_1 = V_1' : X \to \mathrm{T}_\varepsilon Y \quad \Delta\,;\Theta \vdash V_2 = V_2' : X}{\Delta\,;\Theta \vdash V_1 V_2 = V_1' V_2' : \mathrm{T}_\varepsilon Y}$$

$$\frac{\Delta\,;\Theta \vdash V = V' : \mathtt{bool} \quad \Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta \vdash N = N' : \mathrm{T}_{\varepsilon'} X}{\Delta\,;\Theta \vdash (\mathtt{if}\ V\ \mathtt{then}\ M\ \mathtt{else}\ N) = (\mathtt{if}\ V'\ \mathtt{then}\ M'\ \mathtt{else}\ N') : \mathrm{T}_{\varepsilon \cup \varepsilon'} X} \qquad \frac{\Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X \quad \Delta \vdash \mathrm{T}_\varepsilon X \leq \mathrm{T}_{\varepsilon'} X'}{\Delta\,;\Theta \vdash M = M' : \mathrm{T}_{\varepsilon'} X'}$$

$$\frac{\Delta\,;\Theta \vdash V = V' : X}{\Delta\,;\Theta \vdash \mathtt{val}\ V = \mathtt{val}\ V' : \mathrm{T}_{\mathbf{f}} X} \qquad \frac{\Delta\,;\Theta \vdash V = V' : \mathtt{exn}_\varphi}{\Delta\,;\Theta \vdash \mathtt{raise}\ V = \mathtt{raise}\ V' : \mathrm{T}_{\ulcorner \varphi \urcorner} \mathbf{0}_A} \qquad \frac{\Delta \vdash \Theta' \leq \Theta \quad \Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X}{\Delta\,;\Theta' \vdash M = M' : \mathrm{T}_\varepsilon X}$$

$$\frac{\Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta, x : X \vdash P = P' : \mathrm{T}_{\varepsilon'} Y \quad \Delta\,;\Theta \vdash H = H' : \mathrm{T}_{\varepsilon'} Y}{\Delta\,;\Theta \vdash (\mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H) = (\mathtt{try}\ x \Leftarrow M'\ \mathtt{in}\ P'\ \mathtt{unless}\ H') : \mathrm{T}_{(\varepsilon \setminus \ulcorner \mathrm{dom}(H) \urcorner) \cup \varepsilon'} Y}$$

$\boxed{\Delta\,;\Theta \vdash H = H' : \mathrm{T}_\varepsilon X}$

$$\frac{\Delta\,;\Theta \vdash M = M' : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta \vdash H = H' : \mathrm{T}_\varepsilon X}{\Delta\,;\Theta \vdash (E \Rightarrow M, H) = (E \Rightarrow M', H') : \mathrm{T}_\varepsilon X}\ E \in \mathbb{E} \qquad \frac{}{\Delta\,;\Theta \vdash - = - : \mathrm{T}_\varepsilon X}$$

Figure 7. Extended type system

β rules:

$$\frac{\Delta\,;\Theta, x : X, f : X \to \mathrm{T}_{\varepsilon\cup\bot} Y \vdash M : \mathrm{T}_\varepsilon Y \quad \Delta\,;\Theta \vdash V : X \quad \Delta \vdash X \lhd A}{\Delta\,;\Theta \vdash (\mathtt{rec}\, f(x : A) \Leftarrow M)V = M[V/x, \mathtt{rec}\, f(x : A) \Leftarrow M/f] : \mathrm{T}_\varepsilon Y} \qquad \frac{\Delta\,;\Theta \vdash V_1 : X_1 \quad \Delta\,;\Theta \vdash V_2 : X_2}{\Delta\,;\Theta \vdash \pi_i\,(V_1, V_2) = \mathtt{val}\, V_i : \mathrm{T}_{\mathtt{f}} X_i}$$

$$\frac{\Delta\,;\Theta \vdash V : X \quad \Delta\,;\Theta, x : X \vdash P : \mathrm{T}_\varepsilon Y}{\Delta\,;\Theta \vdash (\mathtt{let}\, x \Leftarrow \mathtt{val}\, V\, \mathtt{in}\, P) = P[V/x] : \mathrm{T}_\varepsilon Y} \qquad \frac{\Delta\,;\Theta \vdash V : \mathtt{bool} \quad \Delta\,;\Theta \vdash M = N : \mathrm{T}_\varepsilon X}{\Delta\,;\Theta \vdash (\mathtt{if}\, V\, \mathtt{then}\, M\, \mathtt{else}\, N) = M : \mathrm{T}_\varepsilon X}$$

