Thesis Proposal

Algebraic Approaches to Semantics

Ohad Kammar

February 24, 2010

Introduction

The goal of this document is to outline the problems I address in my research, record my progress in solving them until the present day, and outline the means I plan to approach them. In the past year, I have looked into three problems:

- Static effect analysis and its denotational semantics (chapter 1).
- A new logic for access control (chapter 2).
- Presheaf denotational semantics for dependency analysis (chapter 3).

In the following three chapters I outline these problems, present our current results, and outline future research directions. A rough break-up of the tasks I mention appear in figure 1.

Note that the table contains only the first two problems, as I have chosen not to pursue the third further.

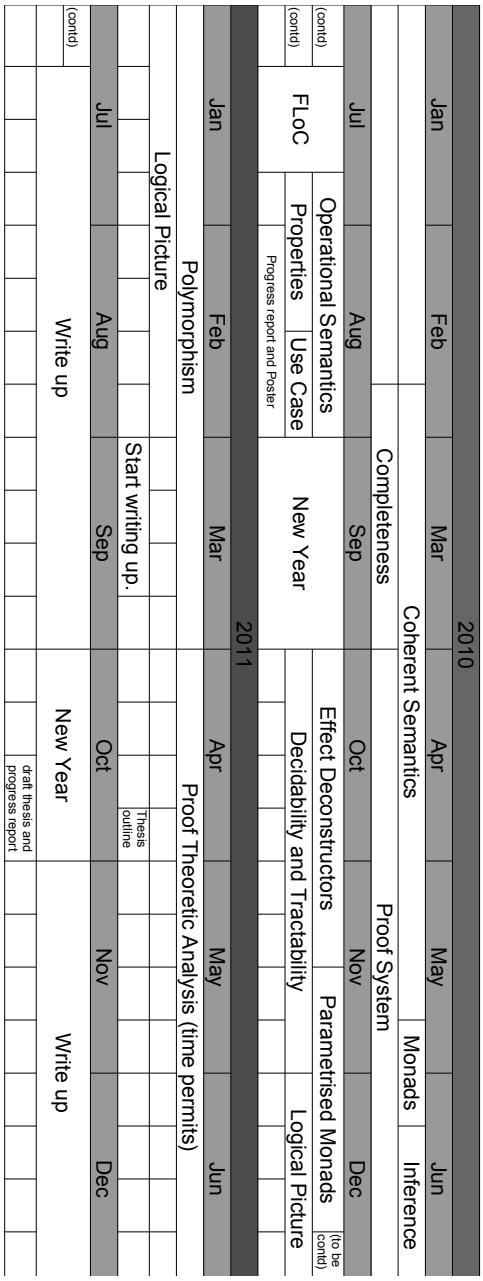

Figure 1: research schedule

Chapter 1

Algebraic Effect Type Systems

1.1 Introduction

Type and effect systems [TJ92a, Luc87] assign to each term in a programming language both a type and an effect set. The type describes the various values the term may evaluate to. The effect set describes the various computational effects (assignments, raising exceptions, IO, etc.) it may cause during its computation. For example, the term

$$M \coloneqq \textbf{if true then } \mathtt{set}(x, 1) \textbf{ else } \mathtt{set}(x, \mathtt{get}(y))$$

will have the unit type `unit` of a command, and the effect set $\{\mathtt{update}, \mathtt{lookup}\}$. These data are usually presented in the form of a type and effect judgement:

$$x\colon \mathbf{Loc}, y\colon \mathbf{Loc} \vdash M\colon \mathtt{unit} \,!\, \{\mathtt{update}, \mathtt{lookup}\}$$

Lacking a generic framework of computational effects, Talpin and Jouvelot designed their type and effect system with only global state as effects.

Wadler [Wad98, WT03] proposed a translation from the type and effect judgement $\Gamma \vdash M\colon A!\varepsilon$ to a computational judgement of the form $\Gamma' \vdash M'\colon T_\varepsilon A$. Again, Wadler did not provide a generic framework for generic effects, but used global state and remarked that "It seems clear that other effect systems can be [.. dealt with ..] in a similar way."

The translation was given both operational and denotational semantics. However, the only denotational semantics given were in terms of one monad, including all possible effects. Thus, the additional information given by the effect set is not fully used semantically.

Wadler conjectured the existence of a semantic model in which each syntactic monad T_ε denotes an actual monad, and the subset relation between effect sets factors through the monads: if $\varepsilon \subseteq \varepsilon'$ then there is a suitable monad

morphism $T_\varepsilon \to T_{\varepsilon'}$. He called such semantics *coherent*. To the best of our knowledge, some coherent semantics were given for a particular combination of effects [Tol98, Kie98], but no coherent semantics were given to an arbitrary combination of effects.

Plotkin, Power et. al. introduced the notion of *algebraic* computational effects [PP01b, PP03], and their semantics using Lawvere theories and equational theories [PP04]. These encompass state, non-determinism, exceptions, and IO [PP02]. Each Lawvere theory gives rise to a corresponding free monad, and given a cardinal κ, every κ-ranked monad is the free monad of some κ-Lawvere theory. However, there are monads that are not generated by any Lawvere theory, such as the continuation monad. Lawvere theories encode the way the different effects relate to each other, and allow various notions of compositions of effects [HPP06]. Thus, the algebraic theory of effects offer a modular view of computational effects.

We propose to use the algebraic theory of effects to give coherent denotational semantics to effect analysis. We plan to use some notion of a Lawvere subtheory to capture a subset of the algebraic effects included in a Lawvere theory. Lawvere theories seem to support a natural notion of subtheory: given a set ε of effects, we consider the smallest set of arrows containing the arrows corresponding to effects in ε, and closed in an appropriate way. We can also formulate the approach in equational terms: Let E be the equational theory corresponding to the full language, supporting all possible effects. Loosely speaking, E contains all true equations. Let $F \subseteq E$ be the set of all equations involving effects in ε. As F is an equational theory giving meaning to all effects in ε, it can be used to give semantics to terms whose effects are in ε. We want to investigate whether this method allows us to give coherent semantics to effect analysis.

As an initial step in this direction, we investigated the syntactic side of effect analysis. We provide three novel contributions.

- First, we have generalised the type and effect analysis to deal with arbitrary algebraic effects. The only comparable body of work is Marino's and Millstein's [MM09] generic effect type system, which has no ready connection to semantics.
- The second contribution is an effect type system to CBPV [Lev04], which appears to be completely novel.

  From the resulting effect type system, we can derive a CBV type-system, by following Levy's translation of CBV into CBPV. This CBV system, when instantiated to region analysis, coincides with the traditional CBV described by Wadler.
- Finally, we have also followed Levy's other translation and derived a CBN effect type system, which may be useful for CBN effectful languages such as Scala [OCD+06].

This chapter is organised as follows. Sections 1.2–1.4 provide background about type-and-effect systems, call-by-push-value and the algebraic theory of

effects. Section 1.5 describes the generic type-and-effect system. Section 1.6 derives type-and-effect systems for both CBV and CBN. Finally, section 1.7 presents our conclusions and outlines further work.

1.2 Region Tracking Effect System

Wadler [WT03] demonstrated his translation between traditional effect systems and the monadic effect systems using region analysis. His system included a polymorphic **let** construct and a fixed-point operator, both of which, for simplicity, we omit. Wadler also included memory allocation and references to arbitrary types. As such constructs still do not have satisfactory denotational semantics, we excluded them from this example system, and use only references to integers. The simplified system appears in figure 1.1.

A typical judgement in this type-and-effect system is

$$\Gamma \vdash_{\text{eff}} M : A \,!\, \varepsilon$$

Intuitively, it means that when evaluating the term M, only computational effects from ε will occur.

The expressions $\texttt{get}_\rho$ and $\texttt{set}_\rho$ access a store cell in a memory region ρ. The setting for region annotated programs is an intermediate language between region-free source code and low level machine code. The finite set of memory regions *Regions*, as well as the choice of region to use for each memory access is determined by a *region inference algorithm* [TB98, TBE+01].

Wadler defined three syntactic classes for his effect calculus: values, non-values and expressions. However, he made no explicit use of non-values, hence we follow Levy [Lev04], and only specify a syntactic subclass V of overall terms M.

The effect type system uses finite sets of effects ε, with $\texttt{update}_\rho$ and $\texttt{lookup}_\rho$ being the possible effects, for all regions ρ.

The primitive types are the unit type `unit`, integers **Int** and locations in region ρ, $\mathbf{Loc}_\rho$. For each finite effect set ε, the function type $A_1 \xrightarrow{\varepsilon} A_2$ corresponds to a function that, when applied, can only cause the effects in ε.

The values are either bound identifiers or lambda abstractions. Non-value terms include: post-fix function application M_1‘M_2, where the function M_2 is applied to the argument M_1; a monomorphic binding construction for terms with effects, $\texttt{x to } M_1.\ M_2$, which Wadler called **ilet**; and two store manipulation primitives, $\texttt{get}_\rho$ and $\texttt{set}_\rho$.

The two state primitives differ from Wadler's primitives in several technical ways. The $\texttt{get}_\rho$ operator has the same pragmatics of Wadler's **get**. However, the $\texttt{set}_\rho(M_1, M_2)$ term has the type `unit`, whereas the same term in Wadler's calculus would have the same type as M_2, and would evaluate to the same value as M_2. Also, Wadler's operations did not include the region explicitly. However, the region ρ is inferred explicitly while typing the location value $M_1 : \mathbf{Loc}_\rho \,!\, \varepsilon$, hence this difference is superficial. Finally, Wadler's system allows references to arbitrary types whereas our system only deals with references to integers.

$$\mathtt{x} \in \mathit{Id} \qquad \rho \in \mathit{Regions} \qquad \varepsilon \subseteq_{\text{fin}} \left\{\mathtt{lookup}_\rho, \mathtt{update}_\rho \middle| \rho \in \mathit{Regions}\right\}$$

$$V ::= \mathtt{x} \mid \lambda \mathtt{x}.M$$

$$M ::= V \mid E_1 \text{`} E_2 \mid \mathtt{let}\ M_1 = \mathtt{x}\ \mathtt{in}\ M_2 \mid \mathtt{set}_\rho(M) \mid \mathtt{get}_\rho(M_1, M_2)$$

$$\varepsilon ::= \emptyset \mid \left\{\mathtt{lookup}_\rho\right\} \mid \left\{\mathtt{update}_\rho\right\} \mid \varepsilon_1 \cup \varepsilon_2$$

$$A ::= \mathtt{unit} \mid \mathbf{Int} \mid \mathbf{Loc}_\rho \mid A_1 \xrightarrow{\varepsilon} A_2$$

$$\frac{}{\Gamma \vdash_{\text{eff}} \star \colon \mathtt{unit} \,!\, \emptyset} \qquad \frac{}{\Gamma \vdash_{\text{eff}} n \colon \mathbf{Int} \,!\, \emptyset}$$

$$\frac{}{\Gamma, \mathtt{x} \colon A \vdash_{\text{eff}} \mathtt{x} \colon A \,!\, \emptyset} \qquad \frac{\Gamma \vdash_{\text{eff}} M \colon A \,!\, \varepsilon_1 \quad \varepsilon_1 \subseteq \varepsilon_2}{\Gamma \vdash_{\text{eff}} M \colon A \,!\, \varepsilon_2}$$

$$\frac{\Gamma, \mathtt{x} \colon A_1 \vdash_{\text{eff}} M \colon A_2 \,!\, \varepsilon}{\Gamma \vdash_{\text{eff}} \lambda \mathtt{x}.M \colon A_1 \xrightarrow{\varepsilon} A_2 \,!\, \emptyset} \qquad \frac{\Gamma \vdash_{\text{eff}} M_1 \colon A_1 \,!\, \varepsilon_1 \quad \Gamma \vdash_{\text{eff}} M_2 \colon A_1 \xrightarrow{\varepsilon_2} A_2 \,!\, \varepsilon_3}{\Gamma \vdash_{\text{eff}} M_1 \text{`} M_2 \colon A_2 \,!\, \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3}$$

$$\frac{\Gamma \vdash_{\text{eff}} M_1 \colon A_1 \,!\, \varepsilon_2 \quad \Gamma, \mathtt{x} \colon A_1 \vdash_{\text{eff}} M_2 \colon A_2 \,!\, \varepsilon_2}{\Gamma \vdash_{\text{eff}} \mathtt{let}\ \mathtt{x} = M_1\ \mathtt{in}\ M_2 \colon A_2 \,!\, \varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Gamma \vdash_{\text{eff}} M \colon \mathbf{Loc}_\rho \,!\, \varepsilon}{\Gamma \vdash_{\text{eff}} \mathtt{get}_\rho(M) \colon \mathtt{unit} \,!\, \varepsilon \cup \left\{\mathtt{lookup}_\rho\right\}}$$

$$\frac{\Gamma \vdash_{\text{eff}} M_1 \colon \mathbf{Loc}_\rho \,!\, \varepsilon_1 \quad \Gamma \vdash_{\text{eff}} M_2 \colon \mathbf{Int} \,!\, \varepsilon_2}{\Gamma \vdash_{\text{eff}} \mathtt{set}_\rho(M_1, M_2) \colon \mathtt{unit} \,!\, \varepsilon_1 \cup \varepsilon_2 \cup \left\{\mathtt{update}_\rho\right\}}$$

Figure 1.1: a simplified region tracking system

Wadler, Tofte [Tof88], Talpin and Jouvelot [TJ92b] did not give denotational semantics for their calculi, and we are not familiar with any adequate denotational semantics for arbitrary references that do not involve solving infinite recursive domain equations. Therefore we restrict the system under consideration to integer references only, which indeed have adequate denotational semantics, and show how it generalises to arbitrary algebraic effects.

1.3 Call by Push Value

Call-by-push-value (CBPV) is an abstract programming language introduced by Levy [Lev99], that subsumes the two major flavours of lambda calculus, namely call-by-name (CBN) and call-by-value (CBV). This subsumption extends beyond a simple translation of CBN and CBV into CBPV — computational effect semantics, possible world semantics, game semantics and other semantic concepts, when defined for CBPV, subsume the known corresponding concepts for both CBN and CBV following the same translation mentioned above [Lev04]. Thus, we are motivated to define new concepts for CBPV rather than CBN or CBV, and derive the corresponding concepts for CBN and CBV by translation.

1.3.1 Syntax and Semantics

CBPV discerns syntactically between *values* A and *computations* $\underline{B}$, and has the following types:

$$A ::= U\underline{B} \mid \sum_{i\in I} A_i \mid \mathtt{unit} \mid A \times A \qquad \underline{B} ::= FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B}$$

The type FA is a computation that returns a value of type A. The type $U\underline{B}$ is a computation $\underline{B}$, thUnked into a value. CBPV has two notions of products, for values and for computations. The index set I appearing in the sum values and product computations is assumed to be finite. Note that functions are computations $A \to \underline{B}$, taking a value A to a computation $\underline{B}$.

The terms and type system of CBPV are given in figure 1.2. Big-step semantics for CBPV are given in figure 1.3. Levy gives additional semantics for CBPV: small-step operational semantics which use an auxiliary stack of contexts, hence the name call-by-*push*-value; categorical semantics which use the existence of a particular adjunction to give semantics to CBPV, and other semantics. Note that the adjunctions in Levy's categorical semantics arise out of any strong monad, hence generalise Moggi's computational lambda calculus and include the algebraic theory of effects.

