An Equational Theory for a Region Calculus

Simon Helsen

University of Freiburg*, Germany

Abstract. A region calculus is a polymorphically typed lambda calculus with explicit memory management primitives. Every value is annotated with a region in which it is stored. Regions are allocated and deallocated in a stack-like fashion. The annotations can be statically inferred by a type and effect system, making a region calculus suitable as an intermediate language for a compiler of statically typed programming languages. Although a lot of attention has been paid to type soundness properties of different flavors of region calculi, it seems that little effort has been made to develop a semantic framework. In this paper, we formulate a syntactic equational theory, which is proven sound with respect to contextual equivalence. Soundness is achieved by introducing an appropriate notion of bisimilarity, which provides a co-inductive proof principle for showing equivalences in our region calculus. We show that bisimilarity includes the equational theory and that it coincides with contextual equivalence.

Key Words: regions, equational theory, bisimulation, operational equivalence

1 Introduction

Some programming languages, like C, C++ and Pascal, leave dynamic memory management entirely to the programmer. In such a setting, memory deallocation is a notorious source for bugs: objects may be deallocated too late, causing memory leaks, or when deallocation happens too early, the program may crash or produces wrong results. Trace-based garbage collection [55] transfers this burden to the runtime system and guarantees deallocation safety at the cost of some runtime overhead. Implemented in the first LISP systems, garbage collection has found its way in many modern languages such as Smalltalk, ML and Java.

The desire to have static control of memory management, retaining deallocation safety, led to the introduction of *regions* [14, 27, 46, 50]. Region annotations can be automatically inferred by means of a type and effect analysis [6, 47, 48, 51]. As such, it is possible to eliminate the runtime cost of trace-based garbage collection without losing deallocation safety. Based on the region calculus by Tofte and Talpin [51], the ML-kit [49], a compiler for the functional programming language Standard ML [34], implements a practical implementation of region

* Institut für Informatik, Georges-Köhler-Allee Geb. 079, D-79110 Freiburg, Germany. Tel: +49-7612038053, Fax: +49-7612038052, e-mail: shelsen@computer.org

inference for a realistic language. The ML-kit does not rely on runtime garbage collection[1] to obtain automatic memory management.

1.1 Contribution and Related Work

Since its introduction, region-based memory management has been improved [3, 7] and transferred to other languages [26, 28, 52]. In addition, many different flavors of the region calculus have been developed [4,9,12,13,20,23,53,54], usually to cater for more flexible and powerful memory management. Our syntactic theory focuses on the region calculus by Tofte and Talpin, which, henceforth, is being referred to as λ^{region}.

Whereas type soundness properties for region calculi have been extensively examined [5, 8, 10, 51, 56], little attention has been paid to the development of a semantic framework to reason about region-annotated programs. In this paper, we develop a typed equational theory to reason about region annotated programs. We formalize our theory for a small variation of the λ^{region}-calculus, including recursive function objects with polymorphic recursion for regions. In other work, we use our theory to prove the soundness of a specializer for an ML-polymorphic language [22]. We refrain from investigating an untyped equational theory since our region calculus is inherently type-based: region insertion is guided by a type and effect analysis.

Using bisimulation, a proof method originating from work on CCS [33], we prove that the equational theory is sound with respect to Morris-style contextual equivalence [35, 41]. Following methods by Gordon, Abramsky and others [2, 11, 16–19], we specify a notion of applicative bisimilarity for the region calculus. With a proof technique due to Howe [24, 25], re-casted to typed functional programming languages by Gordon [16], we show that bisimilarity is a congruence for our region calculus. We use this result to prove that bisimilarity equals contextual equivalence and includes the equational theory.

Plotkin [41] simplified Morris-style contextual equivalence to weak-head reduction and made it a standard notion to reason about program equivalence. However, since it is not very practical to prove contextual equivalence directly, alternative characterizations have been proposed. For instance, Milner [32] introduced his context lemma to reduce the need for full-blown contexts to evaluation contexts. The use of CCS-style bisimilarity [33] as a simpler operational method to reason about equivalences in functional programming languages, dates from the early 90s, when Abramsky [1] and Abramsky and Ong [2] define a notion of applicative bisimilarity for the lazy lambda calculus. Since then, bisimilarity has been extended and fine-tuned by Gordon [16–18] and Crole and Gordon [11] to other programming models. Notably, Gordon [18] proved that a form of bisimilarity for PCF with lazy streams is fully abstract with respect to contextual equivalence.

