Monadic Regions

[Extended Abstract]

Matthew Fluet
Cornell University
Department of Computer Science
4133 Upson Hall
Ithaca, NY 14853
fluet@cs.cornell.edu

ABSTRACT

Drawing together two lines of research (that done in type-safe region-based memory management and that done in monadic encapsuation of effects), we give a type-preserving translation from a variation of the region calculus of Tofte and Talpin into an extension of System F augmented with monadic types and operations. Our source language is a novel region calculus, dubbed the Single Effect Calculus, in which sets of effects are specified by a single region representing an upper bound on the set. Our target language is $\mathsf{F}^{\mathsf{RGN}}$, which provides an encapsulation operator whose parametric type ensures that regions (and values allocated therein) are neither accessible nor visible outside the appropriate scope.

1. INTRODUCTION

Region-based memory management is a particular way to manage the dynamically (or *heap*) allocated memory of a program. It stands in contrast to explicit memory management by the programmer using operations like C's `malloc` and `free` and to fully automatic memory management using a garbage collector. In a region-based memory management system, regions are areas of memory holding heap allocated data. Regions have syntactic lifetimes, following the block structure of the program. A region is created upon entering a region delimited block; for the duration of the block, data can be allocated into the region; upon exiting the block, the entire region (including all data allocated within it) is destroyed. Tofte and Talpin's *region calculus* [24, 25] introduced a type system, based on effects, that ensures the safety of this allocation and deallocation mechanism. A unique feature of this scheme is that evaluation can lead to *dangling pointers*: a pointer to data that has been deallocated. So long as the program never dereferences such a pointer (a fact that the type system verifies), the program can be safely run. This aspect of region-based memory management systems can lead to better memory usage than that achieved by garbage collectors, which do not allow dangling pointers, on some programs.

While there has been some work aimed at integrating garbage collection and region-based memory management [9, 6], it is not possible to manage particular data by one scheme or the other. Cyclone [12] offers multiple forms of memory management in the context of a safe dialect of C. However, it makes use of a sophisticated type-and-effect system to ensure safety. We seek, therefore, a simple account that supports region-based memory management within a "traditional" functional programming language that primarily relies upon garbage collection and a simple type system.

A separate line of research has investigated mechanisms by which imperative (and otherwise "foreign") constructs can be safely integrated into pure functional languages. Launchbury and Peyton Jones [14, 15] introduced monadic state as a means by which imperative computations could be embedded in the pure evaluation of a functional program. The encapsulation operator runST has a type that statically guarantees that stateful computations appear as pure functions to the rest of the program. Inspired by this work, we propose monadic regions as a mechanism for embedding region-based memory management within a pure functional language.

The main contributions of this paper are three-fold. First, we introduce a novel region calculus, dubbed the *Single Effect Calculus*. The Single Effect Calculus tracks a partial order on regions; this order can be used allow a single region to bound the effect of a set of regions. Second, we introduce a novel monadic language, dubbed $\mathsf{F}^{\mathsf{RGN}}$, which is an extension of System F that adds monadic types and operations for manipulating regions. A key aspect of this language is that no extension (beyond adding three type constructors) to the type system of System F is required; in particular, type equality in $\mathsf{F}^{\mathsf{RGN}}$ is syntactic and all new language expressions can be interpreted as constants with polymorphic types. Encapsulation of region computations in $\mathsf{F}^{\mathsf{RGN}}$ is ensured by the type system, using parametricity. Finally, we give a type preserving translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$, an important first step towards demonstrating the adequacy of $\mathsf{F}^{\mathsf{RGN}}$ for expressing region-based memory management.

The remainder of this paper is structured as follows. In the next two sections, we describe the Single Effect Calculus and $\mathsf{F}^{\mathsf{RGN}}$. Section 4 presents a type preserving translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$. In Section 5, we consider related work. Section 6 concludes and notes

$$
\begin{array}{lcl}
i & \in & \mathbb{Z} \\
\varepsilon, \varrho & \in & \mathit{RVars}^{SEC} \qquad \text{where } \mathcal{H} \in \mathit{RVars}^{SEC} \\
f, x & \in & \mathit{Vars}^{SEC}
\end{array}
$$

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\text{Types}} \\
\tau & ::= & \mathsf{bool} \mid (\mu, \rho) \\
\multicolumn{3}{l}{\text{Boxed types}} \\
\mu & ::= & \mathsf{int} \mid \tau_1 \xrightarrow{\epsilon} \tau_2 \mid \tau_1 \times \tau_2 \mid \Pi\varrho \succeq \varphi.^{\epsilon}\tau \\
\multicolumn{3}{l}{\text{Effects}} \\
\varphi & ::= & \{\rho_1, \ldots, \rho_n\} \\
\multicolumn{3}{l}{\text{Places}} \\
\epsilon, \rho & ::= & \varrho
\end{array}
$$

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\text{Programs}} \\
p & ::= & e \\
\multicolumn{3}{l}{\text{Terms}} \\
e & ::= & i \ \mathsf{at}\ \rho \mid e_1 \oplus e_2 \ \mathsf{at}\ \rho \mid e_1 \otimes e_2 \mid \\
 & & \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f \mid \\
 & & x \mid \lambda x : \tau.^{\epsilon} e \ \mathsf{at}\ \rho \mid e_1\ e_2 \mid \\
 & & (e_1, e_2)\ \mathsf{at}\ \rho \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \mid \\
 & & \mathsf{new}\ \varrho.e \mid \lambda\varrho \succeq \varphi.^{\epsilon}u \ \mathsf{at}\ \rho \mid e\ [\rho] \mid \\
 & & \mathsf{fix}\ f : \tau.u \\
\multicolumn{3}{l}{\text{Abstractions}} \\
u & ::= & \lambda x : \tau.^{\epsilon} e \ \mathsf{at}\ \rho \mid \lambda\varrho \succeq \varphi.^{\epsilon}u \ \mathsf{at}\ \rho
\end{array}
$$

Figure 1: Single Effect Calculus: Syntax

some directions for future work. Supplemental figures can be found in the appendix.

2. THE SINGLE EFFECT CALCULUS

The Single Effect Calculus is a variation of the region calculus of Tofte and Talpin [24, 25], in the spirit of [11], and taking inspiration from the Capability Calculus [5] and Cyclone [8]. Essentially, the Single Effect Calculus capitalizes on the fact that a LIFO stack of regions imposes a partial order on live (allocated) regions. Regions lower on the stack outlive regions higher on the stack. Hence, the liveness of a region implies the liveness of all regions below it on the stack. Thus, it is the case that a single region can serve as a witness for a set of effects: the region appears as a *single effect* in place of the set.

Figure 1 presents the syntax of "initial programs" (that is, excluding intermediate terms that would appear in an operational semantics) of the Single Effect Calculus. Figures 2 and 3 presents a type system for this external language. Figure 10 in the Appendix presents a simple large-step operational semantics in terms of a run-time store. In the following sections, we explain and motivate the main constructs and typing rules of the Single Effect Calculus.

2.1 Types

In Tofte and Talpin's original region calculus, a region is associated with every type that requires heap allocated storage. We assume that integers, pairs, and closures require heap allocated storage, while booleans do not. The type (μ, ρ) pairs together a boxed type (a type requiring heap allocated storage) and a place (a region); we interpret (μ, ρ) as the type of objects of boxed type μ, allocated in region ρ. For our external language, it suffices to allows places to range over region variables (RVars^{SEC}), which include a distinguished member $\mathcal{H}$, corresponding to a global region that remains live throughout the execution of the program. Various operational semantics extend the syntactic class of places to include concrete region names [25] and/or a special constant corresponding to a deallocated region [4].

Of the boxed types, integers and pairs are standard. More interesting are the two boxed types corresponding to abstractions. Recall that in previous formulations of region calculi, the types of functions and region abstractions are given by:

$$\tau_1 \xrightarrow{\varphi} \tau_2 \quad \text{and} \quad \Pi\varrho.^{\varphi}\tau$$

where φ is an effect, a finite set of places. In the function and region-abstraction types, the effect φ denotes a *latent effect*, the set of regions read from or written to when an argument is supplied to the function or a region supplied to the region abstraction.

Our formulation of function and region abstraction types differ. The type of a function is given by:

$$\tau_1 \xrightarrow{\epsilon} \tau_2$$

where ϵ is a (single) place. Whereas φ denoted the set of regions affected by executing the function, ϵ denotes an upper bound (in the partial order of regions) on the set of regions affected by executing the function. In the case of the former, the typing judgement for an application requires showing that all regions in φ are live. In the case of the latter, the typing judgement for an application only requires showing that ϵ is live; the liveness of all regions below ϵ are implied by the stack discipline.

The type of a region abstraction is given by:

$$\Pi\varrho \succeq \varphi.^{\epsilon}\tau$$

where φ and ϵ are a finite set of places and a place, respectively. (Note that the region variable ϱ is bound within ϵ and τ, but not φ.) The bounded quantification in this type requires that at the instantiation of ϱ by a region ρ, we must be able to show that ρ is outlived by all of the regions in φ. Within the body of the abstraction, we assume that ϱ is an upper bound on the set of regions φ. As with a function type, ϵ denotes an upper bound on the set of regions affected when the region abstraction is applied to a region.

2.2 Programs and Terms

Programs in the Single Effect Calculus are simply terms. We distinguish programs as a syntactic class because the type system presented in the next section has a special judgement for top-level programs. Essentially, this judgement establishes reasonable "boundary conditions" for a program's execution, an aspect that is often overlooked in previous descriptions of region calculi.

Terms in the Single Effect Calculus are similar to those found in the λ-calculus. One major difference is that terms yielding heap allocated values carry a region annotation $\mathsf{at}\ \rho$, which indicates the region in which the value is to be allocated. New regions are introduced (and implicitly created and destroyed) by the $\mathsf{new}\ \varrho.e$ term. The region variable ϱ is bound within e, demarcating the scope of the region. Within e, values may be read from or allocated in the region ϱ. Executing $\mathsf{new}\ \varrho.e$ allocates a new region of memory, then executes e, and finally deallocates the region.

The term $\Pi\varrho \succeq \varphi.^{\epsilon}u$ at ρ introduces a region abstraction (allocated in the region ρ), where the term u is polymorphic in the region ϱ.[1] Such region polymorphism is particularly useful in the definition of functions, in which we parameterize over the regions necessary for the evaluation of the function. As explained in the previous section, region abstractions make use of bounded quantification; the intention is that ϱ is outlived by all the regions in φ. The term $e\ [\rho]$ eliminates a region abstraction; operationally, it substitutes the place ρ for the region variable ϱ in u and evaluates resulting term.

Finally, we include a fixed-point term, fix $f : \tau.u$. Since we intend the Single Effect Calculus to obey a call-by-value evaluation semantics, we limit the body of a fixed-point to abstractions.

As an example, consider the following term to compute a factorial (in which we elide the type annotation on *fact*):

```
fix $fact$.
    $(\Pi\varrho_i \succeq \{\}.^{\varrho_i}(\Pi\varrho_o \succeq \{\}.^{\varrho_i}(\Pi\varrho_b \succeq \{\varrho_i, \varrho_o\}.^{\varrho_i}(\lambda n : (\mathsf{int}, \varrho_i).^{\varrho_b}$
     if new $\varrho.n \leq (1$ at $\varrho)$
      then $1$ at $\varrho_o$
      else new $\varrho_{o'}.($new $\varrho_{i'}.$
           $(fact\ [\varrho_{i'}]\ [\varrho_{o'}]\ [\varrho_{i'}]$
                ($new $\varrho.n - (1$ at $\varrho)$ at $\varrho_{i'}))) * n$ at $\varrho_o$
    $)$ at $\varrho_i)$ at $\varrho_i)$ at $\varrho_i)$ at $\varrho_f$
```

The function *fact* is parameterized by three regions: ϱ_i is the region in which the input integer is allocated, ϱ_o is the region in which the output integer is to be allocated, and ϱ_b is a region that bounds the latent effect of the function. We see that the bounds on ϱ_i and ϱ_o indicate that they are not constrained to be outlived by any other regions. On the other hand, the bound on ϱ_b indicates that both ϱ_i and ϱ_o must outlive ϱ_b. Hence, ϱ_b suffices to bound the effects within the body of the function, in which we expect region ϱ_i to be read from and region ϱ_o to be allocated in. Note that the regions passed to the recursive call to *fact* satisfy the bounds, as $\varrho_{i'}$ clearly outlives itself and $\varrho_{o'}$ is allocated before (and deallocated after) $\varrho_{i'}$.

2.3 Typing Judgements

The typing rules for the Single Effect Calculus appear in Figures 2 and 3. Region contexts Δ are ordered lists of region variables bounded by effect sets. Value contexts Γ are ordered lists of variables and types. We summarize the main typing judgements in the following table:

Judgement	Meaning
$\Delta \vdash_{\mathrm{btype}} \mu$	Boxed type μ is well-formed.
$\Delta \vdash_{\mathrm{type}} \tau$	Type τ is well-formed.
$\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho$	If region ϵ is live, then region ρ is live. (Alt.: region ρ outlives region ϵ.)
$\Delta \vdash_{\mathrm{re}} \epsilon \succeq \varphi$	If region ϵ is live, then all regions in φ are live. (Alt.: all regions in φ outlive region ϵ.)
$\Delta; \Gamma \vdash_{\mathrm{exp}} e : \tau, \epsilon$	Term e has type τ and effects bounded by region ϵ.
$\vdash_{\mathrm{prog}} p$ ok	Program p is well-typed.

Previous formulations of region calculi make use of a judgement of the form $\Gamma \vdash_{\mathrm{exp}} e : \tau, \varphi$, where φ indicates the set of regions that may be effected by the evaluation of e (and the set of bound region variables is left implicit). The Single Effect Calculus simply replaces φ with a single region ϵ that bounds the effects in φ. In practice, and as suggested by the typing rules, ϵ corresponds to the most recently allocated region (also referred to as the top or current region).

[1]Limiting the body of a region abstraction to abstractions ensures that an erasure function that removes region annotations and produces a λ-calculus term is meaning preserving.

We start by noting that the typing rules for the judgements $\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho$ and $\Delta \vdash_{\mathrm{re}} \epsilon \succeq \varphi$ simply formalize the reflexive, transitive closure of the syntactic constraints in Δ, each of which asserts a particular "outlived by" relation between a region variable and an effect set.

The key judgement in region calculi is the typing rule for new $\varrho.e$:

$$\frac{\varrho \notin dom(\Delta) \qquad \Delta \vdash_{\mathrm{type}} \tau \qquad \Delta, \varrho \succeq \{\epsilon\}; \Gamma \vdash_{\mathrm{exp}} e : \tau, \varrho}{\Delta; \Gamma \vdash_{\mathrm{exp}} \mathsf{new}\ \varrho.e : \tau, \epsilon}$$

The antecedent $\Delta \vdash_{\mathrm{type}} \tau$ asserts that the new region variable ϱ does not appear in the result type, including any effects occurring in region abstraction types that appear in the result type. Note further that the antecedent $\varrho \notin dom(\Delta)$ and the implicit judgement $\vdash_{\mathrm{ctxt}} \Delta; \Gamma; \epsilon$ ensure that ϱ does not appear in the types of the value environment. Together, these facts guarantee that the region ϱ is not needed before the evaluation of e, nor is it needed after, corresponding to the allocation and deallocation of a new region. This new region is clearly related to the current region ϵ — it is outlived by the "old" current region and becomes the "new" current region for the evaluation of e. These facts are captured by the final antecedent $\Delta, \varrho \succeq \{\epsilon\}; \Gamma \vdash_{\mathrm{exp}} e : \tau, \varrho$.