1.3.2 Translating CBV and CBN into CBPV

In this section and the next one, we shall consider the simplified version of the simply typed lambda calculus, whose terms and types appear in figure 1.4. The CBV and CBN semantics are standard, hence omitted.

$$\frac{}{\Gamma \vdash^v_{\mathrm{p}} \star : \mathtt{unit}}$$

$$\frac{}{\Gamma, \mathtt{x} : A \vdash^v_{\mathrm{p}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^v_{\mathrm{p}} V : A \quad \Gamma, \mathtt{x} : A \vdash^c_{\mathrm{p}} M : \underline{B}}{\Gamma \vdash^c_{\mathrm{p}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^v_{\mathrm{p}} V : A}{\Gamma \vdash^c_{\mathrm{p}} \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash^c_{\mathrm{p}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^c_{\mathrm{p}} N : \underline{B}}{\Gamma \vdash^v_{\mathrm{p}} M\ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}}$$

$$\frac{\Gamma \vdash^c_{\mathrm{p}} M : \underline{B}}{\Gamma \vdash^v_{\mathrm{p}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^v_{\mathrm{p}} V : U\underline{B}}{\Gamma \vdash^c_{\mathrm{p}} \mathtt{force}\ V : \underline{B}}$$

$$\frac{\Gamma \vdash^v_{\mathrm{p}} V : A_{\hat{\imath}}}{\Gamma \vdash^v_{\mathrm{p}} (\hat{\imath}, V) : \sum_{i \in I} A_i}\ (\hat{\imath} \in I) \qquad \frac{\Gamma \vdash^v_{\mathrm{p}} V : \sum_{i \in I} A_i \quad \left\langle \Gamma, \mathtt{x} : A_i \vdash^c_{\mathrm{p}} M_i : \underline{B} \right\rangle_{i \in I}}{\Gamma \vdash^c_{\mathrm{p}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle (i, \mathtt{x}).M_i \rangle_{i \in I}\} : \underline{B}\}}$$

$$\frac{\Gamma \vdash^v_{\mathrm{p}} V_1 : A_1 \quad \Gamma \vdash^v_{\mathrm{p}} V_2 : A_2}{\Gamma \vdash^v_{\mathrm{p}} (V_1, V_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash^v_{\mathrm{p}} V : A_1 \times A_2 \quad \Gamma, x_1 : A_1, x_2 : A_2 \vdash^c_{\mathrm{p}} M : \underline{B}}{\Gamma \vdash^c_{\mathrm{p}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}_1, \mathtt{x}_2).M : \underline{B}}$$

$$\frac{\left\langle \Gamma \vdash^c_{\mathrm{p}} M_i : \underline{B}_i \right\rangle_{i \in I}}{\Gamma \vdash^c_{\mathrm{p}} \lambda\{\langle i.M_i \rangle_{i \in I}\} : \prod_{i \in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^c_{\mathrm{p}} M : \prod_{i \in I} \underline{B}_i}{\Gamma \vdash^c_{\mathrm{p}} \hat{\imath}`M : \underline{B}_{\hat{\imath}}}\ (\hat{\imath} \in I)$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^c_{\mathrm{p}} M : \underline{B}}{\Gamma \vdash^c_{\mathrm{p}} \lambda \mathtt{x}.M : (A \to \underline{B})} \qquad \frac{\Gamma \vdash^v_{\mathrm{p}} V : A \quad \Gamma \vdash^c_{\mathrm{p}} M : (A \to \underline{B})}{\Gamma \vdash^c_{\mathrm{p}} V`M : \underline{B}}$$

Figure 1.2: terms and type system of CBPV

$$\frac{}{\texttt{return}\ V \Downarrow \texttt{return}\ V} \qquad \frac{M[V/\texttt{x}] \Downarrow T}{\texttt{let}\ V\ \texttt{be}\ \texttt{x}.\ M \Downarrow T}$$

$$\frac{M \Downarrow \texttt{return}\ V \quad N[V/\texttt{x}] \Downarrow T}{M\ \texttt{to}\ \texttt{x}.\ N \Downarrow T} \qquad \frac{M \Downarrow T}{\texttt{force}\ \texttt{thunk}\ M \Downarrow T}$$

$$\frac{M_{\hat{\imath}}[V/\texttt{x}] \Downarrow T}{\texttt{pm}\ (\hat{\imath}, V)\ \texttt{as}\ \left\{\langle (i, \texttt{x}).M_i\rangle_{i\in I}\right\} \Downarrow T} \qquad \frac{M[V/x, V'/y] \Downarrow T}{\texttt{pm}\ (V, V')\ \texttt{as}\ (\texttt{x}, \texttt{y}).M \Downarrow T}$$

$$\frac{}{\lambda\{\langle i.M_i\rangle_{i\in I}\} \Downarrow \lambda\{\langle i.M_i\rangle_{i\in I}\}} \qquad \frac{M \Downarrow \lambda\{\langle i.N_i\rangle_{i\in I}\} \quad N_{\hat{\imath}} \Downarrow T}{\hat{\imath}`M \Downarrow T}$$

$$\frac{}{\lambda\texttt{x}.M \Downarrow \lambda\texttt{x}.M} \qquad \frac{M \Downarrow \lambda\texttt{x}.N \quad N[V/\texttt{x}] \Downarrow T}{V`M \Downarrow T}$$

Figure 1.3: big-step semantics for CBPV

$$
\begin{aligned}
&\mathtt{x} \in \mathit{Id} \\
&V ::= \mathtt{x} \mid \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid \lambda \mathtt{x}.M \\
&M ::= \mathtt{x} \mid \mathtt{inl}\ M \mid \mathtt{inr}\ M \mid \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N_1, \mathtt{inr}\ \mathtt{x}.N_2\} \\
&\qquad \mid \lambda x.M \mid E_1 \text{`} E_2 \mid \mathtt{let}\ M_1 = \mathtt{x}\ \mathtt{in}\ M_2 \\
&A ::= \mathtt{unit} \mid A_1 + A_2 \mid A_1 \to A_2
\end{aligned}
$$

$$
\frac{}{\Gamma, \mathtt{x}\colon A \vdash_\lambda \mathtt{x}\colon A}
$$

$$
\frac{\Gamma \vdash_\lambda M\colon A}{\Gamma \vdash_\lambda \mathtt{inl}\ M\colon A + B} \qquad \frac{\Gamma \vdash_\lambda M\colon B}{\Gamma \vdash_\lambda \mathtt{inr}\ M\colon A + B}
$$

$$
\frac{\Gamma \vdash_\lambda M\colon A_1 + A_2 \quad \Gamma, \mathtt{x}\colon A_1 \vdash_\lambda N_1\colon B \quad \Gamma, \mathtt{x}\colon A_2 \vdash_\lambda N_2\colon B}{\Gamma \vdash_\lambda \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N_1, \mathtt{inr}\ \mathtt{x}.N_2\}\colon B}
$$

$$
\frac{\Gamma, \mathtt{x}\colon A_1 \vdash_\lambda M\colon A_2}{\Gamma \vdash_\lambda \lambda \mathtt{x}.M\colon A_1 \to A_2} \qquad \frac{\Gamma \vdash_\lambda M_1\colon A_1 \quad \Gamma \vdash_\lambda M_2\colon A_1 \to A_2}{\Gamma \vdash_\lambda M_1 \text{`} M_2\colon A_2}
$$

$$
\frac{\Gamma \vdash_\lambda M_1\colon A_1 \quad \Gamma, \mathtt{x}\colon A_1 \vdash_\lambda M_2\colon A_2}{\Gamma \vdash_\lambda \mathtt{let}\ M_1 = \mathtt{x}\ \mathtt{in}\ M_2\colon A_2}
$$

Figure 1.4: a simply typed lambda calculus

$$\begin{aligned} \texttt{unit}^{\mathrm{v}} &\coloneqq \texttt{unit} \\ (A+B)^{\mathrm{v}} &\coloneqq A^{\mathrm{v}} + B^{\mathrm{v}} \\ (A \to B)^{\mathrm{v}} &\coloneqq U\,(A^{\mathrm{v}} \to FB^{\mathrm{v}}) \end{aligned} \qquad \begin{aligned} \texttt{x}^{\mathrm{val}} &\coloneqq \texttt{x} \\ \texttt{inl}\ V^{\mathrm{val}} &\coloneqq (1, V^{\mathrm{val}}) \\ \texttt{inr}\ V^{\mathrm{val}} &\coloneqq (2, V^{\mathrm{val}}) \\ (\lambda \texttt{x}.M)^{\mathrm{val}} &\coloneqq \texttt{thunk}\ \lambda \texttt{x}.M^{\mathrm{prod}} \end{aligned}$$

$$\begin{aligned} \texttt{x}^{\mathrm{prod}} &\coloneqq \texttt{return x} \\ (\texttt{inl}\ M)^{\mathrm{prod}} &\coloneqq M^{\mathrm{prod}}\ \texttt{to z.}\ \texttt{return}\ (1, \texttt{z}) \\ (\texttt{inr}\ M)^{\mathrm{prod}} &\coloneqq M^{\mathrm{prod}}\ \texttt{to z.}\ \texttt{return}\ (2, \texttt{z}) \\ (\texttt{pm}\ M\ \texttt{as inl x}.N_1, \texttt{inr x}.N_2)^{\mathrm{prod}} &\coloneqq M^{\mathrm{prod}}\ \texttt{to z.}\ \texttt{pm z as}\ \{(1,\texttt{x}).{N_1}^{\mathrm{prod}}, \\ &\qquad\qquad (2,\texttt{x}).{N_2}^{\mathrm{prod}}\} \\ (\lambda \texttt{x}.M)^{\mathrm{prod}} &\coloneqq \texttt{return thunk}\ \lambda \texttt{x}.M^{\mathrm{prod}} \\ (M\,`N)^{\mathrm{prod}} &\coloneqq M^{\mathrm{prod}}\ \texttt{to x.}\ N^{\mathrm{prod}}\ \texttt{to f.}\ \texttt{x`(force f)} \\ (\texttt{let x} = M\ \texttt{in}\ N)^{\mathrm{prod}} &\coloneqq M^{\mathrm{prod}}\ \texttt{to x.}\ N^{\mathrm{prod}} \end{aligned}$$

Figure 1.5: translation of CBV into CBPV

First, we describe the translation of CBV into CBPV.
Levy's translation consists of three functions:

- $-^{\mathrm{v}}$: Translating lambda calculus types into CBPV types.
- $-^{\mathrm{val}}$: Translating lambda calculus value terms V into CBPV values.
- $-^{\mathrm{prod}}$: Translating arbitrary lambda calculus terms into CBPV computations.

The details of the translation appear in figure 1.5. These translations exhibit the relation:

$$V^{\mathrm{prod}} \equiv \texttt{return}\ V^{\mathrm{val}}$$

If we extend the translation $-^{\mathrm{v}}$ to contexts $\Gamma = A_1, \ldots, A_n$ by

$$\Gamma^{\mathrm{v}} \coloneqq {A_1}^{\mathrm{v}}, \ldots, {A_n}^{\mathrm{v}}$$

then the following relation holds:

$$\Gamma \vdash_{\lambda} M \colon A \iff \Gamma^{\mathrm{v}} \vdash^{\mathrm{c}}_{\mathrm{p}} M^{\mathrm{prod}} \colon F(A^{\mathrm{v}}) \tag{1.1}$$

Now we turn to translating CBN into CBPV. Levy defines a translation $-^{\mathrm{n}}$ mapping CBN types and terms to CBPV computation types and terms, respectively. The translation is given in figure 1.6.

Again, if we extend the translation $-^{\mathrm{n}}$ to contexts $\Gamma = A_1, \ldots, A_n$ by

$$\Gamma^{\mathrm{n}} \coloneqq U{A_1}^{\mathrm{n}}, \ldots, U{A_n}^{\mathrm{n}}$$

$$
\begin{aligned}
\mathtt{unit}^{\mathrm{n}} &\coloneqq F\mathtt{unit} \\
(A+B)^{\mathrm{n}} &\coloneqq F(UA^{\mathrm{n}} + UB^{\mathrm{n}}) \\
(A \to B)^{\mathrm{n}} &\coloneqq (UA^{\mathrm{n}}) \to B^{\mathrm{n}}
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{x}^{\mathrm{n}} &\coloneqq \mathtt{force\ x} \\
(\mathtt{inl}\ M)^{\mathrm{n}} &\coloneqq \mathtt{return}\ (1, \mathtt{thunk}\ M^{\mathrm{n}}) \\
(\mathtt{inr}\ M)^{\mathrm{n}} &\coloneqq \mathtt{return}\ (2, \mathtt{thunk}\ M^{\mathrm{n}}) \\
(\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl\ x}.N_1, \mathtt{inr\ x}.N_2\})^{\mathrm{n}} &\coloneqq M^{\mathrm{n}}\ \mathtt{to\ z.\ pm\ z\ as}\ \{\mathtt{inl\ x}.{N_1}^{\mathrm{n}}, \\
&\qquad\qquad \mathtt{inr\ x}.{N_2}^{\mathrm{n}}\} \\
(\lambda \mathtt{x}.M)^{\mathrm{n}} &\coloneqq \lambda x.M^{\mathrm{n}} \\
(N`M)^{\mathrm{n}} &\coloneqq (\mathtt{thunk}\ N^{\mathrm{n}})`M^{\mathrm{n}} \\
(\mathtt{let\ x} = M\ \mathtt{in}\ N)^{\mathrm{n}} &\coloneqq \mathtt{let\ thunk}\ M^{\mathrm{n}}\ \mathtt{be\ x.}\ N^{\mathrm{n}}
\end{aligned}
$$

Figure 1.6: translation of CBN into CBPV

then the following relation holds:

$$\Gamma \vdash_{\lambda} M \colon A \iff \Gamma^{\mathrm{n}} \vdash^{\mathrm{c}}_{\mathrm{p}} M^{\mathrm{n}} \colon A^{\mathrm{n}} \tag{1.2}$$

1.4 Algebraic Theory of Effects

We say that a computational effect is *algebraic*, if its semantics can be captured by a collection of equations.

For example, consider *global state* as a computational effect. Assume that we have a collection of global *regions* in memory, indexed by ρ. Then we can define two operations for each ρ:

- $\mathtt{set}_\rho(V_1, V_2)$, which sets the location $V_1 \in \mathbf{Loc}_\rho$ in region ρ to the value $V_2 \in \mathbf{Int}$.
- $\mathtt{get}_\rho(V)$, which retrieves the value from location $V \in \mathbf{Loc}_\rho$ in region ρ.

