Refinement Types for Computational Effects: An Algebraic Approach

Danel Ahman, Gordon Plotkin
Laboratory for Foundations of Computer Science

THE UNIVERSITY of EDINBURGH

Introduction

The type systems of many programming languages are too weak to encode meaningful specifications in the types of programs, that can be checked statically during type checking. Refinement types are one of the standard methods for extending a type system, to accommodate such detailed specifications. Unfortunately, there does not yet exist a general theory for combining refinement types and computational effects. The existing attempts only try to combine refinement types with specific effects, or only allow weak specifications. In this work we attempt to address these issues.

We investigate an algebraic treatment of propositional refinement types for languages with computational effects, and develop a refinement typed fine-grain call-by-value metalanguage for such refinements. The language has both effect and value refinement types. The former include specifications of computations by sets of algebraic terms, via a fragment of the modal μ-calculus; the latter allow the specification of pure values.

The algebraic approach allows one to treat refinements modularly and compose them with ease. Our leading examples of specifications using refinements include communication sessions, the correct use of files, Hoare Logic, and effect set annotations, as found in type-and-effect systems.

Our refinement type system is equipped with a relational denotational semantics, which allows one to express effect-dependent optimizations as equations between refinement typed terms.

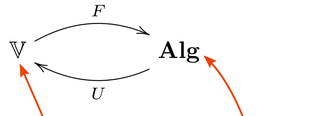
Algebraic Effects

Many computational effects (e.g., state, exceptions, I/O, non-determinism, probabilistic choice) can be presented algebraically [Plotkin & Power].

An **algebraic presentation** (Σ,Eq) consists of
- a set Σ of operations $\mathrm{op} : n$
- a set Eq of equations between derived terms

Many of Moggi's monads (T,η,μ) are induced by the free algebra adjunction for suitable algebraic presentations, i.e.,

$T = U \circ F$ in



Example: **theory of boolean state**
- one binary operation $\mathsf{get} : 2$
- two unary operations $\mathsf{put}_0 : 1$ and $\mathsf{put}_1 : 1$
- subject to the following equations

$$x = \mathsf{get}(\mathsf{put}_0(x),\mathsf{put}_1(x)) \qquad \mathsf{put}_i(\mathsf{put}_j(x)) = \mathsf{put}_j(x)$$

$$\mathsf{put}_i(\mathsf{get}(x_1,x_2)) = \mathsf{put}_i(x_i)$$

x is a computation variable

- this theory induces the well-known state monad $T\,X = 2 \Rightarrow (2 \times X)$

Refinement Types

We take the **fine-grain call-by-value** metalanguage [Levy et al.] as a basis.

- types $\quad A ::= 1 \mid 0 \mid A_1 \times A_2 \mid A_1 + A_2 \mid A_1 \rightharpoonup A_2$
- values $\quad V ::= x \mid \langle\rangle \mid \langle V_1, V_2\rangle \mid \mathrm{pr}_i\, V \mid \mathrm{in}_i\, V \mid \lambda\, x\,.\, M \mid \ldots$
- producers $\quad M ::= \mathrm{return}\ V \mid M_1\ \mathrm{to}\ x\,.\,M_2 \mid \mathrm{op}(M_1, \ldots, M_n) \mid V_1 V_2$

We define **refinement types** separately for values and effects.

- For values, we follow the style of type refinements [Pfenning et al.].

$$\tau ::= 1 \mid 0 \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \overset{\psi}{\rightharpoonup} \tau_2 \mid \bot \mid \tau_1 \vee \tau_2 \mid \tau_1 \wedge \tau_2$$

- For effects, we build the refinements systematically from modalities.

$$\begin{aligned}\psi ::= {} & [] \quad \text{(pureness modality)} \\ \mid {} & \langle \mathrm{op}\rangle(\psi_1, \ldots, \psi_n) \quad \text{(operation modalities for each n-ary operation)} \\ \mid {} & \mathrm{copy}\ \psi_1\ \mathrm{to}\ X_1, \ldots, X_n\ \mathrm{in}\ \psi_2 \quad \text{(explicit "diagonals")} \\ \mid {} & \bot \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid X \mid \mu\, X\,.\,\psi\end{aligned}$$

Refinements for effects describe equivalence classes of computation trees whose nodes are precisely the algebraic operations (i.e., modalities).

The well-formedness of refinement types is defined relative to the underlying FGCBV types, i.e., the well-formedness judgment is $\Delta \vdash \tau : \mathrm{ref}(A)$.

The main tool for logical reasoning is provided by two **subtyping** relations, one for values $\vdash \tau_1 \sqsubseteq_A \tau_2$ and one for effects $\Delta \vdash \psi_1 \sqsubseteq \psi_2$.