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta \vdash N : \mathrm{T}_{\varepsilon'} X}{\Delta\,;\Theta \vdash (\mathtt{if}\, \mathtt{true}\, \mathtt{then}\, M\, \mathtt{else}\, N) = M : \mathrm{T}_\varepsilon X} \qquad \frac{\Delta\,;\Theta \vdash M : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta \vdash N : \mathrm{T}_{\varepsilon'} X}{\Delta\,;\Theta \vdash (\mathtt{if}\, \mathtt{false}\, \mathtt{then}\, M\, \mathtt{else}\, N) = N : \mathrm{T}_{\varepsilon'} X}$$

$$\frac{\Delta\,;\Theta, x : X \vdash P : \mathrm{T}_\varepsilon Y \quad \Delta\,;\Theta \vdash E{\Rightarrow}N : \mathrm{T}_\varepsilon Y \quad \Delta\,;\Theta \vdash H : \mathrm{T}_\varepsilon Y}{\Delta\,;\Theta \vdash (\mathtt{try}\, x \Leftarrow \mathtt{raise}\, E\, \mathtt{in}\, P\, \mathtt{unless}\, E{\Rightarrow}N, H) = N : \mathrm{T}_\varepsilon Y}$$

η rules:

$$\frac{\Delta\,;\Theta \vdash V : X \to \mathrm{T}_\varepsilon Y \quad \Delta \vdash X \lhd A}{\Delta\,;\Theta \vdash (\mathtt{rec}\, f(x : A) \Leftarrow Vx) = V : X \to \mathrm{T}_\varepsilon Y} \qquad \frac{\Delta\,;\Theta \vdash M : \mathrm{T}_\varepsilon X}{\Delta\,;\Theta \vdash (\mathtt{try}\, x \Leftarrow M\, \mathtt{in}\, \mathtt{val}\, x\, \mathtt{unless}\, -) = M : \mathrm{T}_\varepsilon X}$$

$$\frac{\Delta\,;\Theta \vdash V : X \times Y}{\Delta\,;\Theta \vdash \mathtt{val}\, V = \mathtt{let}\, x \Leftarrow \pi_1\, V\, \mathtt{in}\, \mathtt{let}\, y \Leftarrow \pi_2\, V\, \mathtt{in}\, \mathtt{val}\, (x, y) : \mathrm{T}_{\mathtt{f}}(X \times Y)}$$

Figure 8. β and η rules

Duplicated Computation:

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta, x : X, y : X \vdash P : \mathrm{T}_{\varepsilon'} Y \quad \Delta\,;\Theta \vdash H : \mathrm{T}_{\varepsilon'} Y}{\Delta\,;\Theta \vdash \begin{array}{l}\mathtt{try}\, x \Leftarrow M\, \mathtt{in}\, (\mathtt{try}\, y \Leftarrow M\, \mathtt{in}\, P\, \mathtt{unless}\, H)\, \mathtt{unless}\, H \\ = \mathtt{try}\, x \Leftarrow M\, \mathtt{in}\, P[x/y]\, \mathtt{unless}\, H\end{array} : \mathrm{T}_{(\varepsilon \setminus \ulcorner \mathrm{dom}(H) \urcorner) \cup \varepsilon'} Y}$$

Identity Handler Elimination:

$$\frac{\Delta\,;\Theta \vdash H : \mathrm{T}_\varepsilon X}{\Delta\,;\Theta \vdash E{\Rightarrow}\mathtt{raise}\, E, H = H : \mathrm{T}_\varepsilon X}$$

Single Exception Value:

$$\frac{\Delta\,;\Theta \vdash V : \mathtt{exn}_{\{E\}}}{\Delta\,;\Theta \vdash V = E : \mathtt{exn}_{\{E\}}}$$