It is worth comparing the treatment of latent effects in the Single Effect Calculus with their treatment in previous formulations of region calculi. In previous work, the typing rule for application appears as follows:

$$\frac{\Gamma \vdash_{\mathrm{exp}} e_1 : (\tau_1 \xrightarrow{\varphi} \tau_2, \rho), \varphi_1 \qquad \Gamma \vdash_{\mathrm{exp}} e_2 : \tau_2, \varphi_2}{\Gamma \vdash_{\mathrm{exp}} e_1\ e_2 : \tau_2, \varphi \cup \varphi_1 \cup \varphi_2 \cup \{\rho\}}$$

In the Single Effect Calculus, the composite effect $\varphi \cup \varphi_1 \cup \varphi_2 \cup \{\rho\}$ is witnessed by a single effect ϵ that subsumes the effect of the entire expression. We interpret ϵ as an upper bound on the composite effect; hence, ϵ is an upper bound on each of the effect sets φ_1 and φ_2, which explains why ϵ is used in the antecedents that type-check the subexpressions e_1 and e_2. In order to execute the application, the operational semantics must read the function out of the region ρ; therefore, we require ρ to outlive the current region ϵ by the antecedent $\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho$. Finally, we require the latent single effect ϵ', which is an upper bound on the set of regions affected by executing the function, to outlive the current region, which ensures that ϵ is also an upper bound on the set of regions affected by executing the function.

As alluded to in the previous section, the typing rule for region application requires that we be able to show that the formal region parameter ρ is outlived by all of the regions in the region abstraction bound φ.

Finally, the rule for top-level programs requires that an expression evaluate to a boolean value in the context of distinguished region $\mathcal{H}$ that remains live throughout the execution of the program. It also serves as the single effect that bounds the effects of the entire program. Alternative formulations of these "boundary conditions" exist; we have adopted these to simplify the translation in Section 4.

Region contexts
$\Delta \quad ::= \quad \cdot \mid \Delta, \varrho \succeq \varphi$

Value contexts
$\Gamma \quad ::= \quad \cdot \mid \Gamma, x : \tau$

$\boxed{\vdash_{\text{rctxt}} \Delta}$

$$\frac{}{\vdash_{\text{rctxt}} \cdot} \qquad \frac{\vdash_{\text{rctxt}} \Delta \qquad \varrho \notin \mathit{dom}(\Delta) \qquad \Delta \vdash_{\text{eff}} \varphi}{\vdash_{\text{rctxt}} \Delta, \varrho \succeq \varphi}$$

$\boxed{\Delta \vdash_{\text{place}} \rho}$

$$\frac{\varrho \in \mathit{dom}(\Delta)}{\Delta \vdash_{\text{place}} \varrho}$$

$\boxed{\Delta \vdash_{\text{eff}} \varphi}$

$$\frac{\Delta \vdash_{\text{place}} \rho_i \quad ^{i \in 1 \ldots n}}{\Delta \vdash_{\text{eff}} \{\rho_1, \ldots, \rho_n\}}$$

$\boxed{\Delta \vdash_{\text{btype}} \mu}$

$$\frac{}{\Delta \vdash_{\text{btype}} \mathsf{int}} \qquad \frac{\Delta \vdash_{\text{type}} \tau_1 \qquad \Delta \vdash_{\text{place}} \epsilon \qquad \Delta \vdash_{\text{type}} \tau_2}{\Delta \vdash_{\text{btype}} \tau_1 \xrightarrow{\epsilon} \tau_2} \qquad \frac{\Delta \vdash_{\text{type}} \tau_1 \qquad \Delta \vdash_{\text{type}} \tau_2}{\Delta \vdash_{\text{btype}} \tau_1 \times \tau_2}$$

$$\frac{\varrho \notin \mathit{dom}(\Delta) \qquad \Delta \vdash_{\text{eff}} \varphi \qquad \Delta, \varrho \succeq \varphi \vdash_{\text{place}} \epsilon \qquad \Delta, \varrho \succeq \varphi \vdash_{\text{type}} \tau}{\Delta \vdash_{\text{btype}} \Pi \varrho \succeq \varphi . ^{\epsilon} \tau}$$

$\boxed{\Delta \vdash_{\text{type}} \tau}$

$$\frac{}{\Delta \vdash_{\text{type}} \mathsf{bool}} \qquad \frac{\Delta \vdash_{\text{btype}} \mu \qquad \Delta \vdash_{\text{place}} \rho}{\Delta \vdash_{\text{type}} (\mu, \rho)}$$

$\boxed{\Delta \vdash_{\text{rr}} \epsilon \succeq \rho}$

$$\frac{\Delta(\varepsilon) = \{\rho_1, \ldots, \rho_i, \ldots, \rho_n\}}{\Delta \vdash_{\text{rr}} \varepsilon \succeq \rho_i} \qquad \frac{\Delta \vdash_{\text{place}} \epsilon}{\Delta \vdash_{\text{rr}} \epsilon \succeq \epsilon} \qquad \frac{\Delta \vdash_{\text{rr}} \epsilon \succeq \epsilon' \qquad \Delta \vdash_{\text{rr}} \epsilon' \succeq \rho}{\Delta \vdash_{\text{rr}} \epsilon \succeq \rho}$$

$\boxed{\Delta \vdash_{\text{re}} \epsilon \succeq \varphi}$

$$\frac{\Delta \vdash_{\text{rr}} \epsilon \succeq \rho_i \quad ^{i \in 1 \ldots n}}{\Delta \vdash_{\text{re}} \epsilon \succeq \{\rho_1, \ldots, \rho_n\}}$$

$\boxed{\Delta \vdash_{\text{vctxt}} \Gamma}$

$$\frac{}{\Delta \vdash_{\text{vctxt}} \cdot} \qquad \frac{\Delta \vdash_{\text{vctxt}} \Gamma \qquad x \notin \mathit{dom}(\Gamma) \qquad \Delta \vdash_{\text{type}} \tau}{\Delta \vdash_{\text{vctxt}} \Gamma, x : \tau}$$

$\boxed{\vdash_{\text{ctxt}} \Delta; \Gamma; \epsilon}$

$$\frac{\vdash_{\text{rctxt}} \Delta \qquad \Delta \vdash_{\text{vctxt}} \Gamma \qquad \Delta \vdash_{\text{place}} \epsilon}{\vdash_{\text{ctxt}} \Delta; \Gamma; \epsilon}$$

Figure 2: Single Effect Calculus: Static Semantics (I)

$\boxed{\Delta; \Gamma \vdash_{\text{exp}} e : \tau, \epsilon}$

$$\frac{\Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho}{\Delta; \Gamma \vdash_{\text{exp}} i \ \mathsf{at}\ \rho : (\mathsf{int}, \rho), \epsilon}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : (\mathsf{int}, \rho_1), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho_1 \\ \Delta; \Gamma \vdash_{\text{exp}} e_2 : (\mathsf{int}, \rho_2), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho_2 \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e_1 \oplus e_2 \ \mathsf{at}\ \rho : (\mathsf{int}, \rho), \epsilon}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : (\mathsf{int}, \rho_1), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho_1 \\ \Delta; \Gamma \vdash_{\text{exp}} e_2 : (\mathsf{int}, \rho_2), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho_1\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e_1 \oslash e_2 : \mathsf{bool}, \epsilon}$$

$$\frac{}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{tt} : \mathsf{bool}, \epsilon} \qquad \frac{}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{ff} : \mathsf{bool}, \epsilon}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_b : \mathsf{bool}, \epsilon \\ \Delta; \Gamma \vdash_{\text{exp}} e_t : \tau, \epsilon \qquad \Delta; \Gamma \vdash_{\text{exp}} e_f : \tau, \epsilon\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f : \tau, \epsilon} \qquad \frac{\Gamma(x) = \tau}{\Delta; \Gamma \vdash_{\text{exp}} x : \tau, \epsilon}$$

$$\frac{\begin{array}{c}x \notin dom(\Gamma) \qquad \Delta \vdash_{\text{type}} \tau_1 \qquad \Delta \vdash_{\text{place}} \epsilon' \\ \Delta; \Gamma, x : \tau_1 \vdash_{\text{exp}} e : \tau_2, \epsilon' \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \lambda x : \tau_1.^{\epsilon'} e \ \mathsf{at}\ \rho : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), \epsilon}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho \\ \Delta; \Gamma \vdash_{\text{exp}} e_2 : \tau_1, \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \epsilon'\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e_1 \ e_2 : \tau_2, \epsilon}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : \tau_1, \epsilon \\ \Delta; \Gamma \vdash_{\text{exp}} e_2 : \tau_2, \epsilon \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} (e_1, e_2) \ \mathsf{at}\ \rho : (\tau_1 \times \tau_2, \rho), \epsilon} \qquad \frac{\Delta; \Gamma \vdash_{\text{exp}} e : (\tau_1 \times \tau_2, \rho), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{fst}\ \ e : \tau_1, \epsilon}$$

$$\frac{\Delta; \Gamma \vdash_{\text{exp}} e : (\tau_1 \times \tau_2, \rho), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{snd}\ \ e : \tau_2, \epsilon} \qquad \frac{\begin{array}{c}\varrho \notin dom(\Delta) \qquad \Delta \vdash_{\text{type}} \tau \\ \Delta, \varrho \succeq \{\epsilon\}; \Gamma \vdash_{\text{exp}} e : \tau, \varrho\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{new}\ \varrho.e : \tau, \epsilon}$$

$$\frac{\begin{array}{c}\varrho \notin dom(\Delta) \qquad \Delta \vdash_{\text{eff}} \varphi \qquad \Delta \vdash_{\text{place}} \epsilon' \\ \Delta, \varrho \succeq \varphi; \Gamma \vdash_{\text{exp}} u : \tau, \epsilon' \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \lambda \varrho \succeq \varphi.^{\epsilon'} u \ \mathsf{at}\ \rho : (\Pi \varrho \succeq \varphi.^{\epsilon'} \tau, \rho), \epsilon}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e : (\Pi \varrho \succeq \varphi.^{\epsilon'} \tau, \rho'), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho' \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{re}} \rho \succeq \varphi \\ \Delta \vdash_{\text{rr}} \epsilon \succeq \epsilon'[\rho/\varrho]\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e \ [\rho] : \tau[\rho/\varrho], \epsilon} \qquad \frac{\begin{array}{c}f \notin dom(\Gamma) \qquad \Delta \vdash_{\text{type}} \tau \\ \Delta; \Gamma, f : \tau \vdash_{\text{exp}} u : \tau, \epsilon\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{fix}\ f : \tau.u : \tau, \epsilon}$$

$\boxed{\vdash_{\text{prog}} p \ \mathsf{ok}}$

$$\frac{\cdot, \mathcal{H} \succeq \{\}; \cdot \vdash_{\text{exp}} p : \mathsf{bool}, \mathcal{H}}{\vdash_{\text{prog}} p \ \mathsf{ok}}$$

Figure 3: Single Effect Calculus: Static Semantics (II)

2.4 Discussion

An important issue to consider is the expressiveness of the Single Effect Calculus relative to Tofte and Talpin's original region calculus. Tofte and Talpin's formulation of the region calculus as the implicit target of an inference system makes a direct comparison difficult. Fortunately, there has been sufficient interest in region-based memory management to warrant direct presentations of the region calculus [10, 3, 4, 11], which are better suited for comparison. Three aspects of the region calculus are highlighted as essential features: region polymorphism, region polymorphic recursion, and effect polymorphism.

The Single Effect Calculus clearly supports region polymorphism (albeit, in a slightly different form). Furthermore, region polymorphic recursion is supported by fixing a region abstraction, as demonstrated in the *fact* example. One can give a straightforward translation from a source region calculus without effect polymorphism into the Single Effect Calculus. At the type level, this transation expands every function type into a function and region abstraction type:

$$\mathcal{T}\left[\!\!\left[(\tau_1 \xrightarrow{\varphi} \tau_2, \rho)\right]\!\!\right] = (\mathcal{T}\,[\![\tau_1]\!] \xrightarrow{\rho} (\Pi\varepsilon \succeq \varphi.^{\varepsilon}\mathcal{T}\,[\![\tau_2]\!], \rho), \rho)$$

At the term level, source functions become functions and region abstractions and applications become applications and region applications. A similar approach deals with region abstractions in the source language. Essentially, this translation works by looking for the places where region sets are used in the source calculus and simply replacing them by an abstraction over that set. Clearly, this is not the most efficient translation. For example, in places where we could statically identify an upper bound on the region set (e.g., a singleton region set), we could elide the abstraction and simply use the upper bound.

Effect polymorphism can be simulated in the Single Effect Calculus, although at a heavier notational cost. Recall that effect polymorphism provides a means to abstract over an entire set of regions. Effect instantiation applies an effect abstraction to a set of regions. Effect polymorphism is especially useful for typing higher-order functions. For example, the type of the list `map` function is polymorphic in the effect of the functional argument.