Thus, after performing $\mathtt{set}_\rho(V_1, V_2)$, the command $\mathtt{get}_\rho(V_1)$ will evaluate to V_2. These generic operations [PP03] can equivalently be presented in a continuation-passing style fashion:

- Associated with $\mathtt{set}_\rho$ there is an operation $\mathtt{update}_\rho$ such that for each location V_1, value V_2, and term M, the term $\mathtt{update}_\rho^{V_1,V_2}(M)$ corresponds to writing V_2 in location V_1 and then continuing to execute M.
- Similarly, associated with $\mathtt{get}_\rho$ there is an operation $\mathtt{lookup}_\rho$, such that for every V_1 and term M, the term $\mathtt{update}_\rho^V(\mathtt{x}.M)$ corresponds to reading the current value from location V into $\mathtt{x}$, and then executing M.

$$\begin{aligned}
\mathtt{lookup}_\rho^{V_1}(\mathtt{x}.\mathtt{update}_\rho^{V_1,\mathtt{x}}(x)) &= x \\
\mathtt{lookup}_\rho^{V_1}(\mathtt{y}.\mathtt{lookup}_\rho^{V_1}(\mathtt{x}.M[\mathtt{x},\mathtt{y}])) &= \mathtt{lookup}_\rho^{V_1}(\mathtt{x}.M[\mathtt{x},\mathtt{x}]) \\
\mathtt{update}_\rho^{V_1,y}(\mathtt{update}_\rho^{V_1,x}(z)) &= \mathtt{update}_\rho^{V_1,x}(z) \\
\mathtt{update}_\rho^{V_1,x}(\mathtt{lookup}_\rho^{V_1}(\mathtt{y}.M[\mathtt{y}])) &= \mathtt{update}_\rho^{V_1,\mathtt{x}}(M[x]) \\
\mathtt{lookup}_\rho^{V_1}(\mathtt{x}.\mathtt{lookup}_\rho^{V_2}(\mathtt{y}.M[\mathtt{x},\mathtt{y}])) &= \mathtt{lookup}_\rho^{V_2}(\mathtt{y}.\mathtt{lookup}_\rho^{V_1}(\mathtt{x}.M[\mathtt{x},\mathtt{y}])), \quad V_1 \neq V_2 \\
\mathtt{update}_\rho^{V_1,x}(\mathtt{update}_\rho^{V_2,y}(z)) &= \mathtt{update}_\rho^{V_2,y}(\mathtt{update}_\rho^{V_1,x}(z)), \quad V_1 \neq V_2 \\
\mathtt{update}_\rho^{V_1,x}(\mathtt{lookup}_\rho^{V_2}(\mathtt{y}.M[\mathtt{y}])) &= \mathtt{lookup}_\rho^{V_2}(\mathtt{y}.\mathtt{update}_\rho^{V_1,x}(M[\mathtt{y}])), \quad V_1 \neq V_2
\end{aligned}$$

Figure 1.7: equations for global state

Using these operators, the semantics of global state can be captured in seven equations, given in figure 1.7.

This idea of semantics emerging from equational theories can be extended to give precise semantics to programming languages [PP01b]. Any set of equations gives rise to an equational theory, which in turn gives rise to a strong monad, called the *free monad* of the equational theory. In the cases of global state, nondeterminism, exception throwing and IO, the semantics given by equational theories and the usual monadic semantics coincide. In the previous example, when there is only one region, the seven equations above give rise to the familiar global state monad.

We can use this process to give a rigorous definition for an algebraic computational effect: a computational effect given by a monad is *algebraic* if its monad is the free monad of some equational theory. Other examples of algebraic computational effects are errors, input/output, non-determinism and probabilistic choice [PP02]. Not all monads are free monads of some equational theory, for instance the continuation monad.

The $\mathtt{lookup}_\rho^{V_1}(\mathtt{x}.M)$ operation is parametrised by the location V_1, and its argument is a computation M that depends on a value $\mathtt{x}$ of the stored type, say **Int**. Therefore the *signature* of $\mathtt{lookup}_\rho$ is $\mathbf{Loc}_\rho; \mathbf{Int}$.

More generally, we have a set of *signatures*, Σ_{eff}. Each signature is of the form $\mathrm{op}\colon A; A_1, \ldots, A_k$, where each of A and A_i is a *primitive type*. We assume that the primitive types are some objects in the intended semantics category, for example, the singleton $\{\star\}$, and the sets **Int** and $\mathbf{Loc}_\rho$. The meaning of these operations is given by the accompanying equational theory. In the case of global state, these are the equations of figure 1.7.

Thus, given a collection of primitive types $Primitives$, operation signatures Σ_{eff} involving these types, and equations involving these operations, we can consider a corresponding CBPV calculus involving them. Terms in this extended calculus may include values v from primitive types $P \in Primitives$:

$$\frac{}{\Gamma \vdash_{\text{p}}^{v} v\colon P} \tag{1.3}$$

and effect operations, whose semantics is given using their equations:

$$\frac{\Gamma \vdash^{v}_{\mathrm{p}} V \colon A \quad \left\langle \Gamma, \mathtt{x} \colon A_i \vdash^{c}_{\mathrm{p}} M_i \colon \underline{B} \right\rangle_{1 \le i \le n}}{\Gamma \vdash^{c}_{\mathrm{p}} \mathrm{op}_V \left\langle \mathtt{x}.M_i \right\rangle_{1 \le i \le n} : \underline{B}} \left(\mathrm{op} \colon A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}}\right) \tag{1.4}$$

The algebraic operations $\mathtt{update}_\rho$ and $\mathtt{lookup}_\rho$ can be used to define the commands $\mathtt{set}_\rho$ and $\mathtt{get}_\rho$, by

$$\mathtt{set}_\rho(V_1, V_2) \coloneqq \mathtt{update}_\rho^{V_1, V_2}(\mathtt{return}\ \star) \quad \mathtt{get}_\rho(V) \coloneqq \mathtt{lookup}_\rho^{V}(\mathtt{x}.\mathtt{return}\ \mathtt{x})$$

This connection exists in general. Given an algebraic effect of signature

$$\mathrm{op} \colon \alpha; \beta_1, \ldots, \beta_n$$

the *generic operation* [PP01b] associated with it is the term

$$\mathtt{gen}_{\mathrm{op}}(V) \coloneqq \mathrm{op}_V(\langle \mathtt{x}.\mathtt{return}\ (i, \mathtt{x}) \rangle_{1 \le i \le n}) \tag{1.5}$$

Using rule (1.4), we can derive the type of a generic operation

$$\frac{\Gamma \vdash^{v}_{\mathrm{p}} V \colon A}{\Gamma \vdash^{c}_{\mathrm{p}} \mathtt{gen}_{\mathrm{op}}(V) \colon F\,(A_1 + \ldots + A_n)} \left(\mathrm{op} \colon A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}}\right) \tag{1.6}$$

In existing programming languages, effects are given using generic operations and algebraic operations.

An equivalent treatment can be given using Lawvere theories [HP07]. Lawvere [Law63] proposed abstract (syntactic) categories as a means of treating universal equations. The operations are replaced by morphisms in a suitable category, and the equations are replaced by commutative diagrams. Thus, instead of talking about an equational theory, one talks about an abstract category. The commuting diagrams in this category correspond to the valid equations in the theory. A model for the equational theory corresponds to a functor from the abstract category into **Set** (or any other suitable model-class).

Lawvere theories can be composed in various manners [HPP06]. For instance, take L to be the Lawvere theory corresponding to the equations in figure 1.7 for all ρ, and let L_ρ be the corresponding theory corresponding to these equations for some fixed ρ.

As we have no equations relating effects related to region ρ with effects related to region ρ', in the semantics of L the order in which ρ and ρ' effects are applied matter. The theory L is called the *sum* of the theories L_ρ. In general, given Lawvere theories L_1 and L_2, their *sum theory* $L_1 + L_2$ corresponds to the equational theory without any equations between terms in L_1 and terms in L_2.

The preceding example motivates another kind of composition. Operations on different regions should be independent of each other. Therefore, a more accurate semantics for region tracking would include equations stating that operations on different regions *commute* with each other. For example:

$$\mathtt{update}_{\rho_1}^{V_1, V_2}(\mathtt{update}_{\rho_2}^{V_3, V_4}(M)) = \mathtt{update}_{\rho_2}^{V_3, V_4}(\mathtt{update}_{\rho_1}^{V_1, V_2}(M))$$

The resulting Lawvere theory is called the *tensor* of the Lawvere theories L_ρ. In general, given Lawvere theories L_1 and L_2, their *tensor theory*, $L_1 \otimes L_2$, corresponds to the theory in which operations from different theories commute with each other.

1.5 Effect Type System

We annotate CBPV computation types with finite sets of effects. These effect sets over-approximate the effects the computation will incur. The annotated types are given by

$$A ::= U\underline{B} \mid \sum_{i\in I} A_i \mid \mathtt{unit} \mid A \times A \qquad \underline{B} ::= (FA)\,!\,\varepsilon \mid (\prod_{i\in I} \underline{B}_i)\,!\,\varepsilon \mid (A \to \underline{B})\,!\,\varepsilon$$

We also define an auxiliary type $\hat{\underline{B}}$, such that $\underline{B} \equiv \hat{\underline{B}}\,!\,\varepsilon$:

$$\hat{\underline{B}} ::= FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B}$$

Note that function types have two effect sets associated with them. The first, outermost, set describes the effects required to produce the function. The second, inner, effect set describes the effects that take place when the function is invoked. Similar considerations apply to product computation types.

As in the usual CBPV, we have two kinds of typing judgements, for values and for computation types:

$$\Gamma \vdash^v_{\mathrm{f}} V : A \qquad\qquad \Gamma \vdash^c_{\mathrm{f}} M : \underline{B}$$

Let *Primitives* and Σ_{eff} be the primitive types and effect signature of an algebraic effect theory. The full effect type system for the corresponding CBPV calculus is given in figure 1.8.

Using this type system, we can derive a rule for typing generic effects:

$$\frac{\Gamma \vdash^v_{\mathrm{f}} V : A}{\Gamma \vdash^c_{\mathrm{f}} \mathtt{gen}_{\mathrm{op}}(V) : F\,(A_1 + \ldots + A_n)\,!\,\{\mathrm{op}\}}\ (\mathrm{op} : A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}}) \qquad (1.7)$$

Our type-and-effect system is sound in the following sense. One can inductively define the obvious strip $(\underline{B})$ function, which takes an effect annotated type in our type-and-effect system to a CBPV type. For instance,

$$\mathrm{strip}\,(\mathbf{Int} \to (F\,(\mathtt{unit})\,!\,\{\mathtt{update}\})\,!\,\{\mathtt{lookup}\}) = \mathbf{Int} \to F\,(\mathtt{unit})$$

Using this function, the following soundness result holds:

Theorem 1. *If* $\Gamma \vdash_{\mathrm{f}} M : A$ *then* $\mathrm{strip}\,(\Gamma) \vdash_{\mathrm{p}} M : \mathrm{strip}\,(A)$.

$$\frac{}{\Gamma \vdash^v_{\mathrm{f}} v \colon P}\,(v \in P) \qquad \frac{\Gamma \vdash^c_{\mathrm{f}} M \colon \hat{\underline{B}} \,!\, \varepsilon \quad \varepsilon \subseteq \varepsilon'}{\Gamma \vdash^c_{\mathrm{f}} M \colon \hat{\underline{B}} \,!\, \varepsilon'}$$

$$\frac{}{\Gamma, \mathtt{x} \colon A \vdash^v_{\mathrm{f}} \mathtt{x} \colon A} \qquad \frac{\Gamma \vdash^v_{\mathrm{f}} V \colon A \quad \Gamma, \mathtt{x} \colon A \vdash^c_{\mathrm{f}} M \colon \underline{B}}{\Gamma \vdash^c_{\mathrm{f}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M \colon \underline{B}}$$

$$\frac{\Gamma \vdash^v_{\mathrm{f}} V \colon A}{\Gamma \vdash^c_{\mathrm{f}} \mathtt{return}\ V \colon FA \,!\, \emptyset} \qquad \frac{\Gamma \vdash^c_{\mathrm{f}} M \colon FA \,!\, \varepsilon_M \quad \Gamma, \mathtt{x} \colon A \vdash^c_{\mathrm{f}} N \colon \hat{\underline{B}} \,!\, \varepsilon_N}{\Gamma \vdash^v_{\mathrm{f}} M\ \mathtt{to}\ \mathtt{x}.\ N \colon \hat{\underline{B}} \,!\, \varepsilon_M \cup \varepsilon_N}$$

$$\frac{\Gamma \vdash^c_{\mathrm{f}} M \colon \underline{B}}{\Gamma \vdash^v_{\mathrm{f}} \mathtt{thunk}\ M \colon U\underline{B}} \qquad \frac{\Gamma \vdash^v_{\mathrm{f}} V \colon U\underline{B}}{\Gamma \vdash^c_{\mathrm{f}} \mathtt{force}\ V \colon \underline{B}}$$

$$\frac{\Gamma \vdash^v_{\mathrm{f}} V \colon A_{\hat{\imath}}}{\Gamma \vdash^v_{\mathrm{f}} (\hat{\imath}, V) \colon \sum_{i \in I} A_i}\,(\hat{\imath} \in I) \qquad \frac{\Gamma \vdash^v_{\mathrm{f}} V \colon \sum_{i \in I} A_i \quad \left\langle \Gamma, \mathtt{x} \colon A_i \vdash^c_{\mathrm{f}} M_i \colon \hat{\underline{B}} \,!\, \varepsilon_i \right\rangle_{i \in I}}{\Gamma \vdash^c_{\mathrm{f}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle (i, \mathtt{x}).M_i \rangle_{i \in I}\} \colon \hat{\underline{B}} \,!\, \bigcup_{i \in I} \varepsilon_i\}}$$

$$\frac{\Gamma \vdash^v_{\mathrm{f}} V_1 \colon A_1 \quad \Gamma \vdash^v_{\mathrm{f}} V_2 \colon A_2}{\Gamma \vdash^v_{\mathrm{f}} (V_1, V_2) \colon A_1 \times A_2} \qquad \frac{\Gamma \vdash^v_{\mathrm{f}} V \colon A_1 \times A_2 \quad \Gamma, x_1 \colon A_1, x_2 \colon A_2 \vdash^c_{\mathrm{f}} M \colon \underline{B}}{\Gamma \vdash^c_{\mathrm{f}} \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x}_1, \mathtt{x}_2).M \colon \underline{B}}$$

$$\frac{\langle \Gamma \vdash^c_{\mathrm{f}} M_i \colon \underline{B}_i \rangle_{i \in I}}{\Gamma \vdash^c_{\mathrm{f}} \lambda\{\langle i.M_i \rangle_{i \in I}\} \colon (\prod_{i \in I} \underline{B}_i) \,!\, \emptyset} \qquad \frac{\Gamma \vdash^c_{\mathrm{f}} M \colon (\prod_{i \in I} (\hat{\underline{B}}_i \,!\, \varepsilon_i)) \,!\, \varepsilon}{\Gamma \vdash^c_{\mathrm{f}} \hat{\imath}`M \colon \hat{\underline{B}}_{\hat{\imath}} \,!\, \varepsilon \cup \varepsilon_{\hat{\imath}}}\,(\hat{\imath} \in I)$$

$$\frac{\Gamma, \mathtt{x} \colon A \vdash^c_{\mathrm{f}} M \colon \underline{B}}{\Gamma \vdash^c_{\mathrm{f}} \lambda \mathtt{x}.M \colon (A \to \underline{B}) \,!\, \emptyset} \qquad \frac{\Gamma \vdash^v_{\mathrm{f}} V \colon A \quad \Gamma \vdash^c_{\mathrm{f}} M \colon (A \to \hat{\underline{B}} \,!\, \varepsilon_1) \,!\, \varepsilon_2}{\Gamma \vdash^c_{\mathrm{f}} V`M \colon \hat{\underline{B}} \,!\, \varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Gamma \vdash^v_{\mathrm{f}} V \colon A \quad \left\langle \Gamma, \mathtt{x} \colon A_i \vdash^c_{\mathrm{f}} M_i \colon \hat{\underline{B}} \,!\, \varepsilon_i \right\rangle_{1 \le i \le n}}{\Gamma \vdash^c_{\mathrm{f}} \mathrm{op}_V\, \langle \mathtt{x}.M_i \rangle_{1 \le i \le n} : \underline{B} \,!\, \bigcup_{i=1}^{n} \varepsilon_i \cup \{\mathrm{op}\}}\,(\mathrm{op} \colon A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}})$$

Figure 1.8: generic effect type system

1.6 Subsumption

In this section we exploit the translations from CBV and CBN to CBPV, and derive type-and-effect systems for CBV and CBN.