Commuting Conversion 1:

$$\frac{\Delta\,;\Theta \vdash V : \mathtt{bool} \quad \Delta\,;\Theta \vdash M : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta \vdash N : \mathrm{T}_\varepsilon X \quad \Delta\,;\Theta, x : X \vdash P : \mathrm{T}_{\varepsilon'} Y \quad \Delta\,;\Theta \vdash H : \mathrm{T}_{\varepsilon'} Y}{\Delta\,;\Theta \vdash \begin{array}{l}\mathtt{try}\, x \Leftarrow (\mathtt{if}\, V\, \mathtt{then}\, M\, \mathtt{else}\, N)\, \mathtt{in}\, P\, \mathtt{unless}\, H \\ = \mathtt{if}\, V\, \mathtt{then}\, (\mathtt{try}\, x \Leftarrow M\, \mathtt{in}\, P\, \mathtt{unless}\, H)\, \mathtt{else}\, (\mathtt{try}\, x \Leftarrow N\, \mathtt{in}\, P\, \mathtt{unless}\, H)\end{array} : \mathrm{T}_{(\varepsilon \setminus \ulcorner \mathrm{dom}(H) \urcorner) \cup \varepsilon'} Y}$$

Commuting Conversion 2:

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_{\varepsilon_1} Y \quad \Delta\,;\Theta, y : Y \vdash P : \mathrm{T}_{\varepsilon_2} X \quad \Delta\,;\Theta, x : X \vdash P' : \mathrm{T}_{\varepsilon_3} Z \quad \Delta\,;\Theta \vdash H : \mathrm{T}_{\varepsilon_2} X \quad \Delta\,;\Theta \vdash H' : \mathrm{T}_{\varepsilon_3} Z}{\Delta\,;\Theta \vdash \begin{array}{l}\mathtt{try}\, x \Leftarrow (\mathtt{try}\, y \Leftarrow M\, \mathtt{in}\, P\, \mathtt{unless}\, H)\, \mathtt{in}\, P'\, \mathtt{unless}\, H' \\ = \mathtt{try}\, y \Leftarrow M\, \mathtt{in}\, (\mathtt{try}\, x \Leftarrow P\, \mathtt{in}\, P'\, \mathtt{unless}\, H')\, \mathtt{unless} \\ \{E{\Rightarrow}\mathtt{try}\, x \Leftarrow N\, \mathtt{in}\, P'\, \mathtt{unless}\, H' \mid (E{\Rightarrow}N) \in H\} \\ \cup \{E{\Rightarrow}N \mid (E{\Rightarrow}N) \in H' \;\&\; E \notin \mathrm{dom}(H)\}\end{array} : \mathrm{T}_{(((\varepsilon_1 \setminus \ulcorner \mathrm{dom}(H) \urcorner) \cup \varepsilon_2) \setminus \ulcorner \mathrm{dom}(H') \urcorner) \cup \varepsilon_3} Z}$$

Figure 9. Effect-independent equivalences

Dead Computation:

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_{\mathbf{f}} X \quad \Delta\,;\Theta \vdash P : \mathrm{T}_{\varepsilon} Y}{\Delta\,;\Theta \vdash \mathtt{let}\ x \Leftarrow M\ \mathtt{in}\ P = P : \mathrm{T}_{\varepsilon} Y}\ x \notin \mathrm{dom}(\Theta)$$

Must Raise:

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_{E} \mathbf{0}_A}{\Delta\,;\Theta \vdash M = \mathtt{raise}\ E : \mathrm{T}_{E} \mathbf{0}_A}$$

Diverging Computation:

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_{\bot} \mathbf{0}_A}{\Delta\,;\Theta \vdash M = \Omega : \mathrm{T}_{\bot} \mathbf{0}_A}$$

Commuting Computations:

$$\frac{\begin{array}{c}\Delta\,;\Theta \vdash M_1 : \mathrm{T}_{\varepsilon_1} X_1 \qquad \Delta\,;\Theta \vdash M_2 : \mathrm{T}_{\varepsilon_2} X_2 \\ \Delta\,;\Theta, x_1 : X_1, x_2 : X_2 \vdash N : \mathrm{T}_{\varepsilon_3} Y \qquad \Delta\,;\Theta \vdash H : \mathrm{T}_{\varepsilon_3} Y\end{array}}{\Delta\,;\Theta \vdash \begin{array}{l}\mathtt{try}\ x_1 \Leftarrow M_1\ \mathtt{in}\ (\mathtt{try}\ x_2 \Leftarrow M_2\ \mathtt{in}\ N\ \mathtt{unless}\ H)\ \mathtt{unless}\ H \\ = \mathtt{try}\ x_2 \Leftarrow M_2\ \mathtt{in}\ (\mathtt{try}\ x_1 \Leftarrow M_1\ \mathtt{in}\ N\ \mathtt{unless}\ H)\ \mathtt{unless}\ H\end{array} : \mathrm{T}_{((\varepsilon_1 \cup \varepsilon_2) \setminus \ulcorner \mathrm{dom}(H) \urcorner) \cup \varepsilon_3} Y}$$