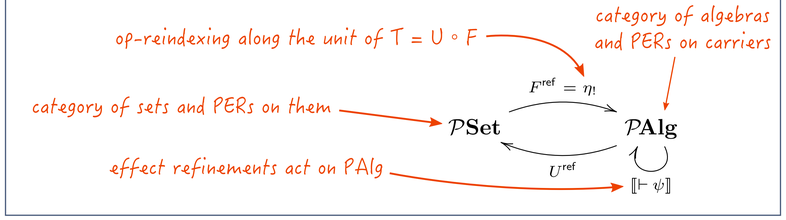
We need **explicit diagonals** to include all equations from the algebraic presentation (Σ,Eq) at hand, especially the non-linear ones, e.g., despite $\mathsf{get}(\mathsf{put}_0(x)\,,\,\mathsf{put}_1(x)) = x$ in Eq, $\langle\mathsf{get}\rangle(\langle\mathsf{put}_0\rangle(\psi)\,,\,\langle\mathsf{put}_1\rangle(\psi)) \sqsubseteq \psi$ is not valid.

The **typing judgments** for terms are $\Gamma \vdash V : \tau$ and $\Gamma \vdash M : \tau\ !\ \psi$. Below are example typing rules for trivial computations, sequential composition and effects.

$$\frac{\Gamma \vdash V : \tau}{\Gamma \vdash \mathtt{return}\, V : \tau\ !\ []} \qquad \frac{\Gamma \vdash M_1 : \tau_1\ !\ \psi_1 \quad \Gamma, x : \tau_1 \vdash M_2 : \tau_2\ !\ \psi_2}{\Gamma \vdash M_1\, \mathtt{to}\, x.\, M_2 : \tau_2\ !\ \psi_1[\psi_2]}$$

$$\frac{\Gamma \vdash M_1 : \tau\ !\ \psi_1 \qquad \Gamma \vdash M_n : \tau\ !\ \psi_n}{\Gamma \vdash \mathsf{op}(M_1, \ldots, M_n) : \tau\ !\ \langle \mathsf{op}\rangle(\psi_1, \ldots, \psi_n)}$$

The **denotational semantics** is based on the following adjunction.



Optimizations

We can use our type system to validate **effect-dependent optimizations**, in the style of effect-and-type systems [Kammar & Plotkin], [Benton et al.], where types are annotated with additional effect information $\Gamma \vdash M : \tau\ !\ \varepsilon$. The effect annotations ε are usually sets of effect labels, e.g., $\varepsilon = \{\mathsf{put}_1\}$.

We take $\varepsilon \subseteq \Sigma$ and define the corresponding effect refinement ψ_ε as set of computation trees built from ε.

$$\psi_\varepsilon = \mu\, X\,.\, [] \vee \bigvee_{\mathsf{op} \in \varepsilon} \langle \mathsf{op}\rangle(X, \ldots, X)$$

Local optimizations are verified by using the the equations induced by (Σ,Eq).

Global optimizations depend on the properties of (Σ,Eq), e.g., **discard**

$$\frac{\Gamma \vdash M_1 : \tau_1\ !\ \psi_{\varepsilon_1} \qquad \Gamma \vdash M_2 : \tau_2\ !\ \psi_{\varepsilon_2} \qquad t(x, \ldots, x) = x \text{ for } t \text{ built from } \varepsilon_1}{M_1\, \mathtt{to}\, x.\, M_2 = M_2}$$

Refinement types allow our optimizations to be extensional, e.g.,

$\mathsf{get}(\mathsf{put}_0(\mathrm{return}\ \langle\rangle),\mathsf{put}_1(\mathrm{return}\ \langle\rangle))$ satisfies $\psi_\varnothing$ (no observable effects)

Hoare Logic

We can also use our type system to write specifications in the style of Hoare Logic, similarly to the Hoare state monad [Borgström et al.].

The **Hoare refinement** $\{P\}\ []\ \{Q\}$ for the theory of boolean state is given by an effect refinement, defined by the analysis on $P \subseteq \{0,1\}$, e.g.,

$$\{\{1\}\}\ []\ \{Q\} = \langle\mathsf{get}\rangle(\mathsf{V}_{i \in \{0,1\}} \langle\mathsf{put}_i\rangle([])\ ,\ \mathsf{V}_{q \in Q} \langle\mathsf{put}_q\rangle([]))$$

$\{P\}\ []\ \{Q\}$ classify certain normal forms of the theory of boolean state.

Standard rules from Hoare Logic become admissible, e.g.,

$$\frac{\Gamma \vdash M_1 : \tau_1\ !\ \{P\}\ []\ \{Q\} \qquad \Gamma, x : \tau_1 \vdash M_2 : \tau_2\ !\ \{Q\}\ []\ \{R\}}{\Gamma \vdash M_1\, \mathtt{to}\, x.\, M_2 : \tau_2\ !\ \{P\}\ []\ \{R\}}$$

Sessions, Protocols, Files

We can also express specifications on **sessions** and **protocols**. A simple example protocol deals with the correct use of a file.

$$\psi_{\mathrm{file}} = \langle\mathsf{open}\rangle(\mu\, X\,.\, \langle\mathsf{close}\rangle([]) \vee \langle\mathsf{write}_i\rangle(X) \vee \langle\mathsf{read}\rangle(X, X))$$

Further examples cover specifications on input/output behavior.

Extensions and Further Work

- Investigating the affine operadic theory of effect refinements.
- Extending the refinement type sytem to account for operations with value parameters and binding of return values, e.g., $\mathsf{get}_{\mathsf{loc}}((x : \mathrm{val})\,.\,M)$.
- The above also leads to a system with value dependencies in refinements.
- Extending the sytem with local effects and effect handlers.