First, we extend our simplified lambda calculus with effect operations. For any signature $\mathrm{op}\colon A; A_1, \ldots, A_n$, we add a rule:

$$\frac{\Gamma \vdash_\lambda M\colon A \quad \langle \Gamma, \mathtt{x}_i\colon A_i \vdash_\lambda M_i\colon B\rangle_{1\le i\le n}}{\Gamma \vdash_\lambda \mathrm{op}_M \langle \mathtt{x}_i.M_i\rangle_{1\le i\le n} : B}\ (\mathrm{op}\colon A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}}) \tag{1.8}$$

Using figure 1.4 and this last rule, we can derive a typing rule for generic effects:

$$\frac{\Gamma \vdash_\lambda M\colon A}{\Gamma \vdash_\lambda \mathtt{gen}_{\mathrm{op}}(M)\colon A_1 + \ldots + A_n}\ (\mathrm{op}\colon A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}}) \tag{1.9}$$

1.6.1 Subsuming CBV

We extend the translation in figure 1.5 to include effect operations:

$$\Big(\mathrm{op}_M(\langle \mathtt{x}.M_i\rangle_{1\le i\le n})\Big)^{\mathrm{prod}} \coloneqq M^{\mathrm{prod}}\ \mathtt{to}\ \mathtt{x}.\ \mathrm{op}_{\mathtt{x}}(\Big\langle \mathtt{x}.{M_i}^{\mathrm{prod}}\Big\rangle_{1\le i\le n})$$

and primitive values:

$$v^{\mathrm{val}} \coloneqq v$$

This extension preserves relation (1.1), in the following sense:

Theorem 2. *Let $\vdash_\lambda$ be a the type system for a simplified lambda calculus extended with effects of signature Σ_{eff}. Let $\vdash^c_{\mathrm{p}}, \vdash^v_{\mathrm{p}}$ be the CBPV type system corresponding to the effects of Σ_{eff}.*

Then the following relation holds:

$$\Gamma \vdash_\lambda M\colon A \iff \Gamma^{\mathrm{v}} \vdash^c_{\mathrm{p}} M^{\mathrm{prod}}\colon FA^{\mathrm{v}}$$

The proof is by induction on terms of the simplified lambda calculus, noting that each typing rule of $\vdash_\lambda$ corresponds to a valid derivation in $\vdash^c_{\mathrm{p}}, \vdash^v_{\mathrm{p}}$, that the end of each such CBPV derivation necessitates the premises of each $\vdash_\lambda$ derivation, and that the primitive types appearing in each signature are translated to themselves.

Now we are in the following position: For each lambda calculus term M, we have a corresponding CBPV term M^{prod}, reflecting its CBV meaning. Using our type-and-effect system, we can try to associate a type-and-effect judgement with M^{prod}. We define type-and-effect judgements for the lambda calculus which are subsumed by our type-and-effect system for CBPV. This type-and-effect system is defined in figure 1.9.

As function types are thunked computations, they always have an effect set thunked inside them. We expose this effect set in the type system by annotating each function type with an effect set. Type judgements take the form

$$\mathtt{x} \in \mathit{Id} \qquad \varepsilon \subseteq_{\text{fin}} \Sigma_{\text{eff}} \qquad P \in \mathit{Primitives} \qquad v \in P$$
$$V ::= v \mid \mathtt{x} \mid \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid \lambda \mathtt{x}.M$$
$$M ::= v \mid \mathtt{x} \mid \mathtt{inl}\ M \mid \mathtt{inr}\ M \mid \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N_1, \mathtt{inr}\ \mathtt{x}.N_2\}$$
$$\mid \lambda x.M \mid E_1 \text{`} E_2 \mid \mathtt{let}\ M_1 = \mathtt{x}\ \mathtt{in}\ M_2$$
$$A ::= P \mid A_1 + A_2 \mid A_1 \xrightarrow{\varepsilon} A_2$$

$$\frac{}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} v \colon P \,!\, \emptyset}$$

$$\frac{}{\Gamma, \mathtt{x} \colon A \vdash^{\text{cbv}}_{\lambda\varepsilon} \mathtt{x} \colon A \,!\, \emptyset} \qquad \frac{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M \colon A \,!\, \varepsilon_1 \quad \varepsilon_1 \subseteq \varepsilon_2}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M \colon A \,!\, \varepsilon_2}$$

$$\frac{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M \colon A \,!\, \varepsilon}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} \mathtt{inl}\ M \colon A + B \,!\, \varepsilon} \qquad \frac{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M \colon B \,!\, \varepsilon}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} \mathtt{inr}\ M \colon A + B \,!\, \varepsilon}$$

$$\frac{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M \colon A_1 + A_2 \,!\, \varepsilon \quad \Gamma, \mathtt{x} \colon A_1 \vdash^{\text{cbv}}_{\lambda\varepsilon} N_1 \colon B \,!\, \varepsilon_1 \quad \Gamma, \mathtt{x} \colon A_2 \vdash^{\text{cbv}}_{\lambda\varepsilon} N_2 \colon B \,!\, \varepsilon_2}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N_1, \mathtt{inr}\ \mathtt{x}.N_2\} \,!\, \varepsilon \cup \varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Gamma, \mathtt{x} \colon A_1 \vdash^{\text{cbv}}_{\lambda\varepsilon} M \colon A_2 \,!\, \varepsilon}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} \lambda \mathtt{x}.M \colon A_1 \xrightarrow{\varepsilon} A_2 \,!\, \emptyset} \qquad \frac{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M_1 \colon A_1 \,!\, \varepsilon_1 \quad \Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M_2 \colon A_1 \xrightarrow{\varepsilon_2} A_2 \,!\, \varepsilon_3}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M_1 \text{`} M_2 \colon A_2 \,!\, \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3}$$

$$\frac{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M_1 \colon A_1 \,!\, \varepsilon_2 \quad \Gamma, \mathtt{x} \colon A_1 \vdash^{\text{cbv}}_{\lambda\varepsilon} M_2 \colon A_2 \,!\, \varepsilon_2}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} \mathtt{let}\ M_1 = \mathtt{x}\ \mathtt{in}\ M_2 \colon A_2 \,!\, \varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} M \colon A \,!\, \varepsilon_0 \quad \left\langle \Gamma, \mathtt{x}_i \colon A_i \vdash^{\text{cbv}}_{\lambda\varepsilon} M_i \colon B \,!\, \varepsilon_i \right\rangle_{1 \le i \le n}}{\Gamma \vdash^{\text{cbv}}_{\lambda\varepsilon} \text{op}_M \left\langle \mathtt{x}_i.M_i \right\rangle_{1 \le i \le n} : B \,!\, \bigcup_{i=0}^{n} \varepsilon_i} \ (\text{op} \colon A; A_1, \ldots, A_n \in \Sigma_{\text{eff}})$$

Figure 1.9: a CBV type-and-effect system

$\Gamma \vdash^{\mathrm{cbv}}_{\lambda\varepsilon} M \colon A \,!\, \varepsilon$. Intuitively, this judgement means that evaluating M will only cause computational effects in ε.

In order to relate the CBV and CBPV type-and-effect systems, we present a translation from the former to the latter, using the original translation for terms, but changing the translation for types:

$$\begin{aligned} P^{\mathrm{v}'} &\coloneqq P \quad (P \in \mathit{Primitives}) \\ (A + B)^{\mathrm{v}'} &\coloneqq A^{\mathrm{v}'} + B^{\mathrm{v}'} \\ (A \xrightarrow{\varepsilon} B)^{\mathrm{v}'} &\coloneqq U(A^{\mathrm{v}'} \to F(B^{\mathrm{v}'} \,!\, \varepsilon)) \end{aligned} \tag{1.10}$$

The following theorem shows that this type-and-effect system captures the full expressive power of our CBPV system, for the lambda calculus:

Theorem 3. *For any CBV type-and-effect context Γ, lambda calculus term M, CBV type A and effect set ε, the following holds:*

$$\Gamma \vdash^{\mathrm{cbv}}_{\lambda\varepsilon} M \colon A \,!\, \varepsilon \iff \Gamma^{\mathrm{v}'} \vdash_{\mathrm{f}} M^{\mathrm{prod}} \colon FA^{\mathrm{v}'} \,!\, \varepsilon$$

From this system, we can also derive a rule for the generic operations:

$$\frac{\Gamma \vdash_{\lambda} M \colon A \,!\, \varepsilon}{\Gamma \vdash_{\lambda} \mathtt{gen}_{\mathrm{op}}(M) \colon A_1 + \ldots + A_n \,!\, \varepsilon \cup \{\mathrm{op}\}} \,(\mathrm{op} \colon A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}}) \tag{1.11}$$

By choosing our effects to be $\mathtt{lookup}_\rho$ and $\mathtt{update}_\rho$ for all $\rho \in \mathit{Regions}$, with suitable equations, we can instantiate the CBV type-and-effect system to obtain the simplified Wadler's type-and-effect system from figure 1.1.

1.6.2 Subsuming CBN

We now turn to CBN. We extend the translation in figure 1.6 to include effect operations:

$$\left(\mathrm{op}_M(\langle \mathtt{x}.M_i \rangle_{i \in I})\right)^{\mathrm{n}} \coloneqq M^{\mathrm{n}} \ \mathtt{to}\ \mathtt{x}.\ \mathrm{op}_{\mathtt{x}}(\langle \mathtt{x}.\mathtt{let\ thunk\ return\ x\ be\ x}.\ M_i{}^{\mathrm{n}} \rangle_{i \in I})$$

and value types:

$$v^{\mathrm{n}} \coloneqq \mathtt{return}\ v$$

This extension preserves relation (1.2), in the following sense:

Theorem 4. *Let $\vdash_\lambda$ be the type system for a simplified lambda calculus extended with effects of signature Σ_{eff}. Let $\vdash^c_{\mathrm{p}}, \vdash^v_{\mathrm{p}}$ be the CBPV type system corresponding to the effects of ${\Sigma_{\mathrm{eff}}}^{\mathrm{v}}$.*

Then the following relation holds:

$$\Gamma \vdash_\lambda M \colon A \iff \Gamma^{\mathrm{n}} \vdash^c_{\mathrm{p}} M^{\mathrm{n}} \colon A^{\mathrm{n}}$$

$$\mathtt{x} \in Id \qquad \varepsilon \subseteq_{\text{fin}} \Sigma_{\text{eff}} \qquad P \in Primitives \qquad v \in P$$

$$V ::= v \mid \mathtt{x} \mid \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid \lambda\mathtt{x}.M$$

$$M ::= v \mid \mathtt{x} \mid \mathtt{inl}\ M \mid \mathtt{inr}\ M \mid \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N_1, \mathtt{inr}\ \mathtt{x}.N_2\} \mid \lambda x.M \mid E_1\text{`}E_2 \mid \mathtt{let}\ M_1 = \mathtt{x}\ \mathtt{in}\ M_2$$

$$A ::= P\,!\,\varepsilon \mid (A_1 + A_2)\,!\,\varepsilon \mid (A_1 \to A_2)\,!\,\varepsilon$$

$$\hat{A} ::= P \mid A_1 + A_2 \mid A_1 \to A_2 \text{ (hence } A \equiv \hat{A}\,!\,\varepsilon)$$

$$\frac{}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} v \colon P\,!\,\emptyset}$$

$$\frac{}{\Gamma, \mathtt{x} \colon A \vdash^{\text{cbn}}_{\lambda\varepsilon} \mathtt{x} \colon A} \qquad \frac{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M \colon \hat{A}\,!\,\varepsilon_1 \quad \varepsilon_1 \subseteq \varepsilon_2}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M \colon \hat{A}\,!\,\varepsilon_2}$$

$$\frac{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M \colon A}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} \mathtt{inl}\ M \colon A + B\,!\,\emptyset} \qquad \frac{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M \colon B}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} \mathtt{inr}\ M \colon A + B\,!\,\emptyset}$$

$$\frac{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M \colon (A_1 + A_2)\,!\,\varepsilon_0 \quad \Gamma, \mathtt{x} \colon A_1 \vdash^{\text{cbn}}_{\lambda\varepsilon} N_1 \colon B\,!\,\varepsilon_1 \quad \Gamma, \mathtt{x} \colon A_2 \vdash^{\text{cbn}}_{\lambda\varepsilon} N_2 \colon B\,!\,\varepsilon_2}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N_1, \mathtt{inr}\ \mathtt{x}.N_2\} \colon B\,!\,\varepsilon_0 \cup \varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Gamma, \mathtt{x} \colon A_1 \vdash^{\text{cbn}}_{\lambda\varepsilon} M \colon A_2}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} \lambda\mathtt{x}.M \colon (A_1 \to A_2)\,!\,\emptyset} \qquad \frac{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M_1 \colon A_1 \quad \Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M_2 \colon (A_1 \to \hat{A}_2\,!\,\varepsilon_1)\,!\,\varepsilon_2}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M_1\text{`}M_2 \colon \hat{A}_2\,!\,\varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M_1 \colon A_1 \quad \Gamma, \mathtt{x} \colon A_1 \vdash^{\text{cbn}}_{\lambda\varepsilon} M_2 \colon A_2}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} \mathtt{let}\ M_1 = \mathtt{x}\ \mathtt{in}\ M_2 \colon A_2}$$

$$\frac{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} M \colon \hat{A}\,!\,\varepsilon_0 \quad \left\langle \Gamma, \mathtt{x} \colon \hat{A}_i\,!\,\emptyset \vdash^{\text{cbn}}_{\lambda\varepsilon} M_i \colon \hat{B}\,!\,\varepsilon_i \right\rangle_{1 \le i \le n}}{\Gamma \vdash^{\text{cbn}}_{\lambda\varepsilon} \text{op}_M\, \langle \mathtt{x}_i.M_i \rangle_{1 \le i \le n} : \hat{B}\,!\,\bigcup_{i=0}^{n} \varepsilon_i \cup \{\text{op}\}}\ (\text{op} \colon A; A_1, \ldots, A_n \in \Sigma_{\text{eff}})$$

Figure 1.10: a CBN type-and-effect system

The proof is similar to the proof of theorem 2. This time, we rely on the fact that all primitive types P are translated into FP.

Just as with CBV, we can now associate a CBPV type-and-effect judgement with each term. We present the system in figure 1.10.

As CBN types are translated into computations, they all have an effect set associated with them. Therefore, the CBN type-and-effect system associates an effect set with each type. Just as with the CBPV system, we define an auxiliary type $\hat{A}$, such that $A \equiv \hat{A} \,!\, \varepsilon$.

Another noteworthy property of this system involves contexts. As each type is of the form $\hat{A} \,!\, \varepsilon$, our typing judgements are of the form

$$\mathtt{x}_1 \colon \hat{A}_1 \,!\, \varepsilon_1, \ldots, \mathtt{x}_n \colon \hat{A}_n \,!\, \varepsilon_n \vdash^{\mathrm{cbn}}_{\lambda\varepsilon} M \colon \hat{A} \,!\, \varepsilon$$

Intuitively, such judgement means that given identifiers $\mathtt{x}_i$ that will cause effects from ε_i when evaluated, the term M will cause effects from ε when evaluated.