[1] The latest version 4 of the ML-kit actually incorporates a two-copy collector to optimize memory management within one region.

We closely follow Gordon's results on contextual equivalence for a polymorphically typed object calculus [17]. However, our syntactic theory is based on a concise small-step operational semantics from earlier work [21]. We found it therefore more natural to use a labeled transition system [11,18] to define bisimulation, although this is probably a matter of taste.

Applicative bisimilarity is not always complete with respect to contextual equivalence, for example, Gordon and Rees [19] found that a translation of Abramsky's applicative bisimilarity for a first-order object calculus with subtyping tells more programs apart than contextual equivalence. Experimental equivalence, a co-inductive variant of Milner's context lemma, turns out to be complete for such object calculi [17]. The bisimilarity notion presented in this paper is applicative and complete with respect to contextual equivalence.

Although the region calculus is designed to make memory operations explicit, it does not necessarily provide for local state. This is a non-trivial addition with respect to an equational theory. Mason and Talcott define a context lemma particularly tailored towards state operations. This is known as CIU-equivalence [29–31, 45]. Pitts [37] and Pitts and Stark [39] use logical relations and an unwinding theorem to prove equivalences in state-based languages. In this paper, we do not pursue these alternative methods since bisimilarity seems to work well for state-less polymorphic calculi [17, 36].

As far as we know, the only other work which directly discusses equational rules for the region calculus is a technical report by Dal Zilio and Gordon on the pi-calculus with name groups [56]. They derive their rules by looking at similar rules for their extended pi-calculus and conjecture soundness. However, they do not give proofs, treat only a monomorphic calculus and have a somewhat awkward equational theory due to a particular choice of formulating the type system.

1.2 Overview

Our language of discourse, the $\lambda^{\text{f-region}}$-calculus, is a simple generalization of λ^{region} and introduced in section 2. We give it a type system based on an effect analysis and specify its dynamic semantics as a small-step reduction relation. Section 3 introduces typed relations and open extensions. We use our small-step rewrite semantics to specify a notion of contextual equivalence in section 4. In section 5, we continue to define a typed equational theory for $\lambda^{\text{f-region}}$. In order to prove that this theory is sound with respect to contextual equivalence, section 6 defines applicative bisimilarity by means of a labeled transition system. A congruence property is proven in section 7, which is then used in section 8 to show that bisimilarity includes the equational theory. Finally, in section 9, we prove that bisimilarity equals contextual equivalence. Conclusions and future work follow.

$$\mathsf{Term} \ni e ::= x \mid e@e \mid \mathtt{let}\ x = e\ \mathtt{in}\ e \mid \Lambda\,\varrho.e \mid o\ \mathtt{at}\ \rho \mid v \mid e\,[\rho_1, \ldots, \rho_n]\ \mathtt{at}\ \rho$$

$$\mathsf{Object} \ni o ::= \lambda\, x.e \mid \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \qquad \mathsf{Value} \ni v ::= \langle o \rangle_\rho$$

$$x \in \mathsf{Var} \qquad \varrho \in \mathsf{RegionVar} \qquad \rho \in \mathsf{Placeholder} = \mathsf{RegionVar} \cup \{\bullet\}$$

Fig. 1. The $\lambda^{\text{f-region}}$-calculus

1.3 Notation

Let $f \in A \to B$ be a partial function from set A to set B. The *domain* of f is $Dom(f) = \{x \in A \mid \exists y \in B \wedge f(x) = y\}$ and its *range* is $Ran(f) = \{y \in B \mid \exists x \in A \wedge f(x) = y\}$. We write $\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$ for a function f with $Dom(f) = \{x_1, \ldots, x_n\}$ and $\{\,\}$ for the empty function. A function $f \in A \to B$ is *total* whenever $Dom(f) = A$. The extension $f + \{x \mapsto a\}$ of f is a partial function where $(f + \{x \mapsto a\})(x') = f(x')$ if $x' \in Dom(f) \setminus \{x\}$ and $(f + \{x \mapsto a\})(x) = a$ otherwise. The *co-restriction* $f - A$ of a partial function f is defined by $(f - A)(x) = f(x)$ if $x \in Dom(f) \setminus A$ and undefined otherwise. We write $f - x$ as an abbreviation for $f - \{x\}$. A sequence of symbols a_i is written $\boldsymbol{a}$ and the set of the elements in a sequence $\boldsymbol{a}$ is denoted by $\{\boldsymbol{a}\}$. Finally, the symbol $=$ denotes equality of terms modulo α-conversion.