Encoding effect polymorphism in the Single Effect Calculus begins by replacing effect abstractions ($\forall\varepsilon.^{\rho}\tau$) by region abstractions with an empty bound ($\Pi\varepsilon \succeq \{\}.^{\rho}\tau$). Effect instantiation must be translated to region instantiation; in particular, the set of regions must be translated to a single region denoting the upper bound of the set. In the presence of region polymorphism, this can be complicated, because a set of region variables may have no obvious upper bound. Hence, we must extend the translation to include upper bounds for each set of region variables that may be used in an effect instantiation. For example, a source type like $(\Pi\varrho_1.^{\varphi_1}(\Pi\varrho_2.^{\varphi_2}.(\Pi\varrho_3.^{\varphi_3}\tau, \rho_3), \rho_2), \rho_3)$ (where any subset of $\{\varrho_1, \varrho_2, \varrho_3\}$ may be used in an effect instantiation) is translated to

$$\begin{array}{l}
(\Pi\varrho_1 \succeq \{\}.^{\rho_1}(\Pi\varepsilon_1 \succeq \varphi_1.^{\varepsilon_1} \\
\quad(\Pi\varrho_2 \succeq \{\}.^{\rho_2}(\Pi\varepsilon_2 \succeq \varphi_2.^{\varepsilon_2} \\
\quad\quad(\Pi\varrho_{12} \succeq \{\varrho_1, \varrho_2\}.^{\rho_2} \\
\quad\quad\quad(\Pi\varrho_3 \succeq \{\}.^{\rho_3}(\Pi\varepsilon_3 \succeq \varphi_3.^{\varepsilon_3} \\
\quad\quad\quad\quad(\Pi\varrho_{13} \succeq \{\varrho_1, \varrho_3\}.^{\rho_3}(\Pi\varrho_{23} \succeq \{\varrho_2, \varrho_3\}.^{\rho_3} \\
\quad\quad\quad\quad\quad(\Pi\varrho_{123} \succeq \{\varrho_1, \varrho_2, \varrho_3\}.^{\rho_3} \\
\quad\quad\quad\quad\quad\quad(\mathcal{T}\,[\![\tau]\!], \rho_3), \rho_3), \rho_3), \rho_3), \rho_3), \rho_3), \rho_2).\rho_2), \rho_2), \rho_1), \rho_1)
\end{array}$$

$$\begin{array}{rcl}
i & \in & \mathbb{Z} \\
\alpha & \in & \mathit{TVars}^{FRGN} \\
f, x & \in & \mathit{Vars}^{FRGN} \\
\\
\multicolumn{3}{l}{\text{Types}} \\
\tau & ::= & \mathsf{int} \mid \mathsf{bool} \mid \tau_1 \to \tau_2 \mid \tau_1 \times \cdots \times \tau_n \mid \alpha \mid \forall\alpha.\tau \mid \\
 & & \mathsf{RGN}\ \tau_r\ \tau_a \mid \mathsf{RGNVar}\ \tau_r\ \tau_a \mid \mathsf{RGNHandle}\ \tau_r \\
\\
\multicolumn{3}{l}{\text{Terms}} \\
e & ::= & i \mid e_1 \oplus e_2 \mid e_1 \otimes e_3 \mid \\
 & & \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f \mid \\
 & & x \mid \lambda x : \tau.e \mid e_1\ e_2 \mid (e_1, \ldots, e_n) \mid \mathsf{sel}_i\ e \mid \\
 & & \Lambda\alpha.e \mid e\ [\tau] \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \\
 & & \mathsf{runRGN}\ [\tau_a]\ v \mid \kappa \\
\multicolumn{3}{l}{\text{Commands}} \\
\kappa & ::= & \underline{\mathsf{returnRGN}}\ [\tau_r]\ [\tau_a]\ v \mid \\
 & & \underline{\mathsf{thenRGN}}\ [\tau_r]\ [\tau_a]\ [\tau_b]\ v_1\ v_2 \mid \\
 & & \underline{\mathsf{allocRGNVar}}\ [\tau_r]\ [\tau_a]\ v_r\ v_a \mid \\
 & & \underline{\mathsf{readRGNVar}}\ [\tau_r]\ [\tau_a]\ v \mid \\
 & & \underline{\mathsf{fixRGNVar}}\ [\tau_r]\ [\tau_a]\ v_r\ v_f \mid \\
 & & \underline{\mathsf{newRGN}}\ [\tau_r]\ [\tau_a]\ v \\
\multicolumn{3}{l}{\text{Values}} \\
v & ::= & i \mid \mathsf{tt} \mid \mathsf{ff} \mid x \mid \lambda x : \tau.e \mid \Lambda\alpha.e \mid (v_1, v_2) \mid \kappa
\end{array}$$

Figure 4: $\mathsf{F}^{\mathsf{RGN}}$: Syntax

In the term translation, ϱ_{12}, ϱ_{23}, and ϱ_{123} can be used for region instantiations when the source term performs an effect instantiation with the corresponding set of region variables. The burden of instantiating ϱ_{12}, ϱ_{23}, and ϱ_{123} falls to the term that instantiates ϱ_1, ϱ_2, and ϱ_3, which will have sufficient information to choose the right upper bounds.

A formal translation could be given, but doing so detracts from the work at hand.

3. THE $\mathsf{F}^{\mathsf{RGN}}$ CALCULUS

The language $\mathsf{F}^{\mathsf{RGN}}$ is an extension of System F [20, 7] (also referred to as the polymorphic λ-calculus), adding monadic types and operations and taking inspiration from the work on monadic state [14, 15, 16, 1, 23, 19]. Essentially, $\mathsf{F}^{\mathsf{RGN}}$ uses an explicit region monad to enforce the locality of region allocated values.

Figure 4 presents the syntax of "initial programs" (that is, excluding intermediate terms that would appear in an operational semantics) of $\mathsf{F}^{\mathsf{RGN}}$. Figure 5 presents a type system for this external language. Figure 11 in the Appendix presents a simple large-step operational semantics in terms of a run-time store. In the following sections, we explain and motivate the main constructs and typing rules of $\mathsf{F}^{\mathsf{RGN}}$.

3.1 Types

Types in $\mathsf{F}^{\mathsf{RGN}}$ are similar to those found in System F. We include the primitives types int and bool, function and product types, and type abstractions. In addition, we have $\mathsf{RGN}\ \tau_r\ \tau_a$ as the type of monadic region computations, $\mathsf{RGNVar}\ \tau_r\ \tau_a$ as the type of region allocated values, and $\mathsf{RGNHandle}\ \tau_r$ as the type of region handles. Intuitively, $\mathsf{RGN}\ \tau_r\ \tau_a$ is the type of computations that yield values of type τ_a and that take place in the region indexed by the type τ_r. Likewise, $\mathsf{RGNVar}\ \tau_r\ \tau_a$ is the type of values of type

τ_a allocated in the region indexed by the type τ_r. Finally, $\mathsf{RGNHandle}\ \tau_r$ is the type of handles for the region indexed by the type τ_r. A value of such a type is a *region handle* – a run-time value holding the data necessary to allocate values within a region. Region indices (types) and region handles (values) are distinguished in order to maintain a phase distinction between compile-time and run-time expressions and to more accurately reflect implementation of regions. Region indices, like other types, have no run-time significance and may be erased from compiled code. On the other hand, region handles are necessary at run-time to allocate values within a region.

Although the remainder of this paper will never require a region index to be represented by anything other than a type variable, we choose to allow an arbitrary type in the first argument of the RGN monad type. We can thus interpret RGN as a primitive type constructor, without any special restrictions that may not be expressible in a practical programming language. Furthermore, in a semantics not based on type-erasure, type variables used as region indices will be instantiated with region types (see Figure 11).

3.2 Terms

As with types, most of the terms in $\mathsf{F}^{\mathsf{RGN}}$ are similar to those found in System F. Constants, arithmetic and boolean operations, function abstraction and application, tuple introduction and elimination, and type abstraction and instantiation are all completely standard.

We let κ range over the syntactic class of monadic commands. (Equivalently, and as suggested by the explicit type annotations and the restriction of sub-expressions to values, we can consider the monadic commands as constants with polymorphic types in a call-by-value interpretation of $\mathsf{F}^{\mathsf{RGN}}$.) Each command corresponds to a particular transformation on a monadic region. The commands $\underline{\mathsf{returnRGN}}$ and $\underline{\mathsf{thenRGN}}$ are the *unit* and *bind* operations of the region monad respectively.

The command $\underline{\mathsf{allocRGNVar}}\ [\tau_r]\ [\tau_a]\ v_r\ v_a$ allocates the value v_a of type τ_a in the region indexed by the type τ_r. The additional value v_r is the region handle for the region indexed by τ_r, which is necessary to allocate values within the region.

The command $\underline{\mathsf{readRGNVar}}\ [\tau_r]\ [\tau_a]\ v$ reads a value of type τ_a stored at the location v in the region indexed by the type τ_r. The command $\underline{\mathsf{fixRGNVar}}\ [\tau_r]\ [\tau_a]\ v_r\ v_f$ allocates a value of type τ_a in the region indexed by τ_r; the value is produced by the function v_f which is applied to the location where the allocated value is to be stored. This provides a means of allocating recursive structures. We discuss this command in more detail when we examine the typing rules for $\mathsf{F}^{\mathsf{RGN}}$.

The command $\underline{\mathsf{newRGN}}\ [\tau_r]\ [\tau_a]\ v$ first, creates a new region, executes the region computation described by v in the new region, and finally deallocates the new region. This entire execution is a computation that yields a value of type τ_a taking place in the region indexed by τ_r. We will have more to say about the computation described by v shortly.

Finally, the expression $\mathsf{runRGN}\ [\tau_a]\ v$ eliminates region-transformer operations. In particular, if v describes a region computation yielding a value of type τ_a, then $\mathsf{runRGN}\ [\tau_a]\ v$ executes that computation in a region, returning the final value produced by v and destroying the region (and any values allocated within it). The region (and any new regions introduced by $\underline{\mathsf{newRGN}}$) is neither accessible nor visible from outside the $\mathsf{runRGN}\ [\tau_a]\ v$ expression.

3.3 Typing Judgements

The typing rules for $\mathsf{F}^{\mathsf{RGN}}$ appear in Figure 5. Type contexts Δ are ordered lists of type variables and value contexts Γ are ordered lists of variables and types. We introduce the (suggestive) abbreviation $\tau_r \preceq \tau_s$ for a function that coerces any computation taking place in the region indexed by τ_r into a computation taking place in the region indexed by τ_s. We call such functions *witnesses* and explain their role in more detail below.

The only typing judgement of interest is $\Delta;\Gamma \vdash_{\mathrm{exp}} e : \tau$ meaning that term e has type τ. The rules for constants, arithmetic and boolean operations, function abstraction and application, tuple introduction and elimination, and type abstraction and instantiation are all completely standard. As expected in a monadic language, each command expression is given the monadic type $\mathsf{RGN}\ \tau_r\ \tau_a$ for appropriate region index and return type. The typing rules for $\underline{\mathsf{returnRGN}}$ and $\underline{\mathsf{thenRGN}}$ correspond to the standard typing rules for monadic unit and bind operations. The typing rules for $\underline{\mathsf{allocRGNVar}}$ and $\underline{\mathsf{readRGNVar}}$ are straight-forward.

As in the Single Effect Calculus, the key judgements are those relating to the creation of new regions. We first examine the typing rule for the runRGN expression:

$$\frac{\Delta;\Gamma \vdash_{\mathrm{exp}} v : \forall\alpha.\mathsf{RGNHandle}\ \alpha \to \mathsf{RGN}\ \alpha\ \tau_a}{\Delta;\Gamma \vdash_{\mathrm{exp}} \mathsf{runRGN}\ [\tau_a]\ v : \tau_a}$$

As stated above, the argument to runRGN should describe a region computation. In fact, we require v to be a polymorphic function that yields a region computation after being applied to a region handle. Recall that we can consider a value of type $\mathsf{RGN}\ \tau_r\ \tau_a$ as a region-transformer – that is, it accepts a region (indexed by the type τ_r), performs some operations (such as allocating into the region), and returns a value and the modified region. The effect of universally quantifying the region index in the type of v is to require v to make no assumptions about the input region (e.g., the existence of pre-allocated values). Furthermore, all operations that manipulate a region are "infected" with the region index: when combining operations, the rule for $\underline{\mathsf{returnRGN}}$ requires the region index type to be the same; locations allocated and read using $\underline{\mathsf{allocRGNVar}}$ and $\underline{\mathsf{readRGNVar}}$ require the region index of the RGNVar to be the same as the computation in which the operation occurs. Thus, if a region computation $\mathsf{RGN}\ \tau_r\ \tau_a$ were to return a value that depended upon the region indexed by τ_r, then τ_r would appear in the type τ_a. Since the type τ_a appears outside the scope of the type variable α in the typing rule for runRGN, it follows that α cannot appear in the type τ_a. Therefore, it must be the case that the value returned by the computation described by v does not depend upon the region index which will instantiate α. Taken together, these facts ensure that an arbitrary new region can be supplied to the computation and that the value returned will not leak any means of accessing the region or values allocated within it; hence, the region can be destroyed at the end of the computation. Finally, because we require region handles for allocating within regions, we provide the region handle for the newly created region as the argument to a function that yields the computation we wish to execute.

$$\begin{array}{ll} \text{Type contexts} & \text{Value contexts} \\ \Delta \quad ::= \quad \cdot \mid \Delta, \alpha & \Gamma \quad ::= \quad \cdot \mid \Gamma, x : \tau \end{array}$$

$$\tau_r \preceq \tau_s \quad \equiv \quad \forall \alpha. \mathsf{RGN}\ \tau_r\ \alpha \to \mathsf{RGN}\ \tau_s\ \alpha$$

$\boxed{\vdash_{\text{tctxt}} \Delta}$

$$\frac{}{\vdash_{\text{tctxt}} \cdot} \qquad \frac{\vdash_{\text{rctxt}} \Delta \qquad \alpha \notin \mathit{dom}(\Delta)}{\vdash_{\text{tctxt}} \Delta, \alpha}$$

$\boxed{\Delta \vdash_{\text{type}} \tau}$

$$\frac{}{\Delta \vdash_{\text{type}} \mathsf{int}} \qquad \frac{}{\Delta \vdash_{\text{type}} \mathsf{bool}} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash_{\text{type}} \tau_1 \to \tau_2} \qquad \frac{\Delta \vdash_{\text{type}} \tau_i \quad ^{i \in 1 \ldots n}}{\Delta \vdash_{\text{type}} \tau_1 \times \cdots \times \tau_n} \qquad \frac{\alpha \in \mathit{dom}(\Delta)}{\Delta \vdash_{\text{type}} \alpha}$$

$$\frac{\alpha \notin \mathit{dom}(\Delta) \qquad \Delta, \alpha \vdash_{\text{type}} \tau}{\Delta \vdash_{\text{type}} \forall \alpha. \tau} \qquad \frac{\Delta \vdash_{\text{type}} \tau_r \qquad \Delta \vdash_{\text{type}} \tau_a}{\Delta \vdash_{\text{type}} \mathsf{RGN}\ \tau_r\ \tau_a} \qquad \frac{\Delta \vdash_{\text{type}} \tau_r \qquad \Delta \vdash_{\text{type}} \tau_a}{\Delta \vdash \mathsf{RGNVar}\ \tau_r\ \tau_a}$$

$\boxed{\Delta \vdash_{\text{vctxt}} \Gamma}$

$$\frac{}{\Delta \vdash_{\text{vctxt}} \cdot} \qquad \frac{\Delta \vdash_{\text{vctxt}} \Gamma \qquad x \notin \mathit{dom}(\Gamma) \qquad \Delta \vdash_{\text{type}} \tau}{\Delta \vdash_{\text{vctxt}} \Gamma, x : \tau}$$

$\boxed{\vdash_{\text{ctxt}} \Delta; \Gamma}$

$$\frac{\vdash_{\text{tctxt}} \Delta \qquad \Delta \vdash_{\text{vctxt}} \Gamma}{\vdash_{\text{ctxt}} \Delta; \Gamma}$$