$\Delta \vdash \varepsilon_1 = E$ & $\Delta \vdash \varepsilon_2 = E$
or $\Delta \vdash \varepsilon_1 = \bot$ & $\Delta \vdash \varepsilon_2 = \bot$
or $\Delta \vdash \varepsilon_1 = \mathbf{f}$

Pure Lambda Hoist:

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_{\mathbf{f}} Z \quad \Delta\,;\Theta, x : X, f : X \to \mathrm{T}_{\varepsilon \cup \bot} Y, z : Z \vdash P : \mathrm{T}_{\varepsilon} Y \quad \Delta \vdash X \lhd A}{\Delta\,;\Theta \vdash \begin{array}{l}\mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ P) \\ = \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ \mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow P)\end{array} : \mathrm{T}_{\mathbf{f}} (X \to \mathrm{T}_{\varepsilon} Y)}$$

Dead Handler Elimination:

$$\frac{\Delta\,;\Theta \vdash M : \mathrm{T}_{\varepsilon \setminus E} X \quad \Delta\,;\Theta, x : X \vdash P : \mathrm{T}_{\varepsilon'} Y \quad \Delta\,;\Theta \vdash E \Rightarrow N : \mathrm{T}_{\varepsilon''} Y \quad \Delta\,;\Theta \vdash H : \mathrm{T}_{\varepsilon'} Y}{\Delta\,;\Theta \vdash \begin{array}{l}\mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ E \Rightarrow N, H \\ = \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H\end{array} : \mathrm{T}_{((\varepsilon \setminus \ulcorner \mathrm{dom}(H) \urcorner) \setminus E) \cup \varepsilon'} Y}$$

Empty:

$$\frac{\Delta\,;\Theta, x : \mathbf{0}_A \vdash M : \mathrm{T}_{\varepsilon} Y \quad \Delta\,;\Theta, x : \mathbf{0}_A \vdash N : \mathrm{T}_{\varepsilon'} Y}{\Delta\,;\Theta, x : \mathbf{0}_A \vdash M = N : \mathrm{T}_{\varepsilon''} Y}$$

Figure 10. Effect-dependent equivalences

Single Exception Value.
Let $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$. We have to show

$$([\![\Gamma \vdash V : \mathtt{exn}]\!]\rho, [\![\Gamma \vdash E : \mathtt{exn}]\!]\rho') \in [\![\Delta \vdash \mathtt{exn}_{\{E\}} \lhd \mathtt{exn}]\!]\eta \in \{(E, E)\}$$

By definition of the semantics $[\![\Gamma \vdash E : \mathtt{exn}]\!]\rho' = E$ and by assumption

$$([\![\Gamma \vdash V : \mathtt{exn}]\!]\rho, [\![\Gamma \vdash V : \mathtt{exn}]\!]\rho') \in [\![\Delta \vdash \mathtt{exn}_{\{E\}} \lhd \mathtt{exn}]\!]\eta$$

hence $[\![\Gamma \vdash V : \mathtt{exn}]\!]\rho = E$ so we are done.

Dead Computation.
Let $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$. We have to show

$$([\![\Gamma \vdash \mathtt{let}\ x \Leftarrow M\ \mathtt{in}\ P : \mathrm{T}B]\!]\rho, [\![\Gamma \vdash P : \mathrm{T}B]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{\varepsilon} Y \lhd \mathrm{T}B]\!]\eta$$

By assumption

$$([\![\Gamma \vdash M : \mathrm{T}A]\!]\rho, [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{\mathbf{f}} X \lhd \mathrm{T}A]\!]\eta$$

This means there exists $(m, m') \in [\![\Delta \vdash X \lhd A]\!]\eta$ and that $[\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inl}\, m \rfloor$ so by definition of the semantics and the assumption that $x \notin \mathrm{dom}(\Theta)$ we have

$$[\![\Gamma \vdash \mathtt{let}\ x \Leftarrow M\ \mathtt{in}\ P : \mathrm{T}B]\!]\rho = [\![\Gamma \vdash P : \mathrm{T}B]\!]\rho$$

and by assumption

$$([\![\Gamma \vdash P : \mathrm{T}B]\!]\rho, [\![\Gamma \vdash P : \mathrm{T}B]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{\varepsilon} Y \lhd \mathrm{T}B]\!]\eta$$

so we are done.