2 The $\lambda^{\text{f-region}}$-calculus

The $\lambda^{\text{f-region}}$-calculus, is a small variation of the λ^{region}-calculus by Tofte and Talpin [51], similar in spirit to the computational extensions introduced in previous work [8, 21].

2.1 Syntax

Figure 1 shows the syntactic categories for $\lambda^{\text{f-region}}$, where Var and RegionVar are mutually disjoint denumerable sets, standing for expression and region variables respectively.

Variables, function application and the let-construct are standard. The region allocation construct $\Lambda\,\varrho.e$ binds a region of potentially unbounded memory to the region variable ϱ, evaluates e and finally deallocates the region before continuing with its context. We also introduce a distinct *dead* region variable $\bullet$, which serves as a sink region for deallocated objects. Operations involving $\bullet$ are unsound by definition. The region placeholder ρ is a convenient shorthand standing either for a region variable ϱ or the dead region $\bullet$.

Our calculus distinguishes two objects : a lambda abstraction $\lambda\, x.e$ and a recursive function $\mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e$. The latter binds the variables x and f in e, where f is bound to the function object itself. Very much as in the original

Tofte-Talpin calculus, a recursive function introduces region polymorphism by abstracting over the region variables $\varrho_1, \ldots, \varrho_n$. We require $n \geq 1$ to distinguish lambda objects and recursive function objects. It is an easy exercise to extend the theory of this paper for lambda objects, which are recursive, but not region-polymorphic.

An object o can be annotated with $\mathtt{at}\,\rho$, indicating that the object has to be stored in the region bound to ρ. In contrast, a value $\langle o \rangle_\rho$ is a pointer to an object, where the accompanying region, ρ, indicates where the object o currently lives. In either case, it is possible that the placeholder ρ stands for the dead region $\bullet$. The expression $o\,\mathtt{at}\,\bullet$ allocates the object o in the dead region, $\bullet$. Such a term is unsound and should only occur in code which is never executed, *i.e.* dead code. A value $\langle o \rangle_\bullet$ indicates that the object o is deallocated and can be interpreted as a dangling pointer. Note that we do not allow abstraction over the global dead region.

Finally, the construct $e\ [\rho_1, \ldots, \rho_n]\,\mathtt{at}\,\rho$ evaluates e to a recursive function value and applies the regions ρ_i to the abstracted region variables. The resulting lambda abstraction is then stored in the region ρ.

The grammar for $\lambda^{\text{f-region}}$ as described above actually defines *computational terms*. Terms produced by the grammar of Figure 1 without the dead region $\bullet$ and without values v are called *non-computational*. The distinction is not very important for our syntactic theory, but as in the Tofte-Talpin calculus, we may expect that compilers or programmers produce non-computational programs only.

2.2 Static Semantics

The static semantics for $\lambda^{\text{f-region}}$ is a simple adaptation of the type system found in [21, 51]. Its type language is generated by the following grammars, where EffectVar and TypeVar are disjoint denumerable sets of effect and type variables respectively:

$$\epsilon \in \mathsf{EffectVar} \qquad \varphi \subseteq \mathsf{Effect} = \mathsf{Placeholder} \cup \mathsf{EffectVar} \qquad \alpha \in \mathsf{TypeVar}$$

$$\mathsf{Type} \ni \tau ::= \alpha \mid \pi_1 \xrightarrow{\epsilon.\varphi} \pi_2 \qquad \mathsf{ResultType} \ni \hat{\sigma} ::= \forall\, \boldsymbol{\epsilon}\,.\forall\, \boldsymbol{\varrho}\,.\tau$$

$$\mathsf{TypeWPlace} \ni \pi ::= (\hat{\sigma}, \rho) \qquad \mathsf{TypeScheme} \ni \sigma ::= \forall\, \boldsymbol{\alpha}\,.\hat{\sigma}$$

An *effect*, φ, contains regions that may be affected when evaluating an expression. A *type* τ is either a type variable α or an arrow type $\pi_1 \xrightarrow{\epsilon.\varphi} \pi_2$. Its *arrow effect* $\epsilon.\varphi$ describes the effect of the body and is discharged on function application. The effect variable ϵ stands for the arrow effect of the function. It implements a mechanism for parametric polymorphism over arrow effects, particularly useful in the presence of higher-order functions [48]. Effect variables can also occur in an effect set. A *result type* $\hat{\sigma}$ abstracts a type over region and effect variables. As an extension to the Tofte-Talpin system, we take a *type with place* π to be a pair of a result type $\hat{\sigma}$ and a region ρ. Type variables, α, are taken from a denumerable set TypeVar and a *type scheme* σ abstracts a result type over such type variables.