$\boxed{\Delta; \Gamma \vdash_{\text{exp}} e : \tau}$

$$\frac{}{\Delta; \Gamma \vdash_{\text{exp}} i : \mathsf{int}} \qquad \frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : \mathsf{int} \\ \Delta; \Gamma \vdash_{\text{exp}} e_2 : \mathsf{int}\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e_1 \oplus e_2 : \mathsf{int}} \qquad \frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : \mathsf{int} \\ \Delta; \Gamma \vdash_{\text{exp}} e_2 : \mathsf{int}\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e_1 \otimes e_2 : \mathsf{bool}} \qquad \frac{}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{tt} : \mathsf{bool}} \qquad \frac{}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{ff} : \mathsf{bool}}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_b : \mathsf{bool} \\ \Delta; \Gamma \vdash_{\text{exp}} e_t : \tau \qquad \Delta; \Gamma \vdash_{\text{exp}} e_f : \tau\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f : \tau} \qquad \frac{\Gamma(x) = \tau}{\Delta; \Gamma \vdash_{\text{exp}} x : \tau} \qquad \frac{\begin{array}{c}\Delta \vdash_{\text{type}} \tau_1 \qquad x \notin \mathit{dom}(\Gamma) \\ \Delta; \Gamma, x : \tau_1 \vdash_{\text{exp}} e : \tau_2\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \lambda x : \tau_1 . e : \tau_1 \to \tau_2} \qquad \frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : \tau_1 \to \tau_2 \\ \Delta; \Gamma \vdash_{\text{exp}} e_2 : \tau_1\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e_1\ e_2 : \tau_2}$$

$$\frac{\Delta; \Gamma \vdash_{\text{exp}} e_1 : \tau_i \quad ^{i \in 1 \ldots n}}{\Delta; \Gamma \vdash_{\text{exp}} (e_1, \ldots, e_n) : \tau_1 \times \cdots \times \tau_n} \qquad \frac{\begin{array}{c}1 \leq i \leq n \\ \Delta; \Gamma \vdash_{\text{exp}} e : \tau_1 \times \cdots \times \tau_n\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{sel}_i\ e : \tau_i} \qquad \frac{\begin{array}{c}\alpha \notin \mathit{dom}(\Delta) \\ \Delta, \alpha; \Gamma \vdash_{\text{exp}} e : \tau\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \Lambda \alpha . e : \forall \alpha . \tau} \qquad \frac{\begin{array}{c}\Delta \vdash_{\text{type}} \tau' \\ \Delta; \Gamma \vdash_{\text{exp}} e : \forall \alpha . \tau\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} e\ [\tau'] : \tau[\tau'/\alpha]}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} e_1 : \tau_1 \\ x \notin \mathit{dom}(\Gamma) \qquad \Delta; \Gamma, x : \tau_1 \vdash_{\text{exp}} e_2 : \tau_2\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2} \qquad \frac{\Delta; \Gamma \vdash_{\text{exp}} v : \forall \alpha . \mathsf{RGNHandle}\ \alpha \to \mathsf{RGN}\ \alpha\ \tau_a}{\Delta; \Gamma \vdash_{\text{exp}} \mathsf{runRGN}\ [\tau_a]\ v : \tau_a}$$

$$\frac{\Delta \vdash_{\text{type}} \tau_r \qquad \Delta; \Gamma \vdash_{\text{exp}} v : \tau_a}{\Delta; \Gamma \vdash_{\text{exp}} \underline{\mathsf{returnRGN}}\ [\tau_r]\ [\tau_a]\ v : \mathsf{RGN}\ \tau_r\ \tau_a} \qquad \frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} v_1 : \mathsf{RGN}\ \tau_r\ \tau_a \\ \Delta; \Gamma \vdash_{\text{exp}} v_2 : \tau_a \to \mathsf{RGN}\ \tau_r\ \tau_b\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \underline{\mathsf{thenRGN}}\ [\tau_r]\ [\tau_a]\ [\tau_b]\ v_1\ v_2 : \mathsf{RGN}\ \tau_r\ \tau_b}$$

$$\frac{\Delta; \Gamma \vdash_{\text{exp}} v_1 : \mathsf{RGNHandle}\ \tau_r \qquad \Delta; \Gamma \vdash_{\text{exp}} v_2 : \tau_a}{\Delta; \Gamma \vdash_{\text{exp}} \underline{\mathsf{allocRGNVar}}\ [\tau_r]\ [\tau_a]\ v_1\ v_2 : \mathsf{RGN}\ \tau_r\ (\mathsf{RGNVar}\ \tau_r\ \tau_a)} \qquad \frac{\Delta; \Gamma \vdash_{\text{exp}} v : \mathsf{RGNVar}\ \tau_r\ \tau_a}{\Delta; \Gamma \vdash_{\text{exp}} \underline{\mathsf{readRGNVar}}\ [\tau_r]\ [\tau_a]\ v : \mathsf{RGN}\ \tau_r\ \tau_a}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash_{\text{exp}} v_1 : \mathsf{RGNHandle}\ \tau_r \\ \Delta; \Gamma \vdash_{\text{exp}} v_2 : \mathsf{RGNVar}\ \tau_r\ \tau_a \to \tau_a\end{array}}{\Delta; \Gamma \vdash_{\text{exp}} \underline{\mathsf{fixRGNVar}}\ [\tau_r]\ [\tau_a]\ v_1\ v_2 : \mathsf{RGN}\ \tau_r\ (\mathsf{RGNVar}\ \tau_r\ \tau_a)} \qquad \frac{\Delta; \Gamma \vdash_{\text{exp}} v : \forall \alpha . \tau_r \preceq \alpha \to \mathsf{RGNHandle}\ \alpha \to \mathsf{RGN}\ \alpha\ \tau_a}{\Delta; \Gamma \vdash_{\text{exp}} \underline{\mathsf{newRGN}}\ [\tau_r]\ [\tau_a]\ v : \mathsf{RGN}\ \tau_r\ \tau_a}$$

Figure 5: $\mathsf{F}^{\mathsf{RGN}}$: Static Semantics

The typing rule for <u>newRGN</u> is very similar:

$$\frac{\Delta; \Gamma \vdash_{\text{exp}} v : \forall \alpha. \tau_r \preceq \alpha \to \mathsf{RGNHandle}\ \alpha \to \mathsf{RGN}\ \alpha\ \tau_a}{\Delta; \Gamma \vdash_{\text{exp}} \underline{\mathsf{newRGN}}\ [\tau_r]\ [\tau_a]\ v : \mathsf{RGN}\ \tau_r\ \tau_a}$$

Ignoring for the moment the argument of type $\tau_r \preceq \alpha$, we see that exactly the same argument as above applies. In particular, the computation makes no assumptions about the newly created region, nor can the region be leaked through the returned value of type τ_a. What, then, is the role of the witness argument? The answer lies in the fact that we do not really intend the execution to take place in an arbitrary region. Instead, we expect the newly allocated region to be related to previously allocated regions according to a stack discipline (just as in region calculi). Hence, the notion of "execution taking place in a region" is somewhat inaccurate; instead, we have executions taking place in a stack of regions. The region index in a type RGN τ_r τ_a indicates a particular member of the region stack; in practice, it often coincides with the most recently allocated region. Thus, any computation taking place in a stack of regions where τ_s is a member (i.e., a RGN τ_s τ_a term) is also a computation taking place in a stack of regions where τ_r is a member (i.e., a RGN τ_r τ_a term) whenever τ_s outlives τ_r. A function of type $\tau_s \preceq \tau_r$ witnesses this coercion. This explains the role of the witness argument – it is provided to the computation taking place in the inner region in order to coerce computations (such as allocating a new value in the outer region) from the outer region to the inner region. Operationally, such a witness function acts as the identity function.

Finally, we note that the typing rule for <u>fixRGNVar</u> requires that the function v_f has the type RGNVar τ_r $\tau_a \to \tau_a$. Note that this is a pure function, not a monadic computation. Hence, it is safe to pass the location where the allocated value is to be stored, because no computation (hence, no reading of region allocated values) can occur during the evaluation of the application of v_f to a location. On the other hand, v_f can return a computation that reads the allocated value, since this computation cannot occur until after the knot has been tied.

4. THE TRANSLATION

In this section we present a type-preserving translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$. Many of the key components of the translation should be obvious from the suggestive naming of the previous sections. We clearly intend new to be translated (in some fashion) to <u>newRGN</u>. Likewise, we can expect types of the form (μ, ρ) to be translated to types of the form RGNVar. It further seems likely that the outlives relation $\epsilon \succeq \rho$ should be related to the witness functions $\tau_r \preceq \tau_s$. We present the translation in stages, as there are some subtleties that require explanation.

We start with a few preliminaries. We assume injections from the sets $RVars^{SEC}$ and $Vars^{SEC}$ to the sets $TVars^{FRGN}$ and $Vars^{FRGN}$ respectively. In the translation, applications of such injections will be clear from context and we freely use variables from source objects in target objects. We further assume two additional injections from the set $RVars^{SEC}$ to the set $Vars^{FRGN}$; the image of a region variable ϱ under these injections are written ϱ^h and ϱ^w respectively.

The translation is a typed call-by-value monad translation, similar to the standard translation given by Sabry and Wadler [22]. We have not attempted to optimize the translation to avoid the introduction of "administrative" redexes. We feel that this simplifies the translation, although it is likely to complicate a proof that the translation preserves the semantics. We intend to investigate an optimized translation in future work.

Figure 6 shows the translation of types and contexts. As expected, the type (μ, ρ) is translated to RGNVar ρ $\mathcal{T}_\mu [\![\mu]\!]$, whereby region allocated values in the source are also region allocated in the target. The translations of primitive types and product types are trivial. More interesting are the translations of function types and region abstraction types. Functions with effects bounded by the region ϵ are translated into pure functions that yield computations taking place in stack of regions with ϵ as a member. Region abstractions are translated into type abstractions. Because the target calculus requires explicit region handles for allocation, each time a region is in scope in the source calculus, the region handle must be in scope in the target calculus. This explains the appearance of the RGNHandle ϱ type in the translation. Likewise, the target calculus makes witness functions explicit, whereas in the source calculus such coercions are implied by $\succeq$ related regions. Hence, we interpret $\varrho \succeq \{\rho_1, \ldots, \rho_n\}$ as an n-tuple of functions, each witnessing a coercion from region ρ_i to ϱ. This interpretation is formalized by the $\mathcal{T}_\succeq [\![\cdot]\!]$ translations.[2]

We extend the type translation to contexts in the obvious way. In addition to translating region variables to type variables and translating the types of variables in value contexts, we have additional translations from region contexts to value contexts. As explained above, region handles and witness functions are explicit values in the target calculus. Hence, our translation maintains the invariant that whenever a region variable ϱ is in scope in the source calculus, the variables ϱ^h and ϱ^w are in scope in the target calculus. The variable ϱ^h (of type RGNHandle ϱ) is the handle for the region ϱ and the variable ϱ^w (of type $\mathcal{T}_\succeq [\![\varrho \succeq \varphi]\!]$) is the tuple holding the witness functions that coerce to region ϱ.

Figure 7 shows the translation of witness terms. The first three translations map the reflexive, transitive closure of the syntactic constraints in a source Δ into an appropriate coercion function. The final translation collects a set of coercion functions into a tuple; such a term is suitable as an argument to the translation of a region abstraction.

Figure 8 shows the translation of key terms. Figures 12 and 13 in the Appendix show the translation of the other terms; these additional translations are straight-forward in light of the translations given in Figure 8. In order to make the translation easier to read, we introduce the following notation:

$$\begin{aligned} \mathsf{bind}\ f : \tau_a \Leftarrow e_1; e_2 \equiv\ & \mathsf{let}\ k = e_1\ \mathsf{in} \\ & \underline{\mathsf{thenRGN}}\ [\tau_r]\ [\tau_a]\ [\tau_b]\ k\ (\lambda f : \tau_a . e_2) \\ & \text{where } k \text{ fresh} \end{aligned}$$

[2]Note that in the Single Effect Calculus, we only substitutes regions for region variables. This means that the sets of regions that appear in the program never change size (although they may change elements as a result of substitution). The $\mathcal{T}_\succeq [\![\cdot]\!]$ translations require keeping the ordering of regions in a set $\{\rho_1, \ldots, \rho_n\}$ constant. It does not require a global ordering on region variables; such an ordering would not suffice for our purposes, because the ordering of elements in a set might change after substitution. Instead, we take $\{\rho_1, \ldots, \rho_n\}$ as an ordered list, where substitution preserves the order.

Lemma 1 (Translation preserves types (types and contexts)).

- *If* $\vdash_{\text{rctxt}} \Delta$, *then* $\vdash_{\text{rctxt}} \mathcal{C}^{\mathcal{T}}_{\Delta} [\![\Delta]\!]$.
- *If* $\vdash_{\text{rctxt}} \Delta$ *and* $\Delta \vdash_{\text{btype}} \mu$, *then* $\mathcal{C}^{\mathcal{T}}_{\Delta} [\![\Delta]\!] \vdash_{\text{type}} \mathcal{T}_{\mu} [\![\mu]\!]$.
- *If* $\vdash_{\text{rctxt}} \Delta$ *and* $\Delta \vdash_{\text{type}} \tau$, *then* $\mathcal{C}^{\mathcal{T}}_{\Delta} [\![\Delta]\!] \vdash_{\text{type}} \mathcal{T}_{\tau} [\![\tau]\!]$.
- *If* $\vdash_{\text{rctxt}} \Delta$ *and* $\Delta \vdash_{\text{vctxt}} \Gamma$, *then* $\mathcal{C}^{\mathcal{E}}_{\Delta} [\![\Delta]\!] \vdash_{\text{vctxt}} \mathcal{C}^{\mathcal{E}}_{\Delta} [\![\Delta]\!], \mathcal{C}^{\mathcal{E}}_{\Gamma} [\![\Gamma]\!]$.

Translations yielding types

Types

$$\begin{array}{rcl}
\mathcal{T}_{\tau} [\![\mathsf{bool}]\!] & = & \mathsf{bool} \\
\mathcal{T}_{\tau} [\![(\mu, \rho)]\!] & = & \mathsf{RGNVar}\ \rho\ \mathcal{T}_{\mu} [\![\mu]\!]
\end{array}$$

Boxed types

$$\begin{array}{rcl}
\mathcal{T}_{\mu} [\![\mathsf{int}]\!] & = & \mathsf{int} \\
\mathcal{T}_{\mu} \left[\!\!\left[\tau_1 \xrightarrow{\epsilon} \tau_2\right]\!\!\right] & = & \mathcal{T}_{\tau} [\![\tau_1]\!] \rightarrow \mathsf{RGN}\ \epsilon\ \mathcal{T}_{\tau} [\![\tau_1]\!] \\
\mathcal{T}_{\mu} [\![\tau_1 \times \tau_2]\!] & = & \mathcal{T}_{\tau} [\![\tau_1]\!] \times \mathcal{T}_{\tau} [\![\tau_2]\!] \\
\mathcal{T}_{\mu} [\![\Pi \varrho \succeq \varphi.^{\varepsilon} \tau]\!] & = & \forall \varrho. \mathcal{T}_{\succeq} [\![\varrho \succeq \varphi]\!] \rightarrow \mathsf{RGNHandle}\ \varrho \rightarrow \mathsf{RGN}\ \epsilon\ \mathcal{T}_{\tau} [\![\tau]\!]
\end{array}$$

Effects

$$\begin{array}{rcl}
\mathcal{T}_{\succeq} [\![\epsilon \succeq \rho]\!] & = & \rho \preceq \epsilon \\
\mathcal{T}_{\succeq} [\![\epsilon \succeq \{\rho_1, \ldots, \rho_n\}]\!] & = & \mathcal{T}_{\succeq} [\![\epsilon \succeq \rho_1]\!] \times \cdots \times \mathcal{T}_{\succeq} [\![\epsilon \succeq \rho_n]\!]
\end{array}$$

Translations yielding type contexts

Region contexts

$$\begin{array}{rcl}
\mathcal{C}^{\mathcal{T}}_{\Delta} [\![\cdot]\!] & = & \cdot \\
\mathcal{C}^{\mathcal{T}}_{\Delta} [\![\Delta, \varrho \succeq \varphi]\!] & = & \mathcal{C}^{\mathcal{T}}_{\Delta} [\![\Delta]\!], \varrho
\end{array}$$

Translations yielding value contexts

Region contexts

$$\begin{array}{rcl}
\mathcal{C}^{\mathcal{E}}_{\Delta} [\![\cdot]\!] & = & \cdot \\
\mathcal{C}^{\mathcal{E}}_{\Delta} [\![\Delta, \varrho \succeq \varphi]\!] & = & \mathcal{C}^{\mathcal{E}}_{\Delta} [\![\Delta]\!], \varrho^{h} : \mathsf{RGNHandle}\ \varrho, \varrho^{w} : \mathcal{T}_{\succeq} [\![\varrho \succeq \varphi]\!]
\end{array}$$

Value contexts

$$\begin{array}{rcl}
\mathcal{C}^{\mathcal{E}}_{\Gamma} [\![\cdot]\!] & = & \cdot \\
\mathcal{C}^{\mathcal{E}}_{\Gamma} [\![\Gamma, x : \tau]\!] & = & \mathcal{C}^{\mathcal{E}}_{\Gamma} [\![\Gamma]\!], x : \mathcal{T}_{\tau} [\![\tau]\!]
\end{array}$$

Figure 6: Translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$ (Types and Contexts)

Lemma 2 (Translation preserves types (Witnesses)).