Must Raise.
Let $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$. We have to show

$$([\![\Gamma \vdash M : \mathrm{T}A]\!]\rho, [\![\Gamma \vdash \mathtt{raise}\ E : \mathrm{T}A]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{E} \mathbf{0}_A \lhd \mathrm{T}A]\!]\eta \in \{(\lfloor \mathrm{inr}\, E \rfloor, \lfloor \mathrm{inr}\, E \rfloor)\}$$

By definition of the semantics we have $[\![\Gamma \vdash \mathtt{raise}\ E : \mathrm{T}A]\!]\rho' = \lfloor \mathrm{inr}\, E \rfloor$. By assumption

$$([\![\Gamma \vdash M : \mathrm{T}A]\!]\rho, [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{E} \mathbf{0}_A \lhd \mathrm{T}A]\!]\eta$$

This means that $[\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inr}\, E \rfloor$ so we are done.

Diverging Computation.
Let $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$. We have to show

$$([\![\Gamma \vdash M : \mathrm{T}A]\!]\rho, [\![\Gamma \vdash \Omega : \mathrm{T}A]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{\bot} \mathbf{0}_A \lhd \mathrm{T}A]\!]\eta \in \{(\bot, \bot)\}$$

By assumption

$$([\![\Gamma \vdash M : \mathrm{T}A]\!]\rho, [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{\bot} \mathbf{0}_A \lhd \mathrm{T}A]\!]\eta$$

hence $[\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \bot$. By definition of the semantics

$$[\![\Gamma \vdash \Omega : \mathrm{T}A]\!]\rho' = (\bigsqcup_{i \geq 0} f_i) * = \bot$$

where f_i is defined by

$$\begin{array}{rcl} f_0 & = & \lambda x' \in \mathbf{1}.\bot_{\mathrm{T}A} \\ f_{i+1} & = & \lambda x' \in \mathbf{1}.(f_i)(x') \end{array}$$

so we are done.

Pure Lambda Hoist.
Let $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$. We have to show

$$\begin{array}{l}([\![\Gamma \vdash \mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ P) : \mathrm{T}(A \to \mathrm{T}B)]\!]\rho, \\ [\![\Gamma \vdash \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ \mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow P) : \mathrm{T}(A \to \mathrm{T}B)]\!]\rho') \\ \in [\![\Delta \vdash \mathrm{T}_{\mathbf{f}} (X \to \mathrm{T}_{\varepsilon} Y) \lhd \mathrm{T}(A \to \mathrm{T}B)]\!]\eta\end{array}$$

By assumption

$$\begin{array}{l}([\![\Gamma \vdash M : \mathrm{T}C]\!]\rho, [\![\Gamma \vdash M : \mathrm{T}C]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{\mathbf{f}} Z \lhd \mathrm{T}C]\!]\eta \\ \in \{(\lfloor \mathrm{inl}\, m \rfloor, \lfloor \mathrm{inl}\, m' \rfloor) \mid (m, m') \in [\![\Delta \vdash Z \lhd C]\!]\eta\}\end{array}$$

Hence we have $(m, m') \in [\![\Delta \vdash Z \lhd C]\!]\eta$ and $[\![\Gamma \vdash M : \mathrm{T}C]\!]\rho = \lfloor \mathrm{inl}\, m \rfloor$ so by definition of the semantics

$$\begin{array}{l}[\![\Gamma \vdash \mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ P) : \mathrm{T}(A \to \mathrm{T}B)]\!]\rho \\ = [\![\Gamma \vdash \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ \mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow P) : \mathrm{T}(A \to \mathrm{T}B)]\!]\rho\end{array}$$

By assumption

$$\begin{array}{l}([\![\Gamma \vdash \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ \mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow P) : \mathrm{T}(A \to \mathrm{T}B)]\!]\rho, \\ [\![\Gamma \vdash \mathtt{let}\ z \Leftarrow M\ \mathtt{in}\ \mathtt{val}\ (\mathtt{rec}\ f(x : A) \Leftarrow P) : \mathrm{T}(A \to \mathrm{T}B)]\!]\rho') \\ \in [\![\Delta \vdash \mathrm{T}_{\mathbf{f}} (X \to \mathrm{T}_{\varepsilon} Y) \lhd \mathrm{T}(A \to \mathrm{T}B)]\!]\eta\end{array}$$

so we are done.