In order to relate the CBN and CBPV type-and-effect systems, we present a translation for CBN types:

$$\begin{aligned} (\hat{A} \,!\, \varepsilon)^{\mathrm{n}'} &\coloneqq \hat{A}^{\mathrm{n}'} \,!\, \varepsilon \\ P^{\mathrm{n}'} &\coloneqq FP \quad (P \in Primitives) \\ (A + B)^{\mathrm{n}'} &\coloneqq F(UA^{\mathrm{n}'} + UB^{\mathrm{n}'}) \\ (A \to B)^{\mathrm{n}'} &\coloneqq (UA^{\mathrm{n}'}) \to B^{\mathrm{n}'} \end{aligned} \tag{1.12}$$

The following theorem shows that this type-and-effect system captures the full expressive power of our CBPV system, for the lambda calculus:

Theorem 5. *For any CBN type-and-effect context Γ, lambda calculus term M, CBV type-and-effect A, the following holds:*

$$\Gamma \vdash^{\mathrm{cbn}}_{\lambda\varepsilon} M \colon A \iff \Gamma^{\mathrm{n}'} \vdash_{\mathrm{f}} M^{\mathrm{n}'} \colon FA^{\mathrm{n}'}$$

From this system, we can also derive a rule for the generic operations:

$$\frac{\Gamma \vdash_{\lambda} M \colon A \,!\, \varepsilon}{\Gamma \vdash_{\lambda} \mathtt{gen}_{\mathrm{op}}(M) \colon A_1 + \ldots + A_n \,!\, \varepsilon \cup \{\mathrm{op}\}} \,(\mathrm{op} \colon A; A_1, \ldots, A_n \in \Sigma_{\mathrm{eff}}) \tag{1.13}$$

The resulting type system was anticipated by Benton, Moggi and Hughes [BHM00][1]. In their words:

> The first thing to remark about the form of a type and effect judgement is that an effect appears on the right of the turnstile, but not on the left. This is because we are only considering CBV languages, and that means that at runtime free variables will always be bound to values, which have no effect. An effect system for an impure CBN language, were there any such thing, would have pairs of types and effects in the context too.

[1] Chapter "Type and Effect Systems", section 2, paragraph beginning with "The first thing to remark...".

The programming language Scala [OCD+06] is impure, and has CBN parameters. Up to this day, only one type-and-effect system has been suggested and implemented for Scala [RMO09]. This type-and-effect system only tracks one kind of effect: delimited continuations. As they are not covered by the algebraic theory of effects, we cannot compare the two systems. However, a general purpose type-and-effect system for Scala is an ongoing research topic, and we hope this work to contribute to this research.

1.7 Conclusions and Further Work

We have given a generic effect type system, subsuming the familiar CBV systems based on Talpin and Jouvelot. We have given a novel CBPV effect type system. We have instantiated our generic system to obtain a CBN type system. Therefore, the algebraic theory of effects enables a generic syntactic effect analysis.

The next step is to try to use the effect analysis to present coherent semantics, as described in the introduction. Doing so might prove complicated, or require changes in the proposed type system. Such changes might break subsumption of existing systems. The difference between the systems would be interesting to note, and will shed light on inadequacy of either system or semantics. Failure to form any kind of denotational semantics will limit the interest in our type system, but its generality as a syntactic analysis is still of interest, as it allows us to deal generally with the syntactic issues of effect analysis: effect inference, polymorphic effect type systems and proof theoretic results, as well as operational semantics. Even just the syntactic results can be compared with Marino's and Millstein's work on generic effect type systems [MM09], as it is purely syntactic.

With his translation, Wadler [Wad98] also presented a type inference algorithm for both the Talpin and Jouvelot and the monadic type systems. We would like such an algorithm in the general setting. If such an algorithm is not possible, we would like to know what limitations does the system suffer with respect to inference.

The over-approximation of the effects involving each term is reflected in our system by the subtyping rule:

$$\frac{\Gamma \vdash^c_{\mathrm{p}} \underline{B} \,!\, \varepsilon \quad \varepsilon \subseteq \varepsilon'}{\Gamma \vdash^c_{\mathrm{p}} \underline{B} \,!\, \varepsilon'}$$

However, polymorphism seems like a viable solution as well. Polymorphism is especially attractive if we consider high-order functions. For instance, a function that may ignore its parameter might be typed:

$$\forall \varepsilon \left((A \xrightarrow{\varepsilon} B) \xrightarrow{\emptyset} C \right)$$

While a function that uses both arguments might be typed:

$$\forall \varepsilon_1, \varepsilon_2 \left((A \xrightarrow{\varepsilon_1} B) \xrightarrow{\emptyset} (C \xrightarrow{\varepsilon_2} D) \xrightarrow{\varepsilon_1 \cup \varepsilon_2} E \right)$$

We would be interested in such generic type systems, how they relate to subtyping systems, and whether they can both be incorporated in the same system.

Wadler [Wad98] also gave operational semantics to both effect type systems, the Talpin and Jouvelot style and the monadic. It would be interesting to see how the existing operational semantics to algebraic effects [PP01a, Plo09], or variants thereof, relate to Wadler's operational semantics: bisimulation or even subsumption of Wadler's semantics by the existing semantics.

If coherent denotational semantics are found and verified to subsume existing systems and previous approaches, there are several avenues which we would like to explore.

The algebraic theory of effects also contains a notion of effect *deconstructors*, such as exception handlers [PP09]. We would like to add them to both effect analysis and the resulting semantics. To incorporate them we add effects to handler types. Handler types are given by:

$$\underline{D} ::= (A; \varepsilon) \Rightarrow \underline{B}$$

That is, handlers are parametrised by a value of type A, and when handling a computation whose effect set is ε, will perform a computation of type $\underline{B}$.

We have a prototype rule for effect handlers:

$$\frac{\left\langle \mathtt{x}_0 \colon A_0, \langle \mathtt{x}_v \colon A_v, \mathtt{x}_i \colon U((A_i \to \underline{B}) \,!\, \emptyset) \rangle_{1 \leq i \leq n} \vdash^c_{\mathrm{f}} M_{\mathrm{op}} \colon \hat{\underline{B}} \,!\, \varepsilon_{\mathrm{op}} \right\rangle_{\mathrm{op} \colon A; A_1, \ldots, A_n \in \varepsilon}}{\vdash^h_{\mathrm{f}} \mathtt{x}_0.\{\langle \mathrm{op}_{\mathtt{x}_v}(\mathtt{force}\ \mathtt{x}_c) = M_{\mathrm{op}} \rangle\} \colon (A_0; \varepsilon) \Rightarrow \hat{\underline{B}} \,!\, \bigcup_{\mathrm{op} \in \varepsilon} \varepsilon_{\mathrm{op}}}$$

Informally, if, for each effect in the handled set ε, the handler will perform a computation of type $\underline{B}$, then the handler has the type $(A; \varepsilon) \Rightarrow \underline{B}$. Recall that, since our system uses subtyping, each handler may have multiple types, corresponding to handling different effect sets.

Handler invocation is typed as follows:

$$\frac{\Gamma \vdash^c_{\mathrm{f}} M \colon FA_M \,!\, \varepsilon_M \quad \vdash^h_{\mathrm{f}} H \colon (A; \varepsilon_M) \Rightarrow \underline{B} \quad \Gamma \vdash^v_{\mathrm{f}} V \colon A \quad \Gamma, x \colon A_M \vdash^c_{\mathrm{f}} N \colon \underline{B}}{\Gamma \vdash^c_{\mathrm{f}} \mathtt{try}\ M\ \mathtt{with}\ H(V)\ \mathtt{as}\ \mathtt{x}.\ N \colon \underline{B}}$$

Thus, the handler removes all previous effects and causes only the effects in its type.

We still need to verify these rules subsume existing effect type systems (for example, Benton's and Kennedy's MIL-lite type system [BK99]). Then, semantics and inference need to be dealt with.

Another interesting research direction is to translate the semantic results from the language of Lawvere theories to monads. Once we know the semantic construction in terms of monads, we can try to generalise it from algebraic theories to arbitrary monads, or show that it cannot be generalised.

The notion of effects in the context of the typing judgement is reminiscent of Atkey's work on parameterised monads [Atk09]. Atkey gives an effect type

system for *permissions*, which has a single effect on each side of the judgement, which correspond to the sets of permissions required before and available after execution of the term in question. While related to our work, we have not spotted a direct relation between the two concepts yet.

In his work, Atkey abstracts from sets and unions and talks about a abstract operations. It might be possible to abstract the system from sets into a more general structure, perhaps a semilattice, or semigroup of some kind.

As our syntactic analysis already captures existing systems, a proof theoretic analysis for it will include accounts for the covered systems. Therefore, we are interested in pursuit of the Curry-Howard correspondence for our type system, and analysis of the resulting logic. In particular, it would be interesting to note whether it is constructive. If it is not always constructive, we would like to know what conditions are required to make it constructive, and for which effects these conditions are not met.

Chapter 2

Principal Logic for Access Control

2.1 Introduction

Current day computing involves many distributed systems. With the internet, cluster computing, distributed file systems and many other examples, centralised control is impossible. In such environments, different agents, or *principals*, have different capabilities: not everyone can access any web page, and only certain users can access a certain file. Deciding which principal can access which capability is the interest of access control.

The distributed nature of today's system forces us to break from the traditional centralised approach and consider distributed access control: we can no longer ask a centralised mechanism whether access should be granted, but localised access control monitors should reason about pieces of information obtained from several sources and base their decisions on them.

For instance, the monitor might have obtained the following pieces of information:

- A says all of B's friends can access her photographs.
- If B says C is his friend, then C is his friend.
- B says D is his friend.

The controller should deduce that "A says D can access her photographs", and as A controls all access to her photographs, D should be granted access.

Thus, there is a need for a logical framework modelling our intuitive understanding of access control so that systems and policies could be proved accurately formulated and proved correct. We propose a logic for access control that extends the expressiveness of existing logics.

DCC, the Dependency Core Calculus, is a calculus used by Abadi to analyse dependencies in programs [ABHR99]. By pursuing the Curry-Howard correspondence, Abadi tried to use the resulting logic to reason about access control [Aba06]. However, the logic corresponding to DCC was too powerful, and one could prove undesirable properties, such as A `says` B `says` $\varphi \iff B$ `says` A `says` φ [Aba07].

As a consequence, Abadi considered a subset of DCC, which he called Cut-Down DCC, CDD for short [Aba08]. CDD appears to better serve the requirements of an access control logic. However, its main disadvantage is its lack of relation between principals.

Principals usually come in hierarchies, where some principals are more trusted than others. If A is more trusted than B, then if A says something, we can deduce that B also says it. In its simplest form, CDD simply tags each statement with its source. Thus, there is no notion of trust between principals. One way to encode this notion is using second order propositional logic: everything that A says can be regarded as said by B.

Other flavors of CDD [GA08] allow for more degrees of expressiveness: incorporating propositions relating principals, and Boolean operations between principals that result in new principals.

We investigate whether CDD can be embedded, soundly and completely, in a more expressive access control logic, which we call principal logic. Principal logic treats principals as propositions, allowing us to define operations on principals using the existing logical connectives.

Our main contributions are:

- We follow the principals-as-propositions idea, and present a logic that soundly extends CDD, by its algebraic models and Kripke semantics.
- We show how compound operations on principals can be expressed in the logic.

This chapter is organised as follows. First, section 2.2 introduces CDD and its various flavours. Next, section 2.3 presents a short overview of Heyting algebras and quantales. Then, in section 2.4, we present our logic, principal logic: its syntax, its algebraic semantics, a canonical model, a translation of CDD into principal logic and its Kripke semantics. Finally, we conclude in section 2.5 along with suggestions for further work.

2.2 CDD

CDD formulae are given by the following syntax [Aba08]:

$$s ::= \texttt{true} \mid s \vee s \mid s \wedge s \mid s \supset s \mid A \texttt{ says } s \mid X \mid \forall X.s$$

where A ranges over elements of a set K of principals and X ranges over a set of propositional variables. The variable X is bound in $\forall X.s$.

$$[Var]\ \frac{}{\Gamma, s, \Gamma' \vdash s} \qquad [True]\ \frac{}{\Gamma \vdash \texttt{true}}$$

$$[Lam]\ \frac{\Gamma, s_1 \vdash s_2}{\Gamma \vdash s_1 \supset s_2} \qquad [App]\ \frac{\Gamma \vdash s_1 \supset s_2 \qquad \Gamma \supset s_1}{\Gamma \vdash s_2}$$

$$[Pair]\ \frac{\Gamma \vdash s_1 \qquad \Gamma \vdash s_2}{\Gamma \vdash s_1 \wedge s_2} \qquad [Proj]\ \frac{\Gamma \vdash s_1 \wedge s_2 \qquad i \in \{1,2\}}{\Gamma \vdash s_i}$$

$$[Inj]\ \frac{\Gamma \vdash s_i \qquad i \in \{1,2\}}{\Gamma \vdash s_1 \vee s_2} \qquad [Case]\ \frac{\Gamma \vdash s_1 \vee s_2 \quad \Gamma, s_1 \vdash s \quad \Gamma, s_2 \vdash s}{\Gamma \vdash s}$$

$$[Gen]\ \frac{\Gamma \vdash s}{\Gamma \vdash \forall X.s}\ (X \notin \mathrm{FV}(\Gamma)) \qquad [Spec]\ \frac{\Gamma \vdash \forall X.s}{\Gamma \vdash s[t/X]}$$

$$[Unit]\ \frac{\Gamma \vdash s}{\Gamma \vdash A\ \texttt{says}\ s} \qquad [Bind]\ \frac{\Gamma \vdash A\ \texttt{says}\ s \qquad \Gamma, s \vdash A\ \texttt{says}\ t}{\Gamma \vdash A\ \texttt{says}\ t}$$

Figure 2.1: proof system for CDD

The proof system for CDD is given in figure 2.1. Basic CDD does not deal with relationships between principals. It only deals with the interplay of deduction and the `says` relation.

One way to reason about the relationship between principals is using the speaks-for relation. We say that A speaks for B, and write $A \Rightarrow B$, if whenever A makes a claim, we can deduce that B makes the same claim. Using second-order propositional logic, we can encode the speaks-for relation as follows:

$$A \Rightarrow B \coloneqq \forall X.(A\ \texttt{says}\ X) \supset (B\ \texttt{says}\ X)$$

Abadi [GA08] investigated the use of a primitive construction for the speaks-for relation. He extends the syntax with the construction $s ::= \cdots \mid A \Rightarrow A$, and adds four axioms, given in figure 2.2.

Apart from relating principals to each other, policies may require the notion of compound principals — principals expressed using operations on other principals. For example, the principal A quoting B, written $A \mid B$, corresponds to a principal that only makes claims A claims B made:

$$(A \mid B)\ \texttt{says}\ s \iff A\ \texttt{says}\ (B\ \texttt{says}\ s)$$

One can also define conjunction, disjunction and other operations on principals.

Abadi [GA08] presents such an extension to CDD. Principals A are now given by the following syntax:

$$A ::= a \mid A \wedge A \mid A \vee A \mid A \supset A \mid \top \mid \bot$$

$$[\textit{Refl}] \quad \frac{}{\vdash A \Rightarrow A}$$

$$[\textit{Speaking-for}] \quad \frac{}{\vdash (A \Rightarrow B) \supset (A \texttt{ says } s) \supset (B \texttt{ says } s)}$$

$$[\textit{Trans}] \quad \frac{}{\vdash (A \Rightarrow B) \supset (B \Rightarrow C) \supset (A \Rightarrow C)}$$

$$[\textit{Handoff}] \quad \frac{}{(B \texttt{ says } (A \Rightarrow B)) \supset (A \Rightarrow B)}$$

Figure 2.2: additional axioms for CDD with a primitive speaks-for relation

$$[\textit{trust}] \quad \frac{}{\vdash (\bot \texttt{ says } s) \supset s}$$

$$[\textit{untrust}] \quad \frac{A \equiv \top}{\vdash A \texttt{ says } \bot}$$

$$[\textit{cuc}'] \quad \frac{}{\vdash ((A \supset B) \texttt{ says } s) \supset (A \texttt{ says } s) \supset (B \texttt{ says } s)}$$

Figure 2.3: additional axioms for CDD with Boolean principals

where a ranges over some set of atomic principals. Equality between principals, $A \equiv B$, is defined to be equality between them in classical logic. Thus the set of principals becomes a Boolean algebra. The inference rules of the basic CDD are extended with the rules in figure 2.3. The speaks-for relation can then be expressed as $(A \supset B)$ says $\bot$.