- *If* $\vdash_{\text{rctxt}} \Delta$ *and* $\Delta \vdash_{\text{rr}} \epsilon \succeq \rho$, *then* $\mathcal{C}^{\mathcal{T}}_{\Delta} [\![\Delta]\!] ; \mathcal{C}^{\mathcal{E}}_{\Delta} [\![\Delta]\!] \vdash_{\text{exp}} \mathcal{E}_{\vdash_{\text{rr}}} [\![\epsilon \succeq \rho]\!] : \mathcal{T}_{\succeq} [\![\epsilon \succeq \rho]\!]$.
- *If* $\vdash_{\text{rctxt}} \Delta$ *and* $\Delta \vdash_{\text{re}} \epsilon \succeq \varphi$, *then* $\mathcal{C}^{\mathcal{T}}_{\Delta} [\![\Delta]\!] ; \mathcal{C}^{\mathcal{E}}_{\Delta} [\![\Delta]\!] \vdash_{\text{exp}} \mathcal{E}_{\vdash_{\text{re}}} [\![\epsilon \succeq \varphi]\!] : \mathcal{T}_{\succeq} [\![\epsilon \succeq \varphi]\!]$.

Translations yielding terms (witnesses)

$$\begin{array}{rcl}
\mathcal{E}_{\vdash_{\text{rr}}} \left[\!\!\left[\dfrac{\Delta(\varepsilon) = \{\rho_1, \ldots, \rho_i, \ldots, \rho_n\}}{\Delta \vdash_{\text{rr}} \varepsilon \succeq \rho_i} \right]\!\!\right] & = & \mathsf{sel}_i\ \varepsilon^{w} \\
\mathcal{E}_{\vdash_{\text{rr}}} \left[\!\!\left[\dfrac{\Delta \vdash_{\text{place}} \epsilon}{\Delta \vdash_{\text{rr}} \epsilon \succeq \epsilon} \right]\!\!\right] & = & \Lambda \alpha. \lambda c : \mathsf{RGN}\ \epsilon\ \alpha. c \\
\mathcal{E}_{\vdash_{\text{rr}}} \left[\!\!\left[\dfrac{\Delta \vdash_{\text{rr}} \epsilon \succeq \epsilon' \qquad \Delta \vdash_{\text{rr}} \epsilon' \succeq \rho}{\Delta \vdash_{\text{rr}} \epsilon \succeq \rho} \right]\!\!\right] & = & \Lambda \alpha. \lambda c : \mathsf{RGN}\ \rho\ \alpha. \mathcal{E}_{\vdash_{\text{rr}}} [\![\Delta \vdash_{\text{rr}} \epsilon \succeq \epsilon']\!]\ [\alpha]\ (\mathcal{E}_{\vdash_{\text{rr}}} [\![\Delta \vdash_{\text{rr}} \epsilon' \succeq \rho]\!]\ [\alpha]\ c) \\
\mathcal{E}_{\vdash_{\text{re}}} \left[\!\!\left[\dfrac{\Delta \vdash_{\text{rr}} \epsilon \succeq \rho_i \quad {}^{i \in 1 \ldots n}}{\Delta \vdash_{\text{re}} \epsilon \succeq \{\rho_1, \ldots, \rho_n\}} \right]\!\!\right] & = & (\mathcal{E}_{\vdash_{\text{rr}}} [\![\Delta \vdash_{\text{rr}} \epsilon \succeq \rho_1]\!], \ldots, \mathcal{E}_{\vdash_{\text{rr}}} [\![\Delta \vdash_{\text{rr}} \epsilon \succeq \rho_n]\!])
\end{array}$$

Figure 7: Translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$ (witnesses)

Lemma 3 (Translation preserves types (terms)).

- *If* $\vdash_{\mathrm{ctxt}} \Delta;\Gamma;\epsilon$ *and* $\Delta;\Gamma \vdash_{\mathrm{exp}} e : \tau, \epsilon$, *then* $\mathcal{C}^{\mathcal{T}}_{\Delta} \llbracket \Delta \rrbracket ; \mathcal{C}^{\mathcal{E}}_{\Delta} \llbracket \Delta \rrbracket , \mathcal{C}^{\mathcal{E}}_{\Gamma} \llbracket \Gamma \rrbracket \vdash_{\mathrm{exp}} \mathcal{E}_{\vdash_{\mathrm{exp}}} \llbracket \Delta;\Gamma \vdash_{\mathrm{exp}} e : \tau, \epsilon \rrbracket : \mathsf{RGN}\ \epsilon\ \mathcal{T}_{\tau} \llbracket \tau \rrbracket$.

Translations yielding terms (terms)

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \frac{\Delta \vdash_{\mathrm{place}} \rho \qquad \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho}{\Delta;\Gamma \vdash_{\mathrm{exp}} i \ \mathsf{at}\ \rho : (\mathsf{int}, \rho), \epsilon} \right\rrbracket = \mathcal{E}_{\vdash_{\mathrm{rr}}} \llbracket \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \rrbracket\ [\mathcal{T}_{\tau} \llbracket (\mathsf{int}, \rho) \rrbracket]\ (\underline{\mathsf{allocRGNVar}}\ [\rho]\ [\mathcal{T}_{\mu} \llbracket \mathsf{int} \rrbracket]\ \rho^{h}\ i)$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \frac{\Gamma(x) = \tau}{\Delta;\Gamma \vdash_{\mathrm{exp}} x : \tau, \epsilon} \right\rrbracket = \underline{\mathsf{returnRGN}}\ [\epsilon]\ [\mathcal{T}_{\tau} \llbracket \tau \rrbracket]\ x$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \frac{\begin{array}{c} x \notin dom(\Gamma) \qquad \Delta \vdash_{\mathrm{type}} \tau_1 \qquad \Delta \vdash_{\mathrm{place}} \epsilon' \\ \Delta;\Gamma, x : \tau_1 \vdash_{\mathrm{exp}} e : \tau_2, \epsilon' \\ \Delta \vdash_{\mathrm{place}} \rho \qquad \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} \lambda x : \tau_1 .^{\epsilon'} e \ \mathsf{at}\ \rho : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), \epsilon} \right\rrbracket =$$

$$\begin{array}{l} \mathcal{E}_{\vdash_{\mathrm{rr}}} \llbracket \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \rrbracket\ [\mathcal{T}_{\tau} \left\llbracket (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho) \right\rrbracket] \\ \quad (\underline{\mathsf{allocRGNVar}}\ [\rho]\ [\mathcal{T}_{\mu} \left\llbracket \tau_1 \xrightarrow{\epsilon'} \tau_2 \right\rrbracket] \\ \quad\quad \rho^{h}\ (\lambda x : \mathcal{T}_{\tau} \llbracket \tau_1 \rrbracket . \mathcal{E}_{\vdash_{\mathrm{exp}}} \llbracket \Delta;\Gamma, x : \tau_1 \vdash_{\mathrm{exp}} e : \tau_2, \epsilon' \rrbracket)) \end{array}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \frac{\begin{array}{c} \Delta;\Gamma \vdash_{\mathrm{exp}} e_1 : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), \epsilon \qquad \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \\ \Delta;\Gamma \vdash_{\mathrm{exp}} e_2 : \tau_1, \epsilon \qquad \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \epsilon' \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} e_1\ e_2 : \tau_2, \epsilon} \right\rrbracket =$$

$$\begin{array}{l} \mathsf{bind}\ f : \mathcal{T}_{\tau} \left\llbracket (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho) \right\rrbracket \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \Delta;\Gamma \vdash_{\mathrm{exp}} e_1 : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), \epsilon \right\rrbracket ; \\ \mathsf{bind}\ g : \mathcal{T}_{\mu} \left\llbracket \tau_1 \xrightarrow{\epsilon'} \tau_2 \right\rrbracket \Leftarrow \mathcal{E}_{\vdash_{\mathrm{rr}}} \llbracket \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \rrbracket\ [\mathcal{T}_{\mu} \left\llbracket \tau_1 \xrightarrow{\epsilon'} \tau_2 \right\rrbracket]\ (\underline{\mathsf{readRGNVar}}\ [\rho]\ [\mathcal{T}_{\mu} \left\llbracket \tau_1 \xrightarrow{\epsilon'} \tau_2 \right\rrbracket]\ f); \\ \mathsf{bind}\ a : \mathcal{T}_{\tau} \llbracket \tau_1 \rrbracket \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} \llbracket \Delta;\Gamma \vdash_{\mathrm{exp}} e_2 : \tau_1, \epsilon \rrbracket ; \\ \mathcal{E}_{\vdash_{\mathrm{rr}}} \llbracket \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \epsilon' \rrbracket\ \mathcal{T}_{\tau} \llbracket \tau_2 \rrbracket\ (g\ a) \qquad \text{where } g, a \text{ fresh} \end{array}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \frac{\begin{array}{c} \varrho \notin dom(\Delta) \qquad \Delta \vdash_{\mathrm{type}} \tau \\ \Delta, \varrho \succeq \{\epsilon\};\Gamma \vdash_{\mathrm{exp}} e : \tau, \varrho \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} \mathsf{new}\ \varrho . e : \tau, \epsilon} \right\rrbracket = \begin{array}{l} \underline{\mathsf{newRGN}}\ [\epsilon]\ [\mathcal{T}_{\tau} \llbracket \tau \rrbracket]\ (\Lambda \varrho . \lambda \varrho^{w} : \mathcal{T}_{\succeq} \llbracket \varrho \succeq \{\epsilon\} \rrbracket . \lambda \varrho^{h} : \mathsf{RGNHandle}\ \varrho . \\ \qquad \mathcal{E}_{\vdash_{\mathrm{exp}}} \llbracket \Delta, \varrho \succeq \{\epsilon\};\Gamma \vdash_{\mathrm{exp}} e : \tau, \varrho \rrbracket) \end{array}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \frac{\begin{array}{c} \varrho \notin dom(\Delta) \qquad \Delta \vdash_{\mathrm{eff}} \varphi \qquad \Delta \vdash_{\mathrm{place}} \epsilon' \\ \Delta, \varrho \succeq \varphi;\Gamma \vdash_{\mathrm{exp}} u : \tau, \epsilon' \\ \Delta \vdash_{\mathrm{place}} \rho \qquad \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} \lambda \varrho \succeq \varphi .^{\epsilon'} u \ \mathsf{at}\ \rho : (\Pi \varrho \succeq \varphi .^{\epsilon'} \tau, \rho), \epsilon} \right\rrbracket =$$

$$\begin{array}{l} \mathcal{E}_{\vdash_{\mathrm{rr}}} \llbracket \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \rrbracket\ [\mathcal{T}_{\tau} \left\llbracket (\Pi \varrho .^{\epsilon'} \tau, \rho) \right\rrbracket] \\ \quad (\underline{\mathsf{allocRGNVar}}\ [\rho]\ [\mathcal{T}_{\mu} \left\llbracket \Pi \varrho .^{\epsilon'} \tau \right\rrbracket] \\ \quad\quad \rho^{h}\ (\Lambda \varrho . \lambda \varrho^{w} : \mathcal{T}_{\succeq} \llbracket \varrho \succeq \varphi \rrbracket . \lambda \varrho^{h} : \mathsf{RGNHandle}\ \varrho . \\ \quad\quad\quad \mathcal{E}_{\vdash_{\mathrm{exp}}} \llbracket \Delta, \varrho \succeq \varphi;\Gamma \vdash_{\mathrm{exp}} u : \tau, \epsilon' \rrbracket)) \end{array}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \frac{\begin{array}{c} \Delta;\Gamma \vdash_{\mathrm{exp}} e : (\Pi \varrho \succeq \varphi .^{\epsilon'} \tau, \rho'), \epsilon \qquad \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho' \\ \Delta \vdash_{\mathrm{place}} \rho \qquad \Delta \vdash_{\mathrm{re}} \rho \succeq \varphi \\ \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \epsilon'[\rho/\varrho] \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} e\ [\rho] : \tau[\rho/\varrho], \epsilon} \right\rrbracket =$$

$$\begin{array}{l} \mathsf{bind}\ f : \mathcal{T}_{\tau} \left\llbracket (\Pi \varrho .^{\epsilon'} \tau, \rho') \right\rrbracket \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} \left\llbracket \Delta;\Gamma \vdash_{\mathrm{exp}} e : (\Pi \varrho .^{\epsilon'} \tau, \rho'), \epsilon \right\rrbracket ; \\ \mathsf{bind}\ g : \mathcal{T}_{\mu} \left\llbracket \Pi \varrho .^{\epsilon'} \tau \right\rrbracket \Leftarrow \mathcal{E}_{\vdash_{\mathrm{rr}}} \llbracket \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho' \rrbracket\ [\mathcal{T}_{\mu} \left\llbracket \Pi \varrho .^{\epsilon'} \tau \right\rrbracket]\ (\underline{\mathsf{readRGNVar}}\ [\rho']\ [\mathcal{T}_{\mu} \left\llbracket \Pi \varrho .^{\epsilon'} \tau \right\rrbracket]\ f); \\ \mathcal{E}_{\vdash_{\mathrm{rr}}} \llbracket \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \epsilon' \rrbracket\ \mathcal{T}_{\tau} \llbracket \tau[\rho/\varrho] \rrbracket\ (g\ [\rho]\ \mathcal{E}_{\vdash_{\mathrm{re}}} \llbracket \Delta \vdash_{\mathrm{re}} \rho \succeq \varphi \rrbracket\ \rho^{h}) \qquad \text{where } g \text{ fresh} \end{array}$$

Figure 8: Translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$ (Terms (I))

LEMMA 4 (TRANSLATION PRESERVES TYPES (PROGRAMS)).