Dead Handler Elimination.
Let $\eta \in [\![\Delta]\!]$ and $(\rho, \rho') \in [\![\Delta \vdash \Theta \lhd \Gamma]\!]\eta$. We have to show

$$(\begin{array}{l}[\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ E \Rightarrow N, H : \mathrm{T}B]\!]\rho, \\ [\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H : \mathrm{T}B]\!]\rho') \\ \in [\![\Delta \vdash \mathrm{T}_{((\varepsilon \setminus \ulcorner \mathrm{dom}(H) \urcorner) \setminus E) \cup \varepsilon'} Y \lhd \mathrm{T}B]\!]\eta\end{array}$$

Assume

$$([\![\Gamma \vdash M : \mathrm{T}A]\!]\rho, [\![\Gamma \vdash M : \mathrm{T}A]\!]\rho') \in [\![\Delta \vdash \mathrm{T}_{\varepsilon \setminus E} X \lhd \mathrm{T}A]\!]\eta$$

Take the case where $[\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inl}\, m \rfloor$ then by definition of the semantics

$$\begin{array}{lll} & [\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ E \Rightarrow N, H : \mathrm{T}B]\!]\rho \\ = & [\![\Gamma, x : A \vdash P : \mathrm{T}B]\!]\rho[x \mapsto m] \\ = & [\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H : \mathrm{T}B]\!]\rho \end{array}$$

In the case where $[\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \lfloor \mathrm{inr}\, E' \rfloor$ by the condition on ε we have $E \neq E'$ so by definition of the semantics

$$[\![\Gamma \vdash E \Rightarrow N, H : \mathrm{T}B]\!]\rho E' = [\![\Gamma \vdash H : \mathrm{T}B]\!]\rho E'$$

In the case where $[\![\Gamma \vdash M : \mathrm{T}A]\!]\rho = \bot$ then by definition of the semantics

$$\begin{array}{lll} & [\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ E \Rightarrow N, H : \mathrm{T}B]\!]\rho \\ = & \bot \\ = & [\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H : \mathrm{T}B]\!]\rho \end{array}$$

By assumption

$$(\begin{array}{l}[\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H : \mathrm{T}B]\!]\rho, \\ [\![\Gamma \vdash \mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H : \mathrm{T}B]\!]\rho') \\ \in [\![\Delta \vdash \mathrm{T}_{((\varepsilon \setminus \ulcorner \mathrm{dom}(H) \urcorner) \setminus E) \cup \varepsilon'} Y \lhd \mathrm{T}B]\!]\eta\end{array}$$

so we are done.

Empty.
Immediate from the fact that $[\![\Delta \vdash \Theta, x : \mathbf{0}_A \lhd \Gamma, x : A]\!]\eta$ is empty. □

6. Examples

Equality is type-dependent. To emphasise that our notion of equality really is type-dependent, note that if we write F_1 for

$$\lambda f{:}(\mathtt{int} \to \mathrm{T}\mathtt{int}).\mathtt{let}\ x \Leftarrow f(1)\ \mathtt{in}\ \mathtt{let}\ y \Leftarrow f(2)\ \mathtt{in}\ \mathtt{val}\ (x, y)$$

and F_2 for

$$\lambda f{:}(\mathtt{int} \to \mathrm{T}\mathtt{int}).\mathtt{let}\ y \Leftarrow f(2)\ \mathtt{in}\ \mathtt{let}\ x \Leftarrow f(1)\ \mathtt{in}\ \mathtt{val}\ (x, y)$$

then we can show

$$\vdash F_1 = F_2 : (\mathtt{int} \to \mathrm{T}_E \mathtt{int}) \to \mathrm{T}_E(\mathtt{int} \times \mathtt{int})$$

but *not*

$$\vdash F_1 = F_2 : (\mathtt{int} \to \mathrm{T}_{E_1 \cup E_2} \mathtt{int}) \to \mathrm{T}_{E_1 \cup E_2}(\mathtt{int} \times \mathtt{int})$$

even though $F_1{=}F_1$ and $F_2{=}F_2$ at the type in the latter judgement.