A substitution, $S_s = (S_t, S_e, S_r)$, is a triple of the following total functions: a type substitution $S_t : \mathsf{TypeVar} \to \mathsf{Type}$, an effect substitution $S_e : \mathsf{EffectVar} \to \mathsf{EffectVar} \times \mathsf{Effect}$, and a region substitution $S_r : \mathsf{Placeholder} \to \mathsf{Placeholder}$. The definition of S_s on types and effects is as follows:

$$S_s(\varphi) = \{S_s(\rho) \mid \rho \in \varphi\} \cup \{\eta \mid \epsilon \in \varphi \wedge S_e(\epsilon) = \epsilon'.\varphi' \wedge \eta \in \{\epsilon'\} \cup \varphi'\}$$

$$S_s(\pi \xrightarrow{\epsilon.\varphi} \pi') = S_s(\pi) \xrightarrow{\epsilon'.\varphi'} S_s(\pi') \quad \textit{where } S_e(\epsilon) = \epsilon'.\varphi'' \textit{ and } \varphi' = \varphi'' \cup S_s(\varphi)$$

$$S_s(\rho) = S_r(\rho) \quad S_s(\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}) = \forall\, \boldsymbol{\alpha}\, .(S_s - \{\boldsymbol{\alpha}\})(\hat{\sigma}) \quad S_s(\hat{\sigma}, \rho) = (S_s(\hat{\sigma}), S_s(\rho))$$

$$S_s(\forall\, \boldsymbol{\epsilon}\, .\forall\, \boldsymbol{\varrho}\, .\tau) = \forall\, \boldsymbol{\epsilon}\, .\forall\, \boldsymbol{\varrho}\, .(S_s - (\{\boldsymbol{\epsilon}\} \cup \{\boldsymbol{\varrho}\}))(\tau) \quad S_s(\alpha) = S_t(\alpha)$$

The substitutions I_t and I_r are the identities on type and region variables, respectively, where the substitution I_e maps every effect variable ϵ to $\epsilon.\{\}$. We write S_r as shorthand for (I_t, I_e, S_r) and similarly for the other substitutions. The *support* of a type substitution, $\mathit{Supp}(S_t)$, is the set $\{\alpha \mid S_t(\alpha) \neq \alpha\}$. By abuse of notation we may write $\{\alpha_1 \mapsto S_t(\alpha_1), \ldots, \alpha_n \mapsto S_t(\alpha_n)\}$ whenever the $\mathit{Supp}(S_t) = \{\alpha_1, \ldots, \alpha_n\}$. The $\mathit{Supp}(S_r)$, $\mathit{Supp}(S_e)$ and $\mathit{Supp}(S_s)$ are defined analogously. Moreover, for an object O, we sometimes also write $O[\rho_1 \mapsto \rho'_1, \ldots, \rho_n \mapsto \rho'_n]$ instead of $\{\rho_1 \mapsto \rho'_1, \ldots, \rho_n \mapsto \rho'_n\}(O)$.

A type τ is an *instance* of a result type $\hat{\sigma} = \forall\, \boldsymbol{\epsilon}\, .\forall\, \boldsymbol{\varrho}\, .\tau'$ via substitution $S_s = (I_t, S_e, S_r)$, written $\tau \prec \hat{\sigma}$ *via* S_s, if $\mathit{Supp}(S_e) \subseteq \{\boldsymbol{\epsilon}\}$, $\mathit{Supp}(S_r) \subseteq \{\boldsymbol{\varrho}\}$ and $S_s(\tau') = \tau$. Similarly, we have that a result type $\hat{\sigma}$ is an instance of type scheme $\sigma = \forall\, \boldsymbol{\alpha}\, .\hat{\sigma}'$ via type substitution S_t, written $\hat{\sigma} \prec \sigma$ *via* S_t, if $\mathit{Supp}(S_t) \subseteq \{\boldsymbol{\alpha}\}$ and $S_t(\hat{\sigma}') = \hat{\sigma}$.