2.3 Heyting Algebras and Quantales

A *lattice* is a partially ordered set (poset), $(A, \leq)$, in which every two elements, $a, b \in A$ have a least upper bound, $a \vee b$, also called their join, and a greatest lower bound, $a \wedge b$, also called their meet. A lattice is *bounded* if it has a least element $\bot$ and a greatest element $\top$. The least and greatest elements are the units for the join and the meet operations, respectively.

A *Heyting algebra* is a bounded lattice $(H, \leq)$ which admits a *relative pseudo-complement* operator $\supset$, satisfying the following relation for all $a, b \in A$:

$$x \wedge a \leq b \iff a \supset b$$

In other words, a bounded lattice is a Heyting algebra if and only if, for all $a, b \in A$, there exists a greatest element, denoted by $a \supset b$, in the set

$$\{x | x \wedge a \leq b\}$$

In categorical terms, $a \supset b$ is the exponent b^a, hence a Heyting algebra is a Cartesian closed lattice.

A complete Heyting algebra is a partially ordered set, $(H, \leq)$, which has finite meets and arbitrary joins, which satisfies the following infinite distributivity law for all $x \in H$ and $S \subseteq H$:

$$x \wedge \bigvee S = \bigvee \{x \wedge a | a \in S\}$$

In all complete Heyting algebras, arbitrary meets are defined by

$$\bigwedge S \coloneqq \bigvee \{x | x \text{ is a lower bound of S}\}$$

The relative pseudo-complement of a and b is

$$a \supset b \coloneqq \bigvee \{x | x \wedge a \leq b\}$$

Complete Heyting algebras form a complete and sound model class for Intuitionistic Propositional Logic (IPL): Given a complete Heyting algebra, $(H, \leq)$, any assignment of elements of A to propositional variables gives rise to an interpretation of every IPL formulae as a value in A: propositional variables are interpreted according to the assignment; the logical connectives $\vee$, $\wedge$ and $\supset$ are interpreted using the Heyting algebra operations $\vee$, $\wedge$ and $\supset$, respectively; `true` and `false` are interpreted by $\top$ and $\bot$ respectively; and the quantifiers $\forall$ and $\exists$ are interpreted by arbitrary meets and joins, respectively. It turns out that an IPL formula is provable in IPL if and only if it takes the value $\top$ in any complete Heyting algebra $(A, \leq)$ under every assignment.

Quantales [Ros90, Gol06] are possibly noncommutative generalisations of complete Heyting algebras.

A *posemigroup* $\langle A, \leq, \otimes \rangle$ is a poset $(A, \leq)$ together with an associative binary operation $\otimes$ that is monotone in each argument:

$$x_1 \leq y_1, x_2 \leq y_2 \implies x_1 \otimes x_2 \leq y_1 \otimes y_2$$

Note that the operation $\otimes$ is not necessarily commutative. A posemigroup $\langle A, \leq, \otimes \rangle$ is *residuated* if there exist operations $\supset_l$, $\supset_r$ called *left and right residuals* of $\otimes$, satisfying the following equations:

$$x \otimes a \leq b \iff x \leq a \supset_l b$$
$$a \otimes x \leq b \iff x \leq a \supset_r b$$

In categorical terms, $a \supset_l -$ is the right adjoint to $- \otimes a$.

A *quantale* $\langle Q, \leq, \otimes \rangle$ is a complete poset with an associative operation $\otimes$, which satisfies the following distributivity laws for all $a \in Q$ and $S \subseteq Q$:

$$(\bigvee S) \otimes a = \bigvee \{x \otimes a | x \in S\}$$
$$a \otimes (\bigvee S) = \bigvee \{a \otimes x | x \in S\}$$

Each quantale has residuals, defined by

$$a \supset_l b \coloneqq \bigvee \{x | x \otimes a \leq b\}$$
$$a \supset_r b \coloneqq \bigvee \{x | a \otimes x \leq b\}$$

2.4 Principal Logic

We try to interpret principals as propositions in the calculus. Then, we try to give algebraic semantics to CDD using a suitable algebraic structure.

As CDD contains IL, we require the structure to have a Heyting algebra structure. We choose to interpret the `says` relation as a multiplicative left implication $\multimap$, right adjoint to the quoting relation $\otimes$. Thus the CDD formula

$$(A \text{ quotes } B) \texttt{ says } \varphi \iff A \texttt{ says } (B \texttt{ says } \varphi)$$

manifests as the element

$$(a \otimes b) \multimap \varphi \iff a \multimap (b \multimap \varphi)$$

of our structure, for suitable a, b and φ.

We have chosen to use quantales to model the access control operators.

However, a Heyting algebra and quantalic structure do not suffice. Instantiating the CDD unit axiom yields the following property:

$$a \leq b \multimap a \tag{unit}$$

or equivalently:

$$a \otimes b \leq a$$

Abadi [Aba08] showed that the C4 axiom holds in CDD:

$$(A \texttt{ says } A \texttt{ says } X) \supset A \texttt{ says } X$$

Therefore every model should satisfy the following inequality:

$$a \multimap (a \multimap b) \leq a \multimap b \tag{multiplication}$$

or, equivalently:

$$a \otimes b \leq a \otimes b \otimes b$$

We call these two inequalities the *monadic properties*. It turns out the monadic properties suffice for introducing a sound model class for CDD.

Therefore, let us define the syntax of principal logic:

Definition 6 (principal logic)**.** *Principal logic formulae are given by*

$$\varphi ::= \texttt{true} \mid \texttt{false} \mid \mathbf{P} \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \supset \varphi \mid \varphi \otimes \varphi \mid \varphi \multimap \varphi$$

where $\mathbf{P}$ *ranges over propositional variables.*

We give semantics to the logic via a model class of algebraic structures called *principal algebras*.

In this section, we will define principal algebras and state some of their properties. Then we will present a canonical model — a principal algebra constructed out of any partial order. We shall also present a sound translation of CDD into principal logic. Finally, we will present Kripke semantics for principal logic and state some of their properties.

2.4.1 Principal Algebras

Definition 7 (principal algebra)**.** *A* principal algebra *is a triple* $\langle P, \leq, \otimes \rangle$*, where* $\langle P, \leq \rangle$ *forms a complete Heyting algebra,* $\langle P, \leq, \otimes \rangle$ *forms a quantale, and the two monadic properties are satisfied:*

$$\textbf{unit: } a \otimes b \leq a \qquad \textbf{multiplication: } a \otimes b \leq a \otimes b \otimes b$$

Given a principal algebra $\langle P, \leq, \otimes \rangle$, we denote the meet, join and implication operations of the complete Heyting algebra by $\wedge$, $\vee$ and $\supset$, respectively, and the top and bottom elements by $\top$ and $\bot$, respectively. We denote the left implication by $\multimap$. That is, $x \leq a \multimap y$ if and only if $x \otimes a \leq y$.

Definition 8 (algebraic model)**.** *An* algebraic model *for principal logic is a principal algebra* $\langle P, \leq, \otimes \rangle$ *and an* interpretation *of each propositional variable* $\mathbf{P}$ *as an element* $[\![\mathbf{P}]\!]$ *of* P*.*

Each algebraic model gives rise to a natural interpretation of principal logic in the obvious way:

Definition 9 (interpretation in a model)**.** *Given an algebraic model* $\langle P, [\![-]\!] \rangle$*, the interpretation map can be extended inductively to formulae of principal logic by*

$$\begin{aligned}
[\![\texttt{true}]\!] &\coloneqq \top & [\![\texttt{false}]\!] &\coloneqq \bot \\
[\![\mathbf{P}]\!] &\coloneqq [\![\mathbf{P}]\!] & [\![\varphi_1 \vee \varphi_2]\!] &\coloneqq [\![\varphi_1]\!] \vee [\![\varphi_2]\!] \\
[\![\varphi_1 \wedge \varphi_2]\!] &\coloneqq [\![\varphi_1]\!] \wedge [\![\varphi_2]\!] & [\![\varphi_1 \supset \varphi_2]\!] &\coloneqq [\![\varphi_1]\!] \supset [\![\varphi_2]\!] \\
[\![\varphi_1 \otimes \varphi_2]\!] &\coloneqq [\![\varphi_1]\!] \otimes [\![\varphi_2]\!] & [\![\varphi_1 \multimap \varphi_2]\!] &\coloneqq [\![\varphi_1]\!] \multimap [\![\varphi_2]\!]
\end{aligned}$$

The element $[\![\varphi]\!]$ *is called the* interpretation *of* φ *in the model.*

We define the interpretation of any set of formulae Γ *as their conjunction:*

$$[\![\Gamma]\!] \coloneqq \bigwedge_{\varphi \in \Gamma} [\![\varphi]\!]$$

We intend principal logic not to have quantification. However, principal algebras can interpret quantifiers as well. Moreover, as CDD includes the universal quantifier, we include it to prove soundness. The procedure is standard: we add the notion of an assignment of values in a model to propositional variables, and

how it gives rise to an interpretation. Then, the interpretation of quantified formulae is:

$$[\![\forall X.\varphi]\!]_\sigma \coloneqq \bigwedge_{x\in P} [\![\varphi]\!]_{\sigma[X\mapsto x]} \qquad [\![\exists X.\varphi]\!]_\sigma \coloneqq \bigvee_{x\in P} [\![\varphi]\!]_{\sigma[X\mapsto x]}$$

The notions of a model and interpretation give rise to logical entailment:

Definition 10 (algebraic validity and entailment)**.** *Let Γ be a set of formulae, φ a formula and M an algebraic model.*

We say that φ is valid *in M, or that M* models *φ, and write $M \models_A \varphi$, if $[\![\varphi]\!] = \top$.*

We say that Γ entails *φ (algebraically), and write $\Gamma \models_A \varphi$, if, for every model M: if $M \models_A \psi$ for all $\psi \in \Gamma$, then $M \models_A \varphi$.*

By observing that every model M and element $a \in M$ give rise to a model $M_a \coloneqq \{x \leq a | x \in M\}$, we obtain an alternative criterion for entailment:

Theorem 11. *For every formulae set Γ and formula φ, $\Gamma \models_A \varphi$ if and only if $[\![\Gamma]\!] \leq [\![\varphi]\!]$ in every model.*

Using the previous theorem, we can prove the following theorem, which allow us to express operations on principals in our logic:

Theorem 12. *For all formulae φ_1, φ_2, ψ, ψ_1 and ψ_2, we have:*

1. $\models_A ((\varphi_1 \otimes \varphi_2) \multimap \psi) \backsimeq (\varphi_1 \multimap (\varphi_2 \multimap \psi))$.
2. $\models_A ((\varphi_1 \vee \varphi_2) \multimap \psi) \backsimeq ((\varphi_1 \multimap \psi) \wedge (\varphi_2 \multimap \psi))$.
3. $\models_A ((\varphi_1 \models_A \psi_1) \wedge (\varphi_2 \models_A \psi_2)) \supset ((\varphi_1 \wedge \varphi_2) \multimap (\psi_1 \wedge \psi_2))$.

where $\varphi \backsimeq \psi$ stands for $(\varphi \supset \psi) \wedge (\psi \supset \varphi)$.

Part 1 of the theorem, encourages us to interpret the quoting operation between principals using $\otimes$: the principal A `quotes` B will be represented as the formula $\mathbf{P}_A \otimes \mathbf{P}_B$.

Part 2 of the theorem lets us express the conjunction principal [ABLP93, GA08] $A \wedge B$ by the formula $\mathbf{P}_A \vee \mathbf{P}_B$.

Finally, part 3 of the theorem highlights a possible candidate for the disjunction principal $A \vee B$, which represents the group that only A and B are members of: the formula $\mathbf{P}_A \wedge \mathbf{P}_B$.

Thus we have found that our model class allows us to unify principal and propositional operations.

2.4.2 Canonical Model

We investigated a method to build a canonical model P out of any given principal hierarchy. By principal hierarchy, we assume given a partial order $\langle H, \leq \rangle$ of

principals. Following Abadi [Aba07], the order reflects trust in reverse correlation, thus when $a \leq b$ then a is more trusted than b. The model P is a principal algebra, and H can be embedded in P in reverse order.

The construction is done in two stages.

First, we consider the free partially ordered semigroup $\langle W, \leq, | \rangle$ over H, making $|$ monotone and satisfying the (reverse) monadic properties:

$$a \leq a \mid b \qquad\qquad a \mid b \mid b \leq a \mid b$$

Note that the two reverse monadic properties are equivalent to $- \mid b$ being a monad, for all b, as in W there is at most one morphism between any two objects.

Thus we obtain a posemigroup of principals W with a monotone operation $|$ that satisfies the monadic properties. This kind of structure may be the starting point of the construction in some situations, for instance in the investigation of our Kripke semantics (subsection 2.4.4). We shall construct a principal algebra out of any such W.

The Sierpinski space $\mathbb{O}$ consists of $\bot \leq \top$. Let P be the set of all order-preserving functions from W to $\mathbb{O}$. This set is no other than the functor category $\mathbb{O}^W$. Let $\mathbf{y} \colon W \to P$ be the covariant Yoneda embedding, $\mathbf{y} \colon x \mapsto \mathrm{Hom}\,(x, -)$. The mapping $\mathbf{y}$ embeds W within P — it is injective on the elements of W. The existence of such an embedding is guaranteed, as W is $\mathbb{O}$-enriched [ML98].

The partially ordered set P inherits a binary operation from W, using the Day convolution [Day71, Day70]:

$$(F \otimes G)(z) = \bigvee_{a \mid b \leq z} F(a) \wedge G(b)$$

It turns out that P is a principal algebra, allowing the definition of the operators $\supset$ and $\multimap$. The covariant Yoneda embedding from W to P reverses the order and maps the operation $|$ to $\otimes$.

We used the above construction to investigate the relationship between the algebraic model class and the Kripke semantics defined below (subsection 2.4.4). We also expect to use this construction when proving completeness of CDD in our model class.

2.4.3 CDD Translation

Principal algebras form a sound model class for CDD. We define a translation $(\!|-|\!)$ of CDD formulae to principal logic formulae with universal quantification over propositions. The translation amounts to translating A `says` into $(\!|A|\!) \multimap$:

$$\begin{aligned} (\!|A \texttt{ says } \varphi|\!) &\coloneqq (\!|A|\!) \multimap (\!|\varphi|\!) & (\!|\varphi \vee \psi|\!) &\coloneqq (\!|\varphi|\!) \vee (\!|\psi|\!) \\ (\!|\varphi \wedge \psi|\!) &\coloneqq (\!|\varphi|\!) \wedge (\!|\psi|\!) & (\!|\varphi \supset \psi|\!) &\coloneqq (\!|\varphi|\!) \supset (\!|\psi|\!) \\ (\!|\forall X.\varphi|\!) &\coloneqq \forall X.\, (\!|\varphi|\!) & (\!|\texttt{true}|\!) &\coloneqq \texttt{true} \end{aligned}$$

where the principal A is translated into a unique propositional variable $\mathbf{P}_A$.

Our translation is sound:

Theorem 13. *If $\Gamma \vdash \varphi$ is provable in CDD, then $\langle\!|\Gamma|\!\rangle \models_A \langle\!|\varphi|\!\rangle$.*

2.4.4 Kripke Semantics

Kripke semantics were introduced by Kripke [Kri63] to give a model theory to modal logic and intuitionistic logic. In this section we present Kripke semantics to principal logic and some of its properties, and prove soundness with respect to the algebraic semantics.