Empty types. As an example of the use of empty types write F for

$$\mathtt{rec}\ f(x : \mathtt{int}) \Leftarrow \mathtt{if}\ x = 0\ \mathtt{then}\ \mathtt{raise}\ E\ \mathtt{else}\ f(x - 1)$$

then we can show that

$$\vdash F : \mathtt{int} \to \mathrm{T}_{\bot \cup E} \mathbf{0}_{\mathtt{int}}$$

and hence for any appropriately typed P, P', H, H' and $N : \mathrm{T}_\varepsilon X$

$$n : \mathtt{int} \vdash \begin{array}{l}\mathtt{try}\ x \Leftarrow F(n)\ \mathtt{in}\ P\ \mathtt{unless}\ E \Rightarrow N, H \\ = \mathtt{try}\ x \Leftarrow F(n)\ \mathtt{in}\ P'\ \mathtt{unless}\ E \Rightarrow N, H'\end{array} : \mathrm{T}_{\varepsilon \cup \bot} X$$

Effect polymorphism and conjunction. The following example shows how the combination of effect polymorphism and conjunction allows our system to derive surprisingly accurate information:

$$\begin{array}{l} g : \mathtt{unit} \to \mathrm{T}_\beta \mathtt{int} \vdash \\ \lambda f : (\mathtt{unit} \to \mathrm{T}\mathtt{int}).\mathtt{try}\ x \Leftarrow f()\ \mathtt{in}\ \mathtt{val}\ x\ \mathtt{unless}\ E \Rightarrow g() \\ : \forall \alpha.(\mathtt{unit} \to \mathrm{T}_{\alpha \setminus E} \mathtt{int}) \to \mathrm{T}_{\alpha \setminus E} \mathtt{int} \\ \quad \wedge \forall \alpha.(\mathtt{unit} \to \mathrm{T}_{E \cup \alpha} \mathtt{int}) \to \mathrm{T}_{\beta \cup \alpha} \mathtt{int} \end{array}$$

In the refined type of the expression above, the left side of the intersection type deals with the case when f is known not to raise E. In this case the effect of the entire expression is just that of f. The right side deals with the case when f may raise E and hence the effect of the whole expression is the effect of f (without E) together with that of g (which is called by the handler). Notice that in both cases we are polymorphic in the concrete effects in question.

Derived Equivalences. As well as equivalences involving specific terms, there are of course more generic equations that may be justified as derived or admissible rules from the primitive ones we have presented. For example:

$$\frac{\Theta \vdash M : \mathrm{T}_\varepsilon \mathbf{0}_A \quad \Theta, x : X \vdash P : \mathrm{T}_{\varepsilon'} Y \quad \Theta \vdash H : \mathrm{T}_{\varepsilon'} Y}{\Theta \vdash (\mathtt{try}\ x \Leftarrow M\ \mathtt{in}\ P\ \mathtt{unless}\ H) = M : \mathrm{T}_\varepsilon \mathbf{0}_A}$$

with the side condition that none of the exceptions in the effect of M are handled by H. One can derive this by repeatedly applying the dead handler elimination followed by the empty rule (since x has type $\mathbf{0}_A$) to replace P with $\mathtt{val}\ x$, and finally applying the η rule for the computation type constructor.

7. Discussion

We have shown how a simple effect analysis for exceptions and divergence, and the transformations it enables, may be given a simple and neat semantics using partial equivalence relations over the semantics of the original language. That the development is not especially complex or surprising we regard as a feature not a bug: using the right tools should make simple things simple.[3]

Of the three general advantages of the relational approach that were mentioned in the introduction ('extensionalising' apparently intensional properties, capturing dependency, and dealing with equations in context), only the last is really demonstrated here, though that alone certainly justifies formulating the semantics this way. Exception throwing behaviour is not so inherently relational as information flow or data abstraction, and there isn't really a distinct notion of 'observably' throwing an exception, as opposed to just 'throwing an exception'.