- *If* $\vdash_{\text{prog}} p\ \mathsf{ok}$, *then* $\cdot;\cdot \vdash_{\text{exp}} \mathcal{E}_{\vdash_{\text{prog}}}[\![\vdash_{\text{prog}} p\ \mathsf{ok}]\!] : \mathsf{bool}$.

Translations yielding terms (programs)

$$\mathcal{E}_{\vdash_{\text{prog}}}\left[\!\!\left[\frac{\cdot, \mathcal{H} \succeq \{\}; \cdot \vdash_{\text{exp}} p : \mathsf{bool}, \mathcal{H}}{\vdash_{\text{prog}} p\ \mathsf{ok}}\right]\!\!\right] = \begin{array}{l}\mathsf{runRGN}\ (\Lambda\mathcal{H}.\lambda\mathcal{H}^h : \mathsf{RGNHandle}\ \mathcal{H}.\\ \quad \mathsf{let}\ \mathcal{H}^w = ()\ \mathsf{in}\\ \quad \mathcal{E}_{\vdash_{\text{exp}}}[\![\cdot; \mathcal{H} \succeq \{\}; \cdot \vdash_{\text{exp}} p : \mathsf{bool}, \mathcal{H}]\!])\end{array}$$

Figure 9: Translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$ (Programs)

where τ_r and τ_b are inferred from context.

The translation of an integer constant is a canonical example of allocation in the target calculus. The allocation is accomplished by the $\underline{\mathsf{allocRGNVar}}$ command, applied to the appropriate region handle and value. However, the resulting command has type $\mathsf{RGN}\ \rho\ (\mathsf{RGNVar}\ \rho\ \mathsf{int})$, whereas the source typing judgement requires the computation to be expressed relative to the region ϵ. We coerce the computation using a witness function, whose existence is implied by the judgement $\Delta \vdash_{\text{rr}} \epsilon \succeq \rho$. Allocation of a function proceeds in exactly the same manner. Function application, while notationally heavy, is simple. The $\underline{\mathsf{thenRGN}}$ commands sequence evaluating the function to a location, reading the location, evaluating the argument, and applying the function to the argument.

The translation of $\mathsf{new}\ \varrho.e$ is pleasantly direct. As described above, we introduce ϱ, ϱ^h, and ϱ^w through Λ- and λ-abstractions. The region handle and coercion function are supplied by the $\underline{\mathsf{newRGN}}$ command when the computation is executed.

The translation of region abstraction is similar to the translation of functions. Once again, region handles and witness functions are λ-bound in accordance to the invariants described above. During the translation of region applications, the appropriate tuple of witness functions (constructed by $\mathcal{E}_{\vdash_{\text{re}}}[\![\cdot]\!]$) and region handle are supplied as arguments.

Figure 9 shows the translation of programs. An entire region computation is encapsulated and run by the runRGN expression. We bind $\mathcal{H}^w$ to an empty tuple, which corresponds to the absence of any coercion functions to the region $\mathcal{H}$.

In each figure, we have indicated the particular type preservation lemma implied by each component of the translation. The proofs are by (mutual) induction on the structure of the typing judgements.

5. RELATED WORK

The work in this paper draws heavily from two lines of research. The first is the work done in type-safe region-based memory management, introduced by Tofte and Talpin [24, 25]. Our Single Effect Calculus draws inspiration from the Capability Calculus [5] and Cyclone [8], where the "outlives" relationship between regions is recognized as an important component of type-systems for region calculi.

The work of Banerjee, Heintze and Riecke [2] deserves special mention. They show how to translate the region calculus of Tofte and Talpin into an extension of the polymorphic λ-calculus called $\mathsf{F}_\#$. A new type operator $\#$ is used as a mechanism to hide and reveal the structure of types. Capabilities to allocate and read values from a region are explicitly passed as polymorphic functions of types $\forall\alpha.\alpha \to (\alpha\#\rho)$ and $\forall\alpha.(\alpha\#\rho) \to \alpha$; however, regions have no run-time significance in $\mathsf{F}_\#$ and there is no notion of deallocation upon exiting a region. The equality theory of types in $\mathsf{F}_\#$ is nontrivial, due to the treatment of $\#$; in contrast, type equality on $\mathsf{F}^{\mathsf{RGN}}$ types is purely syntactic. Finally, it is worth noting that there is almost certainly a connection between the $\mathsf{F}_\#$ lift and seq expressions and the monadic return and bind operations, although it is not mentioned or explored in the paper.

The second line of research on which we draw is the work done in monadic encapsulation of effects [17, 18, 21, 14, 26, 15, 16, 22, 1, 13, 23, 19, 27]. The majority of this work has focused on effects arising from reading and writing mutable state. While recent work [26, 19, 27] has considered more general combinations of effects and monads, no work has examined the combination of regions and monads.

Launchbury and Peyton Jones [14, 15] introduced a monadic *state transformer* type $\mathsf{ST}\ s\ \alpha$ for computations which transform a state indexed by s and delivers a value of type α. To run such state transforming computations, they provide a term runST with the type $\forall\alpha.(\forall s.\mathsf{ST}\ s\ \alpha) \to \alpha$. Our typing rules for runRGN and $\underline{\mathsf{newRGN}}$, inspired by that of runST, use the same parametricity to ensure that computations do not leak any (direct or indirect) references to deallocated regions.

Launchbury and Sabry [16] argue that the principle behind runST can be generalized to provide nested scope. They introduce two constants

$$\begin{array}{rcl}\mathsf{blockST} & :: & (\forall\beta.\mathsf{ST}\ (\alpha \times \beta)\ \tau) \to \mathsf{ST}\ \alpha\ \tau\\ \mathsf{importVar} & :: & \mathsf{MutVar}\ \alpha\ \tau \to \mathsf{MutVar}\ (\alpha \times \beta)\ \tau\end{array}$$

where $\mathsf{blockST}$ encapsulates a new scope and $\mathsf{importVar}$ explicitly allows variables from an enclosing scope to be manipulated by the inner scope. Similarly, Peyton Jones[3] suggests introducing the constant

$$\mathsf{liftST}\ ::\ \mathsf{ST}\ \alpha\ \tau \to \mathsf{ST}\ (\alpha \times \beta)\ \tau$$

in lieu of $\mathsf{importVar}$, with the same intention of importing computations from an outer scope into the inner scope. At first glance, this mechanism seems sufficient for supporting a translation from a region calculus. However, in the presence of region polymorphism, such an approach proves difficult. The problem is that the explicit connection between the outer and inner scopes in the product type enforces a total order on regions. This total order is expressed in the

[3]private communication

types of region allocated values. Hence, one cannot write a function polymorphic in the regions ρ_1 and ρ_2 and apply it in all three of the following situations: (a) instantiate ρ_1 and ρ_2 with the same region, (b) instantiate ρ_1 with a region that strictly outlives the region that instantiates ρ_2, (c) vice versa. To put it another way, the function doesn't know what the region stack is going to look like when it is called – it doesn't know where ρ_1 and ρ_2 are relative to each other or to the top of the stack. Hence, we adopt the approach presented in this paper, where we pass evidence showing that each of the regions is live.

Finally, we note that Wadler and Thiemann [27] advocate marrying effects and monads by translating a type $\tau_1 \xrightarrow{\sigma} \tau_2$ to the type $\mathcal{T}\,[\![\tau_1]\!] \to \mathsf{T}^{\sigma}\ \mathcal{T}\,[\![\tau_2]\!]$, where $\mathsf{T}^{\sigma}\ \tau$ represents a computation that yields a value of type τ and has effects delimited by (the set) σ. As with the work of Banerjee et. al. described above, this introduces a nontrivial theory of equality (and subtyping) on types; the types $\mathsf{T}^{\sigma}\ \tau$ and $\mathsf{T}^{\sigma'}\ \tau$ are equal so long as σ and σ' are (encodings of) equivalent sets. However, few programming languages allow one to express such nontrivial equalities between types.

6. CONCLUSIONS AND FUTURE WORK

We have given a type preserving translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$. Both the source and the target calculi use a static type-system to delimit the effects of allocating in and reading from regions. The Single Effect Calculus uses the partial order implied by the "outlives" relation on regions to use single regions as bounds for sets of effects. We feel that this is an important insight that leads to a relatively straight-forward translation into the monadic setting. $\mathsf{F}^{\mathsf{RGN}}$ draws from the work on monadic encapsulation of state to give parametric types to runRGN and $\underline{\mathsf{newRGN}}$ that prevent access of regions beyond their lifetimes. Explicit functions witness the outlives relationship between regions, enabling computations from outer regions to be cast to computations in inner regions. Witness functions cannot be forged and are only introduced via $\underline{\mathsf{newRGN}}$.

There are numerous directions for future work. Finalizing semantics for both the Single Effect Calculus and $\mathsf{F}^{\mathsf{RGN}}$, proving the type safety of each language, and proving that the translation preserves the semantics of programs are foremost on our list. Exploring an optimized monad translation is likely to be important here. We are also interested in mechanisms that relax the notational burden of passing witness functions, such as type-classes. Finally, as is well known, Tofte and Talpin's original region calculus can lead to inefficient memory usage in tail-recursive programs. We would like to explore whether or not techniques developed to overcome these problems (e.g., storage analysis and late allocate/early deallocation analysis) can be adapted to the monadic setting.

Acknowledgements

Greg Morrisett proposed an initial encoding of regions into a monadic setting based on the ST monad and discussed the many refinements leading to the present work. The anonymous referees made a number of helpful suggestions.

7. REFERENCES

[1] Z. Ariola and A. Sabry. Correctness of monadic state: An imperative call-by-need calculus. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98)*, pages 62–74, San Diego, CA, 1998. ACM Press.

[2] A. Banerjee, N. Heintze, and J. Riecke. Region analysis and the polymorphic lambda calculus. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science (LCS'99)*, pages 88–97, Trento, Italy, 1999. IEEE Computer Society Press.

[3] C. Calcagno. Stratified operational semantics for safety and correctness of the region calculus. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 155–165, London, England, 2001. ACM Press.

[4] C. Calcagno, S. Helsen, and P. Thiemann. Syntactic type soundness results for the region calculus. *Information and Computation*, 173(2):199–332, Mar. 2002.

[5] K. Crary, D. Walker, and G. Morrisett. Typed memory management in a calculus of capabilities. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'99)*, pages 262–275. ACM Press, 1999.

[6] M. Elsman. Garbage collection safety for region-based memory management. In *Proceedings of the ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI'03)*, pages 123–134, New Orleans, LA, 2003. ACM Press.

[7] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types.* Cambridge University Press, 1989.

[8] D. Grossman, G. Morrisett, Y. Wang, T. Jim, M. Hicks, and J. Cheney. Formal type soundness for Cyclone's region system. Technical Report 2001-1856, Department of Computer Science, Cornell University, Nov. 2001.

[9] N. Hallenberg, M. Elsman, and M. Tofte. Combining region inference and garbage collection. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'02)*, pages 141–152. ACM Press, 2002. Berlin, Germany.

[10] S. Helsen and P. Thiemann. Syntactic type soundness for the region calculus. In *Proceedings of the 4th International Workshop on Higher Order Operational Techniques in Semantics (HOOTS'00)*, volume 41 of *Electronic Notes in Theoretical Computer Science*, pages 1–19, Montreal, Canada, Sept. 2000. Elsevier Science Publishers.

[11] F. Henglein, H. Makholm, and H. Niss. Effect types and region-based memory management. In B. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 5. MIT Press, 2003. In preparation.

[12] M. Hicks, G. Morrisett, D. Grossman, and T. Jim. Safe and flexible memory management in Cyclone. Technical Report CS-TR-4514, University of Maryland Department of Computer Science, July 2003.

[13] R. Kieburtz. Taming effects with monadic typing. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*, pages 51–62, Baltimore, MD, 1998. ACM Press.

[14] J. Launchbury and S. Peyton Jones. State in Haskell. *Lisp and Symbolic Computation*, 8(4):293–341, 1995.

[15] J. Launchbury and S. Peyton Jones. Lazy functional state threads. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'94)*, pages 24–35, Orlando, FL, 1997. ACM Press.

[16] J. Launchbury and A. Sabry. Monadic state: Axiomatization and type safety. In *Proceedings of the 2nd ACM SIGPLAN International Conference on Functional Programming (ICFP'97)*, pages 227–237, Amsterdam, The Netherlands, 1997. ACM Press.

[17] E. Moggi. Computational lambda calculus and monads. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science (LCS'89)*, pages 14–23, Pacific Grove, CA, 1989.

[18] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, Jan. 1991.

[19] E. Moggi and A. Sabry. Monadic encapsulation of effects: a revised approach (extended version). *Journal of Functional Programming*, 11(6):591–627, Nov. 2001.

[20] J. Reynolds. Towards a theory of type structure. In *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Paris, France, Apr. 1974. Springer-Verlag.

[21] J. Riecke and R. Viswanathan. Isolating side effects in sequential languages. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95)*, pages 1–12, San Francisco, CA, 1995. ACM Press.

[22] A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, 19(6):916–941, Nov. 1997.

[23] M. Semmelroth and A. Sabry. Monadic encapsulation in ml. In *Proceedings of the 4th ACM SIGPLAN International Conference on Functional Programming (ICFP'99)*, pages 8–17, Paris, France, 1999. ACM Press.

[24] M. Tofte and J.-P. Talpin. Implementation of the typed call-by-value λ-calculus using a stack of regions. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94)*, pages 188–201, Portland, OR, 1994. ACM Press.

[25] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, Feb. 1997.

[26] P. Wadler. The marriage of effects and monads. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*, pages 63–74, Baltimore, MD, 1995. ACM Press.

[27] P. Wadler and P. Thiemann. The marriage of effects and monads. *Transactions on Computational Logic*, 4(1):1–32, 2003.

APPENDIX

Figures 10 and 11 present simple large-step operational semantics for the Single Effect Calculus and $\mathsf{F}^{\mathsf{RGN}}$ respectively. Figures 12 and 13 show the translation of terms not covered in Figure 8.