Definition 14 (Kripke models)**.** *A (principal) Kripke model is a quadruple $\langle W, \leq, |, (\!|-|\!)\rangle$ consisting of a monotone associative operator $|$ over a partially ordered set $\langle W, \leq\rangle$, satisfying the monadic properties:*

$$a \leq a \mid b \qquad\qquad a \mid b \mid b \leq a \mid b$$

and a map $(\!|-|\!)$ assigning an upward closed subset of W to every propositional variable:

$$a \leq b, a \in (\!|\mathbf{P}|\!) \implies b \in (\!|\mathbf{P}|\!)$$

We shall call the elements of K principals, or worlds.

The notion of validity in a Kripke model arises out of the notion of a world *forcing* a formula to be valid:

Definition 15 (forcing)**.** *Given a (principal) Kripke model $\langle W, \leq, |, (\!|-|\!)\rangle$, we define a forcing relation $w \Vdash \varphi$ between elements of W and propositions in principal logic as follows:*

- *$w \Vdash \mathbf{P}$ iff $w \in (\!|\mathbf{P}|\!)$.*
- *$w \Vdash \varphi_1 \otimes \varphi_2$ iff there exist w_1, w_2 in W, satisfying $w_1 \mid w_2 \leq w$, such that $w_i \Vdash \varphi_i$ for $i = 1, 2$.*
- *$w \Vdash \varphi \multimap \psi$ iff for all w' in W: if $w' \Vdash \varphi$ then $w \otimes w' \Vdash \psi$.*
- *The forcing relation is defined as usual for the IL portion:*
  - *$w \Vdash$ `true` for all $w \in W$.*
  - *$w \nVdash$ `false` for all $w \in W$.*
  - *$w \Vdash \varphi_1 \wedge \varphi_2$ iff $w \Vdash \varphi_i$, for both $i = 1, 2$.*
  - *$w \Vdash \varphi_1 \vee \varphi_2$ iff $w \Vdash \varphi_i$, for at least one $1 \leq i \leq 2$.*
  - *$w \Vdash \varphi_1 \supset \varphi_2$ iff for all $w' \geq w$: if $w' \Vdash \varphi_1$ then $w' \Vdash \varphi_2$.*

Thus, using the above definition, the following definition is standard:

Definition 16 (Kripke validity and entailment)**.** *Given a (principal) Kripke model W, we say that φ is* valid *in W, or that W* models *φ, and write $W \models_K \varphi$, if for all $w \in W$, $w \Vdash \varphi$.*

Given a set of formulae Γ and a formula φ, we say that Γ entails *φ in Kripke semantics, and write $\Gamma \models_K \varphi$, if for every Kripke model W, for which $W \models_K \psi$ for all $\psi \in \Gamma$, we also have $W \models_K \varphi$.*

As usual for intuitionistic Kripke semantics, the notion validity is monotone in the following sense:

Theorem 17 (monotonicity of Kripke semantics)**.** *Let K be a Kripke model. If $w \Vdash \varphi$ then for any $w' \geq w$, we also have $w' \Vdash \varphi$.*

Any Kripke model $\langle W, \leq, |, (\!|-|\!)\rangle$ allows encoding its principals $w \in K$ as propositions, by interpreting a propositional variable $\mathbf{P}_w$ by:

$$(\!|\mathbf{P}_w|\!) \coloneqq \{w' \in W | w' \geq w\}$$

Theorem 18. *If $(\!|\mathbf{P}_w|\!) = \{w' \in W | w' \geq w\}$, then $v \Vdash \mathbf{P}_w \multimap \varphi$ iff $v \mid w \Vdash \varphi$.*

The following lemma ties Kripke models with principal algebras:

Lemma 19. *For any Kripke model $\langle W, \leq, |, (\!|-|\!)\rangle$, the canonical principal algebra constructed out of W with the interpretation $[\![\mathbf{P}]\!] \coloneqq \mathbf{y}(\!|\mathbf{P}|\!)$, defined using the Yoneda embedding, satisfies:*

$$w \Vdash \varphi \iff [\![\varphi]\!](w) = \top$$

An immediate corollary of the previous lemma is the following completeness result:

Corollary 20 (Kripke completeness)**.** *Kripke models are sound: If $\Gamma \models_A \varphi$ using principal algebras, then $\Gamma \models_K \varphi$ using Kripke models. Consequently, Kripke semantics can be used to model CDD.*

We still need to investigate soundness of Kripke models in the class of principal algebras. Also, the Kripke semantics highlight the following possible incompatibility of principal logic and CDD:

In their modal deconstruction of CDD, Abadi and Garg [GA08] give Kripke semantics to various flavours of CDD. As they do not have an inherent quoting operation, the partial order in each model does not have a semigroup structure. Their semantics for formulae of the form A `says` φ follows the scheme

$w \Vdash A$ `says` φ iff for every $w' \geq w$, ...

Thus, the notion of A `says` φ depends only on worlds extending the current world. As our Kripke semantics depend on *all* possible worlds w', this incompatibility might be an indication for a difference between semantics.

As we have already shown soundness of CDD and completeness of Kripke models, this incompatibility can be one of two kinds. It may be caused by incompleteness of principal algebras: perhaps there are formulae that are true in principal logic but not provable in any of the flavours of CDD. It may also be caused by unsoundness of Kripke models: there might be formulae that are true in all Kripke models, but not true in all principal algebras.

2.5 Conclusion and Further work

We have presented a logic into which CDD can be soundly embedded. The logic treats principals and propositions equally, and admits some operations on principals using the existing connectives. The logic is given in terms of an algebraic model class and of Kripke semantics. The Kripke semantics are complete in the algebraic model class.

The first next step is to investigate the completeness of CDD in the model class of principal logic — whether any CDD formulae that is true in all principal algebras is provable in any of the flavours of CDD. As CDD is the current de-facto authority on access control, if it is indeed incomplete in our model class, we would better have good reasons to want to prove those formulae that are semantically true but unprovable in CDD. If we cannot find such good reasons, we should change our model class, or let go of the idea.

However, changing the model class will not be straightforward, as the two monadic properties follow directly from our translation of CDD. Thus, lack of completeness without sufficient justification will put an end to the principals-as-propositions idea.

If we do manage to prove completeness in some form, then we should devise a proof system, both sound and complete, in the model class of principal algebras. The fact that we have two kinds of arrows may introduce complications. A possible approach would be the same as adopted by bunched logic [OP99, Pym02], albeit noncommutative.

Once we expose the algebraic structure in a proof system, we would like to investigate how the other principal operators [ABLP91] can be expressed in principal logic, what theorems arise out of them, and establish the relation of principal logic to the various flavors of Abadi's CDD [GA08].

In particular, we would like to investigate whether the speaks-for relation can be expressed in principal logic without quantification. Abadi [GA08] has shown that second-order quantification in CDD is not required for reasoning about the speaks-for relation, in the first-order versions of the systems in figures 2.2–2.3. We hope to obtain similar results for principal logic.

To test the devised logic, we are interested in a case study in which one takes a realistic policy, encodes it in principal logic and prove various properties about it. Such a study will hopefully highlight unanticipated issues with the system.

Next, there are complexity and decidability issues that should be figured if principal logic is to become practical, such as proof checking and search. If the problem proves undecided or intractable, we should look for easily verifiable properties on the formulae the guarantee practical complexity.

Once complexity issues have been addressed, it would be interesting to incorporate principal logic within an existing framework (e.g., Alpaca [LLFS$^+$07], Binder [DeT02], DKAL [GN09]). Such an endeavour would put the logic to the test. However, we are not interested in pursuing this direction.

Every quantale has two kinds of arrows, the right adjoints to the functors $- \otimes b$ and $b \otimes -$. However, the principal algebras only make use of one of them.

We are interested in investigating whether the other kind of arrow is superfluous. If the other kind is not required, then we should obtain a smaller model class, and a better classification of the models. If both kinds are required, we should try to find out what information is conveyed by the other arrow.

A more theoretical research direction is to map the logical picture fully. This mapping means completing the investigation of the relationship of Kripke semantics to principal algebras and devising categorical semantics. Also, there should be a form of Stone duality here. Therefore, one should look for the Stone-dual of principal algebras and establish the duality.

Chapter 3

A Presheaf Model for DCC

3.1 Introduction

In [ABHR99], Abadi et. al. introduce the Dependency Core Calculus (DCC). The denotational semantics given there are ad-hoc to DCC, and uses indexed relations. Here we present supplementary denotational semantics for a subset of DCC, using presheaves. The presheaves pack away the indices, and seem less ad-hoc. However, it is not immediatly evident how these should generalise to incorporate recursion. In addition, the closed structure in our denotation does not coincide with the usual presheaf exponentiation, retaining some of the ad-hocness described earlier. We will not pursue this direction further.

The rest of this chapter is organised as follows. In section 3.2 we present the syntax and types of DCC. Next, in section 3.3 we present Abadi's relational semantics for DCC. Section 3.4 contains our presheaf model for DCC, some of its properties, its relation to the relational model and its disadvantages. We conclude and describe further work in section 3.5.

3.2 Dependency Core Calculus

In this chapter we will study a subset of DCC which excludes recursion. The goal is only to introduce the subset of DCC that we will deal with. For a full treatment of DCC, see [ABHR99].

Let $\mathcal{L} = (|\mathcal{L}|, \sqsubseteq)$ be a lattice of protection levels. These levels correspond to different views of the data.

For example, the protection level could describe publicity of the data. Thus the low protection levels reflect a wider availability of the data (e.g., in a website), while higher protection levels reflect more confidential data.

Computations between data introduce dependencies, which we wish to track. In the previous example, a public computation that checks whether a given number is stored on the server reveals information that should remain confidential.

DCC aims to negate this possibility, by ensuring that dependencies will not break the protection hierarchy.

DCC types are given by:

$$s ::= \texttt{int} \mid \texttt{bool} \mid \texttt{unit} \mid s \times s \mid s \Rightarrow s \mid T_l s$$

The monad $T_l s$ describes the type s, protected up to level l. Loosely speaking, $T_l s$ makes sure any level not greater than l will not be able to observe differences between values of this type.

DCC has standard terms, given by:

$$e ::= x \mid () \mid (e, e) \mid \pi_i\, e \mid \lambda x\colon s.e \mid e\, e \mid \eta_l\, e \mid \texttt{bind}\, x = e\, \texttt{in}\, e$$

In order to define typing rules for DCC, we inductively define the notion of "type protected at level l":

- $T_{l'} s$ is protected at level l for all $l \sqsubseteq l'$.
- If s_1, s_2 are protected at level l, then $s_1 \times s_2$ is protected at level l.
- If s is protected at level l then $T_{l'} s$ and $s' \Rightarrow s$ are protected at level l, for all levels $l' \in \mathcal{L}$ and types s'.

Using this notion, we type DCC:

$$\Gamma, x\colon s, \Gamma' \vdash x\colon s \qquad \Gamma \vdash ()\colon \texttt{unit}$$

$$\frac{\Gamma, x\colon s_1 \vdash e\colon s_2}{\Gamma \vdash (\lambda x\colon s_1.e)\colon (s_1 \Rightarrow s_2)} \qquad \frac{\Gamma \vdash e\colon (s_1 \Rightarrow s_2) \quad \Gamma \vdash e'\colon s_1}{\Gamma \vdash (e\, e')\colon s_2}$$

$$\frac{\Gamma \vdash e_1\colon s_1 \quad \Gamma \vdash e_2\colon s_2}{\Gamma \vdash (e_1, e_2)\colon s_1 \times s_2} \qquad \frac{\Gamma \vdash e\colon (s_1 \times s_2)}{\Gamma \vdash (\pi_i\, e)\colon s_i}$$

$$\frac{\Gamma \vdash e\colon s}{\Gamma \vdash (\eta_l\, e)\colon T_l s} \qquad \frac{\Gamma \vdash e\colon T_l s \quad \Gamma, x\colon s \vdash e'\colon s'}{\Gamma \vdash (\texttt{bind}\, x = e\, \texttt{in}\, e')\colon s'}\ s' \text{ protected at level } l$$

Note the unusual rule for bind.

3.3 Abadi's Relational Model

3.3.1 Semantics

To model DCC without recursion, we use sets. Let $\mathcal{R}$ be the following category. Its objects are pairs $A = (|A|, \langle R_l^A \rangle)$ whose first component is a set and second component is a family of relations indexed by $\mathcal{L}$. Its morphisms $f\colon A \to B$ are functions $f\colon |A| \to |B|$ respecting all the relations: if $a R_l^A \hat{a}$, then $f(a) R_l^B f(\hat{a})$, for all $a, \hat{a}$ in A and l in $\mathcal{L}$.

The terminal object is the singleton with diagonal relations, $\mathbf{1} \coloneqq (\{\star\}, \Delta)$. Products are the cartesian products, with relations defined component-wise, $A_1 \times A_2 \coloneqq (|A_1| \times |A_2|, \left\langle \left\{ ((a_1, a_2), (\hat{a}_1, \hat{a}_2)) \middle| (a_i, \hat{a}_i) \in R_l^{A_i} \right\} \right\rangle)$. Exponentiation is the usual collection of morphisms:

$$(A \Rightarrow B) \coloneqq (\mathcal{R}(A, B), \left\langle \left\{ (f, g) \middle| \forall a R_l^A \hat{a} : f(a) R_l^B g(\hat{a}) \right\} \right\rangle)$$

We define the protection monad for level l, T_l by:

$$|T_l A| \coloneqq |A|$$
$$R_{l'}^{T_l A} \coloneqq \begin{cases} R_{l'}^A & l \sqsubseteq l' \\ |A| \times |A| & \text{otherwise} \end{cases}$$

All these constructions extend to morphisms as well.

Finally, we specify two objects: the unprotected booleans, whose carrier set is $\{tt, ff\}$, and the unprotected integers, whose carrier set is $\mathbb{Z}$, both of which have the diagonal relations for all protection levels.

3.3.2 Properties

The relational model has the following properties:

Theorem 21. *For any type s, let $(A, \langle R_l \rangle)$ be its denotation in $\mathcal{R}$.*

1. *Equivalence: R_l is an equivalence relation.*
2. *Contravariance: If $l \sqsupseteq l'$ then $R_l \subseteq R_{l'}$.*
3. *Top identity: If the lattice has a top element $\top$, then $R_\top$ is the diagonal relation.*

Note that these properties are key to modeling DCC. Indeed, our intuition for the relations R_l is as equivalence relations, describing how the carrier set is viewed from some protection level. The contravariance condition corresponds to the decrease in the amount of information available in lesser protection levels. The top identity property corresponds to the fact that nothing is hidden from the most protected level.

Consequently, a more accurate relational model for DCC is the full subcategory of $\mathcal{R}$ consisting of all objects satisfying these properties. That is, all pairs of the form $A = (|A|, \langle R_l \rangle)$, where the relations are equivalence relations satisfying the contravariance and top identity conditions. We will keep our old notation $\mathcal{R}$ for this subcategory.

3.4 Presheaf Model

3.4.1 Semantics

We use presheaves to model DCC. This model is based on the observation that each protection level has its own view of the type, but these views are

suitably composed to respect the ordering on $\mathcal{L}$. Concretely, we consider the full subcategory $\mathcal{P}$ of $\mathbf{Set}^{\mathcal{L}^{\mathrm{op}}}$, induced by all the presheaves whose arrows have sections. That is, presheaves P such that for each $l \sqsupseteq l'$ there exist an arrow $f\colon P(l') \to P(l)$ satisfying $fP(l \sqsupseteq l') = \mathrm{id}$.

The last condition, which seems arbitrary, requires some explanation. Originally, since each level represents a different view of the data, the transitions between levels should be surjective. However, when setting out to prove the equivalence, the actual property used is the one above. In **Set**, these properties are equivalent, but this might not be the case when we generalise to other categories. However, even this property is insufficient, see section 3.4.4.