Since the work of Amadio [3], Cardone [8] and Abadi and Plotkin [2], partial equivalence relations have been used to give semantics to many different 'standard' type systems. They have also been used in modelling static analyses that are obviously concerned with dependency, such as binding time analysis [11], dead-code elimination [9] and secure information flow [1, 21]. That they also naturally justify optimizing transformations that are predicated on analysis results seems to have only been explicitly stated more recently. Our use of the theory of Boolean rings in axiomatizing subtyping on effects is inspired by Kennedy's work [13] on typing record operations.

Many other researchers have considered static analyses of exception-throwing behaviour, including Yi [24] and Leroy and Pessaux [18], and have even combined them with other analyses

[3] And complex things possible, of course. Our other work on relational reasoning about encapsulated state indicates that these techniques do indeed scale along the complexity axis.

such as resource usage analysis [12]. Much of this previous work has been aimed at providing improved static debugging support, rather than optimizing compilation; as far as we are aware, this is the first work explicitly to deal with the program transformations that are enabled by exception analysis, except for the earlier work by Benton and Kennedy [5], which was semantically *considerably* messier than the presentation used here.

Acknowledgments

The second author is supported by a Microsoft Research scholarship. Thanks also to Andrew Kennedy for many useful discussions and suggestions.

References

[1] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL)*, 1999.

[2] M. Abadi and G. D. Plotkin. A PER model of polymorphism and recursive types. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, June 1990.

[3] R. M. Amadio. Recursion over realizability structures. *Information and Computation*, 91(1), 1991.

[4] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *Proceedings of the 31st ACM Symposium on Principles of Programming Languages (POPL)*, January 2004. Revised version available from `http://research.microsoft.com/~nick/publications.htm`.

[5] N. Benton and A. Kennedy. Monads, effects and transformations. In *3rd International Workshop on Higher Order Operational Techniques in Semantics (HOOTS), Paris*, volume 26 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 1999.

[6] N. Benton and A. Kennedy. Exceptional syntax. *Journal of Functional Programming*, 11(4), 2001.

[7] N. Benton, A. Kennedy, M. Hofmann, and L. Beringer. Reading, writing and relations: Towards extensional semantics for effect analyses. In *Proceedings of the Fourth Asian Symposium on Programming Languages and Systems (APLAS)*, volume 4279 of *Lecture Notes in Computer Science*, 2006.

[8] F. Cardone. Relational semantics for recursive types and bounded quantification. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 372 of *Lecture Notes in Computer Science*. Springer, 1989.

[9] F. Damiani and P. Giannini. Automatic useless-code detection and elimination for HOT functional programs. *Journal of Functional Programming*, 10(6), 2000.

[10] J. Gosling, B. Joy, and G. Steele. *The Java(TM) Language Specification*. Addison Wesley, 1996.

[11] S. Hunt and D. Sands. Binding time analysis: A new PERspective. In *Proceedings ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, June 1991.

[12] F. Iwama, A. Igarashi, and N. Kobayashi. Resource usage analysis for a functional language with exceptions. In *Proceedings of the 2006 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, 2006.

[13] A. J. Kennedy. Type inference and equational theories. Technical Report LIX/RR/96/09, École Polytechnique, 1996.

[14] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *Proceedings of the 15th Annual ACM Symposium on Principles of Programming Languages (POPL)*, 1988.

[15] U. Martin and T. Nipkow. Boolean unification – the story so far. *Journal of Symbolic Computation*, 7(3/4), 1989.

[16] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[17] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

[18] F. Pessaux and X. Leroy. Type-based analysis of uncaught exceptions. In *Proceedings of the 26th ACM Symposium on Principles of programming languages (POPL)*, 1999.

[19] G. D. Plotkin. Lambda definability and logical relations. Technical report, Department of AI, University of Edinburgh, 1973.

[20] J. C. Reynolds. The meaning of types – from intrinsic to extrinsic semantics. Technical Report BRICS RS-00-32, BRICS, University of Aarhus, December 2000.

[21] A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 14(1), March 2001.

[22] J.-P. Talpin and P. Jouvelot. The type and effect discipline. *Information and Computation*, 111(2), June 1994.

[23] P. Wadler and P. Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Logic*, 4(1), 2003.

[24] K. Yi. Compile-time detection of uncaught exceptions in Standard ML programs. In *Proceedings of the 1st International Static Analysis Symposium (SAS)*, volume 864 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.