We use the following notational conventions in Figure 10:

- $\bullet$ denotes a deallocated region.
- $\langle l \rangle_\rho$ denotes a pointer to a storable in region ρ.
- Regions and region stacks that differ only in the order of their fields are considered identical.
- The expression $e[e'/x]$ denotes the capture-avoiding substitution of e' for x in e. Similar capture-avoiding substitutions are defined for $e[\rho/\varrho]$ and $w[\rho/\varrho]$.
- Updates of finite maps R are denoted by $M\{l \mapsto w\}$ and updates of finite maps S are denoted by $S\{\varrho \mapsto R\}$. We abbreviate $S\{\varrho \mapsto S(\varrho)\{l \mapsto w\}\}$ by $S\{(\varrho, l) \mapsto w\}$.
- The notation $S - \varrho$ excludes ϱ from the domain of S.
- The notation $S[\rho/\varrho]$ denotes

$$\{\varrho' \mapsto \{l' \mapsto S(\varrho', l')[\rho/\varrho]\}_{l' \in dom(S(\varrho'))}\}_{\varrho' \in dom(S)}.$$

We use the following notational conventions in Figure 11:

- $\bullet$ denotes a deallocated region.
- $\langle l \rangle_\rho$ denotes a pointer to a storable in region ρ.
- Regions and region stacks that differ only in the order of their fields are considered identical.
- The expression $e[e'/x]$ denotes the capture-avoiding substitution of e' for x in e. Similar capture-avoiding substitutions are defined for $e[\rho/r]$, $e[\tau'/\alpha]$, $w[\rho/r]$, $w[\tau'/\alpha]$, and $\tau[\tau'/\alpha]$.
- Updates of finite maps R are denoted by $M\{l \mapsto w\}$ and updates of finite maps S are denoted by $S\{r \mapsto R\}$. We abbreviate $S\{r \mapsto S(r)\{l \mapsto w\}\}$ by $S\{(r, l) \mapsto w\}$.
- The notation $S - r$ excludes r from the domain of S.
- The notation $S[\rho/r]$ denotes

$$\{r' \mapsto \{l' \mapsto S(r', l')[\rho/r]\}_{l' \in dom(S(r'))}\}_{r' \in dom(S)}.$$

$$l \quad \in \quad Locs^{SEC}$$

Places

$$\epsilon, \rho \quad ::= \quad \ldots \mid \bullet$$

Terms

$$e \quad ::= \quad \ldots \mid \langle l \rangle_{\rho}$$

Values

$$v \quad ::= \quad \langle l \rangle_{\rho}$$

Storables

$$w \quad ::= \quad i \mid \lambda x : \tau.^{\epsilon} e \mid (v_1, v_2) \mid \lambda \varrho \succeq \varphi : \tau.^{\epsilon} u$$

Regions

$$R \quad ::= \quad \{l_1 \mapsto w_1, \ldots, l_n \mapsto w_n\}$$

Region stacks

$$S \quad ::= \quad \{\varrho_1 \mapsto R_1, \ldots, \varrho_m \mapsto R_m\}$$

$$\boxed{S, e \hookrightarrow S', v}$$

$$\frac{l \notin dom(S(\varrho))}{S, i \ \mathsf{at}\ \varrho \hookrightarrow S\{(\varrho, l) \mapsto i\}, \langle l \rangle_{\varrho}}
\qquad
\frac{\begin{array}{cc} S, e_1 \hookrightarrow S_1, \langle l_1 \rangle_{\varrho_1} & S_1(\varrho_1, l_1) = i_1 \\ S_1, e_2 \hookrightarrow S_2, \langle l_2 \rangle_{\varrho_2} & S_2(\varrho_2, l_2) = i_2 \\ \multicolumn{2}{c}{l \notin dom(S_2(\varrho))} \end{array}}{S, e_1 \oplus e_2 \ \mathsf{at}\ \varrho \hookrightarrow S_2\{(\varrho, l) \mapsto i_1 \oplus i_2\}, \langle l \rangle_{\varrho}}
\qquad
\frac{\begin{array}{cc} S, e_1 \hookrightarrow S_1, \langle l_1 \rangle_{\varrho_1} & S_1(\varrho_1, l_1) = i_1 \\ S_1, e_2 \hookrightarrow S_2, \langle l_2 \rangle_{\varrho_2} & S_2(\varrho_2, l_2) = i_2 \end{array}}{S, e_1 \otimes e_2 \ \mathsf{at}\ \varrho \hookrightarrow S_2, i_1 \otimes i_2}$$

$$\frac{}{S, \mathsf{tt} \hookrightarrow S, \mathsf{tt}}
\qquad
\frac{}{S, \mathsf{ff} \hookrightarrow S, \mathsf{ff}}
\qquad
\frac{\begin{array}{c} S, e_b \hookrightarrow S, \mathsf{tt} \\ S', e_t \hookrightarrow S'', v'' \end{array}}{S, \mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f \hookrightarrow S'', v''}
\qquad
\frac{\begin{array}{c} S, e_b \hookrightarrow S, \mathsf{ff} \\ S', e_f \hookrightarrow S'', v'' \end{array}}{S, \mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f \hookrightarrow S'', v''}$$

$$\frac{l \notin dom(S(\varrho))}{S, \lambda x : \tau.^{\epsilon} e \ \mathsf{at}\ \varrho \hookrightarrow S\{(\varrho, l) \mapsto \lambda x : \tau.^{\epsilon} e\}, \langle l \rangle_{\varrho}}
\qquad
\frac{\begin{array}{c} S, e_1 \hookrightarrow S_1, \langle l_1 \rangle_{\varrho_1} \\ S_1(\varrho_1, l_1) = \lambda x : \tau.^{\epsilon} e \\ S_1, e_2 \hookrightarrow S_2, v_2 \\ S_2, e[v_2/x] \hookrightarrow S', v' \end{array}}{S, e_1 \ e_2 \hookrightarrow S', v'}
\qquad
\frac{\begin{array}{cc} S, e_1 \hookrightarrow S_1, v_1 & S_1, e_2 \hookrightarrow S_2, v_2 \\ \multicolumn{2}{c}{l \notin dom(S_2(\varrho))} \end{array}}{S, (e_1, e_2) \ \mathsf{at}\ \varrho \hookrightarrow S_2\{(\varrho, l) \mapsto (v_1, v_2)\}, \langle l \rangle_{\varrho}}$$

$$\frac{\begin{array}{c} S, e \hookrightarrow S', \langle l \rangle_{\varrho} \\ S'(\varrho, l) = (v_1, v_2) \end{array}}{S, \mathsf{fst}\ e \hookrightarrow S', v_1}
\qquad
\frac{\begin{array}{c} S, e \hookrightarrow S', \langle l \rangle_{\varrho} \\ S'(\varrho, l) = (v_1, v_2) \end{array}}{S, \mathsf{snd}\ e \hookrightarrow S', v_2}
\qquad
\frac{\varrho \notin dom(S) \qquad S\{\varrho \mapsto \emptyset\}, e \hookrightarrow S', v}{S, \mathsf{new}\ \varrho.e \hookrightarrow (S' - \varrho)[\bullet/\varrho], v[\bullet/\varrho]}$$

$$\frac{l \notin dom(S(\varrho))}{S, \lambda \varrho' \succeq \varphi.^{\epsilon} u \ \mathsf{at}\ \varrho \hookrightarrow S\{(\varrho, l) \mapsto \lambda \varrho' \succeq \varphi.^{\epsilon} u\}, \langle l \rangle_{\varrho}}
\qquad
\frac{\begin{array}{c} S, e \hookrightarrow S', \langle l \rangle_{\varrho} \\ S(\varrho, l) = \lambda \varrho' \succeq \varphi.^{\epsilon} u \\ S', u[\rho/\varrho'] \hookrightarrow S'', v'' \end{array}}{S, e\ [\rho] \hookrightarrow S'', v''}$$

$$\frac{l \notin dom(S(\varrho))}{S, \mathsf{fix}\ f : (\tau_1 \xrightarrow{\epsilon} \tau_2, \varrho).\lambda x : \tau_1.^{\epsilon} e \ \mathsf{at}\ \varrho \hookrightarrow S\{(\varrho, l) \mapsto \lambda x : \tau_1.^{\epsilon} e[\langle l \rangle_{\varrho}/f]\}, \langle l \rangle_{\varrho}}$$

$$\frac{l \notin dom(S(\varrho))}{S, \mathsf{fix}\ f : (\Pi \varrho' \succeq \varphi.^{\epsilon} \tau, \varrho).\lambda \varrho' \succeq \varphi.^{\epsilon} u \ \mathsf{at}\ \varrho \hookrightarrow S\{(\varrho, l) \mapsto \lambda \varrho' \succeq \varphi.^{\epsilon} u[\langle l \rangle_{\varrho}/f]\}, \langle l \rangle_{\varrho}}$$

$$\boxed{p \hookrightarrow v}$$

$$\frac{\{\mathcal{H} \mapsto \emptyset\}, p \hookrightarrow S, v}{p \hookrightarrow v}$$

Figure 10: Single Effect Calculus: Operational Semantics

$$
\begin{array}{lcl}
l & \in & \mathit{Locs}^{FRGN} \\
r & \in & (\mathit{Rname}, \sqsubseteq)
\end{array}
$$

Types

$\tau \quad ::= \quad \ldots \mid \rho$

Places

$\rho \quad ::= \quad r \mid \bullet$

Terms

$e \quad ::= \quad \ldots \mid \mathsf{handle}(\rho) \mid \langle l \rangle_\rho$

Commands

$\kappa \quad ::= \quad \ldots \mid \underline{\mathsf{witnessRGN}}\ [\tau_a]\ [\rho_1]\ [\rho_2]\ v$

Values

$v \quad ::= \quad \ldots \mid \mathsf{handle}(\rho) \mid \langle l \rangle_\rho$

Storables

$w \quad ::= \quad v$

Regions

$R \quad ::= \quad \{l_n \mapsto w_n, \ldots, l_n \mapsto w_n\}$

Region stacks

$S \quad ::= \quad \{r_m \mapsto R_m, \ldots, r_m \mapsto R_m\}$

$\boxed{e \hookrightarrow v}$

$$\frac{}{i \hookrightarrow i} \qquad \frac{e_1 \hookrightarrow i_1 \qquad e_2 \hookrightarrow i_2}{e_1 \oplus e_2 \hookrightarrow i_1 \oplus i_2} \qquad \frac{e_1 \hookrightarrow i_1 \qquad e_2 \hookrightarrow i_2}{e_1 \otimes e_2 \hookrightarrow i_1 \otimes i_2} \qquad \frac{}{\mathsf{tt} \hookrightarrow \mathsf{tt}} \qquad \frac{}{\mathsf{ff} \hookrightarrow \mathsf{ff}} \qquad \frac{e_b \hookrightarrow \mathsf{tt} \qquad e_t \hookrightarrow v}{\mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f \hookrightarrow v}$$

$$\frac{e_b \hookrightarrow \mathsf{ff} \qquad e_f \hookrightarrow v}{\mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f \hookrightarrow v} \qquad \frac{}{\lambda x : \tau.e \hookrightarrow \lambda x : \tau.e} \qquad \frac{e_1 \hookrightarrow \lambda x : \tau.e' \qquad e_2 \hookrightarrow v \qquad e'[v/x] \hookrightarrow v'}{e_1\ e_2 \hookrightarrow v'} \qquad \frac{}{\Lambda\alpha.e \hookrightarrow \Lambda\alpha.e}$$

$$\frac{e \hookrightarrow \Lambda\alpha.e' \qquad e'[\tau/\alpha] \hookrightarrow v}{e\ [\tau] \hookrightarrow v} \qquad \frac{e_1 \hookrightarrow v_1 \qquad e_2 \hookrightarrow v_2}{(e_1, e_2) \hookrightarrow (v_1, v_2)} \qquad \frac{e \hookrightarrow (v_1, v_2)}{\mathsf{fst}\ e \hookrightarrow v_1} \qquad \frac{e \hookrightarrow (v_1, v_2)}{\mathsf{snd}\ e \hookrightarrow v_2}$$

$$\frac{v'\ [r]\ \mathsf{handle}(r) \hookrightarrow \kappa \qquad \{r \mapsto \emptyset\}, \kappa \hookrightarrow S, v'}{\mathsf{runRGN}\ [\tau_a]\ v \hookrightarrow v'[\bullet/r]} \qquad \frac{}{\kappa \hookrightarrow \kappa} \qquad \frac{}{\rho \hookrightarrow \rho} \qquad \frac{}{\langle l \rangle_\rho \hookrightarrow \langle l \rangle_\rho}$$

$\boxed{S, \kappa \hookrightarrow S', v}$

$$\frac{}{S, \underline{\mathsf{returnRGN}}\ [r]\ [\tau_a]\ v \hookrightarrow S, v} \qquad \frac{S, \kappa \hookrightarrow S', v' \qquad v\ v' \hookrightarrow \kappa' \qquad S', \kappa' \hookrightarrow S'', v''}{S, \underline{\mathsf{thenRGN}}\ [r]\ [\tau_a]\ [\tau_b]\ \kappa\ v \hookrightarrow S'', v''}$$

$$\frac{r \in dom(S) \qquad l \notin dom(S(r))}{S, \underline{\mathsf{allocRGNVar}}\ [\tau_a]\ [r]\ \mathsf{handle}(r)\ w \hookrightarrow S\{(r,l) \mapsto w\}, \langle l \rangle_r} \qquad \frac{r \in dom(S) \qquad l \in dom(S(r))}{S, \underline{\mathsf{readRGNVar}}\ [\tau_a]\ [r]\ \langle l \rangle_r \hookrightarrow S, S(r,l)}$$

$$\frac{r \in dom(S) \qquad l \notin dom(S(r)) \qquad v\ \langle l \rangle_r \hookrightarrow w}{S, \underline{\mathsf{fixRGNVar}}\ [\tau_a]\ [r]\ r\ v \hookrightarrow S\{(r,l) \mapsto w\}, \langle l \rangle_r}$$

$$\frac{\begin{array}{c} r \sqsubseteq r' \qquad r' \notin dom(S) \\ v\ (\Lambda\alpha.\lambda c : \mathsf{RGN}\ r\ \alpha.\underline{\mathsf{witnessRGN}}\ [\alpha]\ [r]\ [r']\ c)\ \mathsf{handle}(r') \hookrightarrow \kappa \qquad S\{r' \mapsto \emptyset\}, \kappa \hookrightarrow S'[r' \mapsto R], v' \end{array}}{S, \underline{\mathsf{newRGN}}\ [r]\ [\tau_a]\ v \hookrightarrow S'[\bullet/r'], v'[\bullet/r']}$$

$$\frac{r_1 \sqsubseteq r_2 \qquad S, \kappa \hookrightarrow S', v}{S, \underline{\mathsf{witnessRGN}}\ [\tau_a]\ [r_1]\ [r_2]\ \kappa \hookrightarrow S', v}$$

Figure 11: $\mathsf{F}^{\mathsf{RGN}}$: Operational Semantics

Translations yielding terms (terms)