Denotations for unit and product are the presheaf terminal and presheaf product, accordingly. Exponentiation will be dealt with below. The protection monads are given by:

$$T_l \coloneqq x \mapsto \mathbf{Set}(\mathcal{L}(l, -), x(-))$$

Thus, given a presheaf P, $T_l P = \mathbf{Set}(\mathcal{L}(l, -), P(-))$ is the following functor. If $l \sqsubseteq l'$, then:

$$(T_l P)(l') = \mathbf{Set}(\mathcal{L}(l, l'), P(l')) = \mathbf{Set}(\{\star\}, P(l')) \cong P(l')$$

Otherwise,

$$(T_l P)(l') = \mathbf{Set}(\mathcal{L}(l, l'), P(l')) = \mathbf{Set}(\emptyset, P(l')) = \{\star\}$$

The T_l construction can be realised in the following way. Let $\mathcal{C}$ be a category. Consider the bifunctor $[-,-]\colon \left(\mathbf{Set}^{\mathcal{C}^{\mathrm{op}}}\right)^{\mathrm{op}} \times \mathbf{Set}^{\mathcal{C}} \to \mathbf{Set}^{\mathcal{C}}$, given by $[F, G]\colon x \mapsto \mathbf{Set}(Fx, Gx)$. By fixing the first component, we obtain an endofunctor $\mathbf{Set}^{\mathcal{C}} \to \mathbf{Set}^{\mathcal{C}}$, which can be shown to be a monad. By choosing $\mathcal{C} \coloneqq \mathcal{L}^{\mathrm{op}}$ and fixing the first component to be $\mathbf{y}(l)$, the image of l under the Yoneda embedding $x \mapsto \mathcal{C}(-, x)$, we obtain T_l.

Let us now turn to exponentiation. Unfortunately, $\mathcal{P}$ is not closed under the usual presheaf exponentiation. However, it does have a closed structure. We have found two ways to describe this structure.

The first way is through coreflection. Given any presheaf P, one can assign a presheaf KP by:

$$(KP)(A) \coloneqq \left\{ a \in P(A) \,\middle|\, \forall f\colon A \to B \forall g\colon \hat{A} \to B \exists \hat{a} \in P(\hat{A}) : P(f)(a) = P(g)(\hat{a}) \right\}$$

$$(KP)(A \xrightarrow{f} B) \coloneqq a \mapsto P(f)(a)$$

$$(K(P \xrightarrow{\alpha} Q))_A \coloneqq a \mapsto \alpha_A a$$

It can be shown that this is a right adjoint to the inclusion functor. Therefore, $\mathcal{P}$ is a coreflective subcategory of $\mathbf{Set}^{\mathcal{L}^{\mathrm{op}}}$. Thus, the Cartesian closed structure of

$\mathbf{Set}^{\mathcal{L}^{\mathrm{op}}}$ is inherited in $\mathcal{P}$ by coreflecting the normal constructions. The coreflection of the terminal object and the product are identical, while the coreflection of the exponent is not always identical.

The second construction is fairly straightforward. Given two presheaves P, Q in $\mathcal{P}$, the exponent Q^P is given by:

$$Q^P(l) \coloneqq \{a_l | a \colon P \to Q\}$$

$$Q^P(l \sqsupseteq l') \coloneqq a_l \mapsto a_{l'}$$

It can be shown that if all maps in P and Q have sections, then Q^P is a well-defined presheaf in $\mathcal{P}$ which forms an exponent with the obvious evaluation map.

3.4.2 An Example of Non-Interference

One of the reasons for using DCC is to formulate and prove non-interference properties [ABHR99].

Let $\mathcal{L} \coloneqq \{\mathtt{low}, \mathtt{high}\}$ with the obvious relation $\mathtt{low} \sqsubseteq \mathtt{high}$. Consider the high security booleans, $boolH(\mathtt{low}) \coloneqq \{\star\}$, $boolH(\mathtt{high}) \coloneqq \{tt, ff\}$ and the low security booleans, $boolL(\mathtt{low}) \coloneqq boolL(\mathtt{high}) \coloneqq \{tt, ff\}$.

Any natural transformation $f \colon boolH \to boolL$ must, by naturality, agree with the low security component $f_{\mathtt{low}} \colon \{\star\} \to \{tt, ff\}$. Consequently, all components of f must be constant.

3.4.3 Relation to the Relational Model

Under the additional assumption that the lattice $\mathcal{L}$ has a top element, it is possible to construct a fully faithful functor $F \colon \mathcal{R} \to \mathcal{P}$, which factors through units, products, exponents and the protection monads, by the mappings $F(A) \coloneqq l \mapsto A/R_l$ and $F(A)(l \sqsupseteq l') \coloneqq A/R_l \to A/R_{l'}$.

3.4.4 Disadvantages

The most obvious disadvantage of the presheaf model is that it is not obviously generalized to include recursion. Indeed, in many of the proofs we required that the presheaf mappings will have sections. It is not obvious what restriction should be placed on the relations R_l such that the quotients maps $A \to A/R_l$ will have continuous, monotone sections.

Another disadvantage is the unusual exponentiation. Had $\mathcal{P}$'s closed structure coincided with the usual presheaf exponent, a stronger case for using presheaves for semantics would have been established. The absence of this property limits the interest in the particular presheaf model.

3.5 Conclusions and Further Work

We considered a restricted form of Abadi's relational model, which has ad-hoc constructions and uses indices heavily. Also, the relational model leaves implicit the equivalence, contravariance and top identity properties.

We have managed to pack the indices into presheaves, and presented an equivalent model, when the protection lattice has a top element. We have also limited the ad-hoc constructions to the protection monads and exponentiation only.

The presheaf model, although equivalent to the restricted relational model, is not immediatly generalised to incorporate recursion.

Apart from trying to generalise the presheaf model to include recursion (perhaps using [Lev]), it would also be interesting to see whether the bifunctor $[-,-]$, introduced for constructing the protection monad T_l, crops up elsewhere.

Additional merits to the presheaf model should be considered. For example, whether Abadi's non-interference proofs become easier using the presheaf model, or whether research leaning upon the relational model encountered problems that can be avoided using the presheaf model.

In a different direction, one might be able to pack the indices, but keep the relations, by using presheaf equivalence relations. These are subobjects of the product $F \times F$, which are componentwise symmetric, reflexive and transitive.

Finally, even though our attempt to generically construct presheaf models with the usual presheaf operaions failed here, the attempt should not necessarily be abandoned. Perhaps the coreflective construction presented here is the correct approach to these presheaf semantics. Therefore, it is interesting to try to find other presheaf models that are Cartesian closed by being coreflective subcategories.

We do not plan to pursue this research avenue in the near future.

Bibliography

[Aba06] Martín Abadi, *Access control in a core calculus of dependency*, ICFP (John H. Reppy and Julia L. Lawall, eds.), ACM, 2006, pp. 263–273.

[Aba07] ———, *Access control in a core calculus of dependency*, Electronic Notes in Theoretical Computer Science **172** (2007), 5 – 31, Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin.

[Aba08] ———, *Variations in access control logic*, DEON (Ron van der Meyden and Leendert van der Torre, eds.), Lecture Notes in Computer Science, vol. 5076, Springer, 2008, pp. 96–109.

[ABHR99] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke, *A core calculus of dependency*, In Proc. 26th ACM Symp. on Principles of Programming Languages (POPL), ACM Press, 1999, pp. 147–160.

[ABLP91] Martín Abadi, Michael Burrows, Butler W. Lampson, and Gordon D. Plotkin, *A calculus for access control in distributed systems*, CRYPTO (Joan Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 576, Springer, 1991, pp. 1–23.

[ABLP93] ———, *A calculus for access control in distributed systems*, ACM Trans. Program. Lang. Syst. **15** (1993), no. 4, 706–734.

[Atk09] Robert Atkey, *Algebras for parameterised monads*, in Kurz et al. [KLT09], pp. 3–17.

[BHM00] Nick Benton, John Hughes, and Eugenio Moggi, *Monads and effects*, in International Summer School on Applied Semantics APPSEM2000, Springer-Verlag, 2000, pp. 42–122.

[BK99] Nick Benton and Andrew Kennedy, *Monads, effects and transformations*, Electronic Notes in Theoretical Computer Science, Elsevier, 1999, pp. 1–18.

[Day70] Brian J. Day, *Reports of the midwest category seminar iv*, Lecture Notes in Mathematics, vol. 137, ch. On closed categories of functors, pp. 1–38, Springer Berlin/Heidelberg, 1970.

[Day71] ———, *Construction of biclosed categories*, Bulletin of the Australian Mathematical Society **5** (1971), 139–140.

[DeT02] John DeTreville, *Binder, a logic-based security language*, Security and Privacy, IEEE Symposium on **0** (2002), 105.

[GA08] Deepak Garg and Martín Abadi, *A modal deconstruction of access control logics*, FoSSaCS (Roberto M. Amadio, ed.), Lecture Notes in Computer Science, vol. 4962, Springer, 2008, pp. 216–230.

[GN09] Yuri Gurevich and Itay Neeman, *DKAL 2 — a simplified and improved authorization language*, Tech report, Microsoft Research, February 2009.

[Gol06] Robert Goldblatt, *A Kripke-Joyal semantics for noncommutative logic in quantales*, Advances in Modal Logic (Guido Governatori, Ian M. Hodkinson, and Yde Venema, eds.), College Publications, 2006, pp. 209–225.

[HP07] Martin Hyland and John Power, *The category theoretic understanding of universal algebra: Lawvere theories and monads*, Electr. Notes Theor. Comput. Sci. **172** (2007), 437–458.

[HPP06] Martin Hyland, Gordon D. Plotkin, and John Power, *Combining effects: Sum and tensor*, Theor. Comput. Sci. **357** (2006), no. 1-3, 70–99.

[Kie98] Richard B. Kieburtz, *Taming effects with monadic typing*, ICFP, 1998, pp. 51–62.

[KLT09] Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki (eds.), *Algebra and coalgebra in computer science, third international conference, CALCO 2009, Udine, Italy, September 7-10, 2009. proceedings*, Lecture Notes in Computer Science, vol. 5728, Springer, 2009.

[Kri63] Saul A. Kripke, *Semantical considerations on modal logic*, Acta Philosophica Fennica **16** (1963), 83–94.

[Law63] Bill Lawvere, *Functorial semantics of algebraic theories*, Proceedings of the National Academy of Sciences of the United States of America **50** (1963), no. 1, 869–872.

[Lev] Paul Blain Levy, *How to quotient a cpo*, Can be found in: `http://www.cs.bham.ac.uk/~pbl/papers/quotientcpo.pdf`.

[Lev99] ———, *Call-by-push-value: A subsuming paradigm*, TLCA (Jean-Yves Girard, ed.), Lecture Notes in Computer Science, vol. 1581, Springer, 1999, pp. 228–242.

[Lev04] ———, *Call-by-push-value: A functional/imperative synthesis*, Semantics Structures in Computation, vol. 2, Springer, 2004.

[LLFS+07] Chris Lesniewski-Laas, Bryan Ford, Jacob Strauss, Robert Morris, and M. Frans Kaashoek, *Alpaca: extensible authorization for distributed services*, CCS '07: Proceedings of the 14th ACM conference on Computer and communications security (New York, NY, USA), ACM, 2007, pp. 432–444.

[Luc87] John Lucassen, *Types and effects, towards the integration of functional and imperative programming*, Ph.D. thesis, MIT Laboratory for Computer Science, August 1987.

[ML98] Saunders Mac Lane, *Categories for the working mathematician (graduate texts in mathematics)*, 2nd ed., Springer, September 1998.

[MM09] Daniel Marino and Todd Millstein, *A generic type-and-effect system*, TLDI '09: Proceedings of the 4th international workshop on Types in language design and implementation (New York, NY, USA), ACM, 2009, pp. 39–50.

[OCD+06] Martin Odersky, Vincent Cremet, Iulian Dragos, Gilles Dubochet, Burak Emir, Sean Mcdirmid, Stphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, Lex Spoon, and Matthias Zenger, *An overview of the Scala programming language (second edition)*, Tech. report, Emir B, McDirmid S, Micheloud S, Mihaylov N, Schinz M,. Stenman E, Spoon L, Zenger M, 2006.

[OP99] Peter W. O'Hearn and David J. Pym, *The logic of bunched implications*, Bulletin of Symbolic Logic **5** (1999), no. 2, 215–244.

[Plo09] Gordon D. Plotkin, *Adequacy for infinitary algebraic effects (abstract)*, in Kurz et al. [KLT09], pp. 1–2.

[PP01a] Gordon D. Plotkin and John Power, *Adequacy for algebraic effects*, FoSSaCS (Furio Honsell and Marino Miculan, eds.), Lecture Notes in Computer Science, vol. 2030, Springer, 2001, pp. 1–24.

[PP01b] ———, *Semantics for algebraic operations*, Electr. Notes Theor. Comput. Sci. **45** (2001), 332 – 345.

[PP02] ———, *Notions of computation determine monads*, Proc. FOSSACS 2002, Lecture Notes in Computer Science 2303, Springer, 2002, pp. 342–356.

[PP03] ———, *Algebraic operations and generic effects*, Applied Categorical Structures **11** (2003), 2003.

[PP04] ———, *Computational effects and operations: An overview*, Electr. Notes Theor. Comput. Sci. **73** (2004), 149–163.

[PP09] Gordon D. Plotkin and Matija Pretnar, *Handlers of algebraic effects*, ESOP (Giuseppe Castagna, ed.), Lecture Notes in Computer Science, vol. 5502, Springer, 2009, pp. 80–94.

[Pym02] D.J. Pym, *The semantics and proof theory of the logic of bunched implications*, Applied Logic Series, vol. 26, Kluwer Academic Publishers, 2002.

[RMO09] Tiark Rompf, Ingo Maier, and Martin Odersky, *Implementing first-class polymorphic delimited continuations by a type-directed selective CPS-transform*, ICFP (Graham Hutton and Andrew P. Tolmach, eds.), ACM, 2009, pp. 317–328.

[Ros90] Kimmo Rosenthal, *Quantales and their applications*, Pitman Research Notes in Mathematics Series, vol. 234, Longman Scientific & Technical, 1990.

[TB98] Mads Tofte and Lars Birkedal, *A region inference algorithm*, ACM Trans. Program. Lang. Syst. **20** (1998), no. 4, 724–767.

[TBE+01] Mads Tofte, Lars Birkedal, Martin Elsman, Niels Hallenberg, Tommy Hjfeld Olesen, Peter Sestoft, and Olesen Peter Sestoft, *Programming with regions in the ML Kit (for version 4)*, 2001.

[TJ92a] Jean-Pierre Talpin and Pierre Jouvelot, *Polymorphic type, region and effect inference*, Journal of Functional Programming **2** (1992), 245–271.

[TJ92b] ———, *The type and effect discipline*, LICS, IEEE Computer Society, 1992, pp. 162–173.

[Tof88] Mads Tofte, *Operational semantics and polymorphic type inference*, Ph.D. thesis, University of Edinburgh, 1988.

[Tol98] Andrew P. Tolmach, *Optimizing ML using a hierarchy of monadic types*, Types in Compilation (Xavier Leroy and Atsushi Ohori, eds.), Lecture Notes in Computer Science, vol. 1473, Springer, 1998, pp. 97–115.

[Wad98] Philip Wadler, *The marriage of effects and monads*, ICFP '98: Proceedings of the third ACM SIGPLAN international conference on Functional programming (New York, NY, USA), ACM, 1998, pp. 63–74.

[WT03] Philip Wadler and Peter Thiemann, *The marriage of effects and monads*, ACM Trans. Comput. Log. **4** (2003), no. 1, 1–32.