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left[\!\!\left[\dfrac{\begin{array}{cc} \Delta;\Gamma \vdash_{\mathrm{exp}} e_1 : (\mathsf{int},\rho_1),\epsilon & \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_1 \\ \Delta;\Gamma \vdash_{\mathrm{exp}} e_2 : (\mathsf{int},\rho_2),\epsilon & \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_2 \\ \Delta \vdash_{\mathrm{place}} \rho & \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} e_1 \oplus e_2 \ \mathsf{at}\ \rho : (\mathsf{int},\rho),\epsilon} \right]\!\!\right] =$$

$$\begin{array}{l}
\mathsf{bind}\ a : \mathcal{T}_\tau [\![(\mathsf{int},\rho_1)]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} [\![\Delta;\Gamma \vdash_{\mathrm{exp}} e_1 : (\mathsf{int},\rho_1),\epsilon]\!]\,; \\
\mathsf{bind}\ a' : \mathcal{T}_\mu [\![\mathsf{int}]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{rr}}} [\![\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_1]\!]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ (\underline{\mathsf{readRGNVar}}\ [\rho_1]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ a); \\
\mathsf{bind}\ b : \mathcal{T}_\tau [\![(\mathsf{int},\rho_2)]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} [\![\Delta;\Gamma \vdash_{\mathrm{exp}} e_2 : (\mathsf{int},\rho_2),\epsilon]\!]\,; \\
\mathsf{bind}\ b' : \mathcal{T}_\mu [\![\mathsf{int}]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{rr}}} [\![\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_2]\!]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ (\underline{\mathsf{readRGNVar}}\ [\rho_2]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ b); \\
\mathsf{let}\ i = a' \oplus b'\ \mathsf{in} \\
\mathcal{E}_{\vdash_{\mathrm{rr}}} [\![\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho]\!]\ [\mathcal{T}_\tau [\![(\mathsf{int},\rho)]\!]]\ (\underline{\mathsf{allocRGNVar}}\ [\rho]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ \rho^h\ i) \qquad \text{where } a, a', b, b', i \text{ fresh}
\end{array}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left[\!\!\left[\dfrac{\begin{array}{cc} \Delta;\Gamma \vdash_{\mathrm{exp}} e_1 : (\mathsf{int},\rho_1),\epsilon & \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_1 \\ \Delta;\Gamma \vdash_{\mathrm{exp}} e_2 : (\mathsf{int},\rho_2),\epsilon & \Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_1 \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} e_1 \circledless e_2 : \mathsf{bool},\epsilon} \right]\!\!\right] =$$

$$\begin{array}{l}
\mathsf{bind}\ a : \mathcal{T}_\tau [\![(\mathsf{int},\rho_1)]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} [\![\Delta;\Gamma \vdash_{\mathrm{exp}} e_1 : (\mathsf{int},\rho_1),\epsilon]\!]\,; \\
\mathsf{bind}\ a' : \mathcal{T}_\mu [\![\mathsf{int}]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{rr}}} [\![\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_1]\!]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ (\underline{\mathsf{readRGNVar}}\ [\rho_1]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ a); \\
\mathsf{bind}\ b : \mathcal{T}_\tau [\![(\mathsf{int},\rho_2)]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} [\![\Delta;\Gamma \vdash_{\mathrm{exp}} e_2 : (\mathsf{int},\rho_2),\epsilon]\!]\,; \\
\mathsf{bind}\ b' : \mathcal{T}_\mu [\![\mathsf{int}]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{rr}}} [\![\Delta \vdash_{\mathrm{rr}} \epsilon \succeq \rho_2]\!]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ (\underline{\mathsf{readRGNVar}}\ [\rho_2]\ [\mathcal{T}_\mu [\![\mathsf{int}]\!]]\ b); \\
\mathsf{let}\ i = a' \oplus b'\ \mathsf{in} \\
\underline{\mathsf{returnRGN}}\ [\epsilon]\ [\mathcal{T}_\tau [\![\mathsf{bool}]\!]]\ z \qquad \text{where } a, a', b, b', i \text{ fresh}
\end{array}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left[\!\!\left[\dfrac{}{\Delta;\Gamma \vdash_{\mathrm{exp}} \mathsf{tt} : \mathsf{bool},\epsilon} \right]\!\!\right] = \underline{\mathsf{returnRGN}}\ [\epsilon]\ [\mathcal{T}_\tau [\![\mathsf{bool}]\!]]\ \mathsf{tt}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left[\!\!\left[\dfrac{}{\Delta;\Gamma \vdash_{\mathrm{exp}} \mathsf{ff} : \mathsf{bool},\epsilon} \right]\!\!\right] = \underline{\mathsf{returnRGN}}\ [\epsilon]\ [\mathcal{T}_\tau [\![\mathsf{bool}]\!]]\ \mathsf{ff}$$

$$\mathcal{E}_{\vdash_{\mathrm{exp}}} \left[\!\!\left[\dfrac{\begin{array}{c} \Delta;\Gamma \vdash_{\mathrm{exp}} e_b : \mathsf{bool},\epsilon \\ \Delta;\Gamma \vdash_{\mathrm{exp}} e_t : \tau,\epsilon \qquad \Delta;\Gamma \vdash_{\mathrm{exp}} e_f : \tau,\epsilon \end{array}}{\Delta;\Gamma \vdash_{\mathrm{exp}} \mathsf{if}\ e_b\ \mathsf{then}\ e_t\ \mathsf{else}\ e_f : \tau,\epsilon} \right]\!\!\right] =$$

$$\begin{array}{l}
\mathsf{bind}\ b : \mathcal{T}_\tau [\![\mathsf{bool}]\!] \Leftarrow \mathcal{E}_{\vdash_{\mathrm{exp}}} [\![\Delta;\Gamma \vdash_{\mathrm{exp}} e_b : \mathsf{bool},\epsilon]\!]\,; \\
\mathsf{if}\ b\ \mathsf{then}\ \mathcal{E}_{\vdash_{\mathrm{exp}}} [\![\Delta;\Gamma \vdash_{\mathrm{exp}} e_t : \tau,\epsilon]\!]\ \mathsf{else}\ \mathcal{E}_{\vdash_{\mathrm{exp}}} [\![\Delta;\Gamma \vdash_{\mathrm{exp}} e_f : \tau,\epsilon]\!] \qquad \text{where } b \text{ fresh}
\end{array}$$

Figure 12: Translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$ (Terms (II))

Translations yielding terms (terms)

$$\mathcal{E}_{\vdash_{\text{exp}}} \left\llbracket \frac{\begin{array}{c}\Delta;\Gamma \vdash_{\text{exp}} e_1 : \tau_1, \epsilon \\ \Delta;\Gamma \vdash_{\text{exp}} e_2 : \tau_2, \epsilon \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho\end{array}}{\Delta;\Gamma \vdash_{\text{exp}} (e_1, e_2) \ \mathsf{at}\ \rho : (\tau_1 \times \tau_2, \rho), \epsilon} \right\rrbracket =$$

$$\begin{array}{l}\mathsf{bind}\ a : \mathcal{T}_\tau \llbracket \tau_1 \rrbracket \Leftarrow \mathcal{E}_{\vdash_{\text{exp}}} \llbracket \Delta;\Gamma \vdash_{\text{exp}} e_1 : \tau_1, \epsilon \rrbracket ; \\ \mathsf{bind}\ b : \mathcal{T}_\tau \llbracket \tau_2 \rrbracket \Leftarrow \mathcal{E}_{\vdash_{\text{exp}}} \llbracket \Delta;\Gamma \vdash_{\text{exp}} e_2 : \tau_2, \epsilon \rrbracket ; \\ \mathcal{E}_{\vdash_{\text{rr}}} \llbracket \Delta \vdash_{\text{rr}} \epsilon \succeq \rho \rrbracket \ [\mathcal{T}_\tau \llbracket (\tau_1 \times \tau_2, \rho) \rrbracket] \ (\underline{\mathsf{allocRGNVar}}\ [\rho]\ [\mathcal{T}_\tau \llbracket (\tau_1 \times \tau_2, \rho) \rrbracket]\ \rho^h\ (a, b)) \qquad \text{where } a, b \text{ fresh}\end{array}$$

$$\mathcal{E}_{\vdash_{\text{exp}}} \left\llbracket \frac{\Delta;\Gamma \vdash_{\text{exp}} e : (\tau_1 \times \tau_2, \rho), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho}{\Delta;\Gamma \vdash_{\text{exp}} \mathsf{fst}\ \ e : \tau_1, \epsilon} \right\rrbracket =$$

$$\begin{array}{l}\mathsf{bind}\ x : \mathcal{T}_\tau \llbracket (\tau_1 \times \tau_2, \rho) \rrbracket \Leftarrow \mathcal{E}_{\vdash_{\text{exp}}} \llbracket \Delta;\Gamma \vdash_{\text{exp}} e : (\tau_1 \times \tau_2, \rho), \epsilon \rrbracket ; \\ \mathsf{bind}\ y : \mathcal{T}_\mu \llbracket \tau_1 \times \tau_2 \rrbracket \Leftarrow \mathcal{E}_{\vdash_{\text{rr}}} \llbracket \Delta \vdash_{\text{rr}} \epsilon \succeq \rho \rrbracket \ [\mathcal{T}_\mu \llbracket \tau_1 \times \tau_2 \rrbracket] \ (\underline{\mathsf{readRGNVar}}\ [\rho]\ [\mathcal{T}_\mu \llbracket \tau_1 \times \tau_2 \rrbracket]\ a); \\ \mathsf{let}\ z = \mathsf{sel}_1\ y\ \mathsf{in} \\ \underline{\mathsf{returnRGN}}\ [\epsilon]\ [\mathcal{T}_\tau \llbracket \tau_1 \rrbracket]\ z \qquad \text{where } x, y, z \text{ fresh}\end{array}$$

$$\mathcal{E}_{\vdash_{\text{exp}}} \left\llbracket \frac{\Delta;\Gamma \vdash_{\text{exp}} e : (\tau_1 \times \tau_2, \rho), \epsilon \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho}{\Delta;\Gamma \vdash_{\text{exp}} \mathsf{snd}\ \ e : \tau_2, \epsilon} \right\rrbracket =$$

$$\begin{array}{l}\mathsf{bind}\ x : \mathcal{T}_\tau \llbracket (\tau_1 \times \tau_2, \rho) \rrbracket \Leftarrow \mathcal{E}_{\vdash_{\text{exp}}} \llbracket \Delta;\Gamma \vdash_{\text{exp}} e : (\tau_1 \times \tau_2, \rho), \epsilon \rrbracket ; \\ \mathsf{bind}\ y : \mathcal{T}_\mu \llbracket \tau_1 \times \tau_2 \rrbracket \Leftarrow \mathcal{E}_{\vdash_{\text{rr}}} \llbracket \Delta \vdash_{\text{rr}} \epsilon \succeq \rho \rrbracket \ [\mathcal{T}_\mu \llbracket \tau_1 \times \tau_2 \rrbracket] \ (\underline{\mathsf{readRGNVar}}\ [\rho]\ [\mathcal{T}_\mu \llbracket \tau_1 \times \tau_2 \rrbracket]\ a); \\ \mathsf{let}\ z = \mathsf{sel}_2\ y\ \mathsf{in} \\ \underline{\mathsf{returnRGN}}\ [\epsilon]\ [\mathcal{T}_\tau \llbracket \tau_2 \rrbracket]\ z \qquad \text{where } x, y, z \text{ fresh}\end{array}$$

$$\mathcal{E}_{\vdash_{\text{exp}}} \left\llbracket \frac{\begin{array}{l}f \notin \mathit{dom}(\Gamma) \\ \Delta \vdash_{\text{type}} (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho)\end{array} \qquad \dfrac{\begin{array}{c}x \notin \mathit{dom}(\Gamma, f : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho)) \qquad \Delta \vdash_{\text{type}} \tau_1 \\ \Delta;\Gamma, f : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), x : \tau_1 \vdash_{\text{exp}} e : \tau_2, \epsilon' \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho\end{array}}{\Delta;\Gamma, f : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho) \vdash_{\text{exp}} \lambda x : \tau_1.^{\epsilon'} e\ \mathsf{at}\ \rho : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), \epsilon}}{\Delta;\Gamma \vdash_{\text{exp}} \mathsf{fix}\ f : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho).\lambda x : \tau_1.^{\epsilon'} e\ \mathsf{at}\ \rho : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), \epsilon} \right\rrbracket =$$

$$\begin{array}{l}\mathcal{E}_{\vdash_{\text{rr}}} \llbracket \Delta \vdash_{\text{rr}} \epsilon \succeq \rho \rrbracket \ \mathcal{T}_\tau \left\llbracket (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho) \right\rrbracket \\ \qquad (\underline{\mathsf{fixRGNVar}}\ [\rho]\ [\mathcal{T}_\mu \left\llbracket \tau_1 \xrightarrow{\epsilon'} \tau_2 \right\rrbracket] \\ \qquad\qquad \rho^h\ (\lambda f : \mathcal{T}_\tau \left\llbracket (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho) \right\rrbracket .\lambda x : \mathcal{T}_\tau \llbracket \tau_1 \rrbracket . \\ \qquad\qquad\quad \mathcal{E}_{\vdash_{\text{exp}}} \left\llbracket \Delta;\Gamma, f : (\tau_1 \xrightarrow{\epsilon'} \tau_2, \rho), x : \tau_1 \vdash_{\text{exp}} e : \tau_2, \epsilon' \right\rrbracket))\end{array}$$

$$\mathcal{E}_{\vdash_{\text{exp}}} \left\llbracket \frac{\begin{array}{l}f \notin \mathit{dom}(\Gamma) \\ \Delta \vdash_{\text{type}} (\Pi\varrho \succeq \varphi.^{\epsilon'} \tau, \rho)\end{array} \qquad \dfrac{\begin{array}{c}\varrho \notin \mathit{dom}(\Delta) \qquad \Delta \vdash_{\text{eff}} \varphi \\ \Delta, \varrho \succeq \varphi;\Gamma, f : (\Pi\varrho \succeq \varphi.^{\epsilon'} \tau, \rho) \vdash_{\text{exp}} u : \tau, \epsilon' \\ \Delta \vdash_{\text{place}} \rho \qquad \Delta \vdash_{\text{rr}} \epsilon \succeq \rho\end{array}}{\Delta;\Gamma, f : (\Pi\varrho \succeq \varphi.^{\epsilon'} \tau, \rho) \vdash_{\text{exp}} \lambda\varrho \succeq \varphi.^{\epsilon'} u\ \mathsf{at}\ \rho : (\Pi\varrho \succeq \varphi.^{\epsilon'} \tau, \rho), \epsilon}}{\Delta;\Gamma \vdash_{\text{exp}} \mathsf{fix}\ f : (\Pi\varrho \succeq \varphi.^{\epsilon'} \tau, \rho).\lambda\varrho \succeq \varphi.^{\epsilon'} u\ \mathsf{at}\ \rho : (\Pi\varrho \succeq \varphi.^{\epsilon'} \tau, \rho), \epsilon} \right\rrbracket =$$

$$\begin{array}{l}\mathcal{E}_{\vdash_{\text{rr}}} \llbracket \Delta \vdash_{\text{rr}} \epsilon \succeq \rho \rrbracket \ \mathcal{T}_\tau \left\llbracket (\Pi\varrho.^{\epsilon'}.\tau, \rho) \right\rrbracket \\ \qquad (\underline{\mathsf{fixRGNVar}}\ [\rho]\ [\mathcal{T}_\mu \left\llbracket \Pi\varrho.^{\epsilon'} \tau \right\rrbracket] \\ \qquad\qquad \rho^h\ (\lambda f : \mathcal{T}_\tau \left\llbracket (\Pi\varrho.^{\epsilon'} \tau, \rho) \right\rrbracket .\lambda\varrho^w : \mathcal{T}_\succeq \llbracket \varrho \succeq \varphi \rrbracket .\lambda\varrho^h : \mathsf{RGNHandle}\ \varrho. \\ \qquad\qquad\quad \mathcal{E}_{\vdash_{\text{exp}}} \left\llbracket \Delta, \varrho \succeq \varphi;\Gamma, f : (\Pi\varrho.^{\epsilon'} \tau, \rho) \vdash_{\text{exp}} u : \tau, \epsilon' \right\rrbracket))\end{array}$$

Figure 13: Translation from the Single Effect Calculus to $\mathsf{F}^{\mathsf{RGN}}$ (Terms (III))