A type environment, $TE \in \mathsf{TypeEnv}$, is a partial function that maps variables to pairs of the form (σ, ρ). Substitutions extend to type environments, and expressions in the obvious way, avoiding capture by renaming. The free variables of a semantic object O are defined in the usual way: we write $\mathit{ftv}\,(O)$, $\mathit{frv}\,(O)$, and $\mathit{fev}\,(O)$ for O's free type, region (placeholder) and effect variables respectively. Also, $\mathit{fv}\,(O) = \mathit{ftv}\,(O) \cup \mathit{frv}\,(O) \cup \mathit{fev}\,(O)$.

It is often desirable to know whether a set semantic objects contains the dead region or not. To this end, define the predicate $\mathit{Clean}(A)$ for some set A to be true iff $\mathit{frv}\,(A) \subseteq \mathsf{RegionVar}$ and false otherwise. The negation of $\mathit{Clean}(\cdot)$ is $\mathit{Dirty}(\cdot)$, *i.e.* $\mathit{Dirty}(A)$ holds iff not $\mathit{Clean}(A)$. In the rest of the paper, we assume that for all substitutions S_s if holds that $\mathit{Clean}(\mathit{Supp}(S_s))$.

A typing judgment $TE \vdash_f e : (\hat{\sigma}, \rho), \varphi$ assigns type with place $(\hat{\sigma}, \rho)$ and effect φ to expression e given a type environment TE. The type rules are listed in Figure 2. Since the type rules for $\lambda^{\text{f-region}}$ are very similar to those of λ^{region} [21, 51], we only discuss some peculiarities specific to $\lambda^{\text{f-region}}$.

Because recursive functions are first-class objects, we have separated type polymorphism from region and effect polymorphism. Rules *(Fix)* and *(Fixv)* type a recursive function and its value counterpart respectively. As for λ^{region}, we have polymorphic recursion for region and effect variables, but not for type variables. Note that the use of result types in type judgments introduces higher-order region and effect polymorphism, which may make region reconstruction more

(Var)
$$\frac{TE(x) = (\sigma, \rho) \quad \hat{\sigma} \prec \sigma \ \mathit{via}\ S_t}{TE \vdash_f x : (\hat{\sigma}, \rho), \emptyset}$$

(Lam)
$$\frac{TE + \{x \mapsto \pi_1\} \vdash_f e : \pi_2, \varphi \quad \varphi \subseteq \varphi'}{TE \vdash_f \lambda\, x.e\, \mathtt{at}\, \rho : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \rho), \{\rho\}}$$

(Lamv)
$$\frac{TE + \{x \mapsto \pi_1\} \vdash_f e : \pi_2, \varphi \quad \varphi \subseteq \varphi'}{TE \vdash_f \langle \lambda\, x.e \rangle_\rho : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \rho), \emptyset}$$

(Reglam)
$$\frac{TE \vdash_f e : \pi, \varphi \quad \varrho \notin \mathit{frv}\,(TE, \pi)}{TE \vdash_f \Lambda\, \varrho.e : \pi, \varphi \setminus \{\varrho\}}$$

(App)
$$\frac{TE \vdash_f e_1 : (\pi_1 \xrightarrow{\epsilon.\varphi} \pi_2, \rho), \varphi_1 \quad TE \vdash_f e_2 : \pi_1, \varphi_2}{TE \vdash_f e_1 @\, e_2 : \pi_2, \varphi \cup \varphi_1 \cup \varphi_2 \cup \{\epsilon, \rho\}}$$

(Let)
$$\frac{\begin{array}{c} TE \vdash_f e_1 : (\hat{\sigma}', \rho'), \varphi' \quad \{\boldsymbol{\alpha}\} \cap \mathit{ftv}\,(TE) = \emptyset \\ TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}', \rho')\} \vdash_f e_2 : \pi, \varphi \end{array}}{TE \vdash_f \mathtt{let}\, x = e_1\, \mathtt{in}\, e_2 : \pi, \varphi \cup \varphi'}$$

(Fix)
$$\frac{\begin{array}{c} (\{\varrho_1, \ldots, \varrho_n\} \cup \{\boldsymbol{\epsilon}\}) \cap (\mathit{fv}\,(TE) \cup \{\rho\}) = \emptyset \\ TE + \{f \mapsto (\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \rho)\} \vdash_f \lambda\, x.e\, \mathtt{at}\, \rho : (\tau, \rho), \{\rho\} \end{array}}{TE \vdash_f \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\, \mathtt{at}\, \rho : (\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \rho), \{\rho\}}$$

(Fixv)
$$\frac{\begin{array}{c} (\{\varrho_1, \ldots, \varrho_n\} \cup \{\boldsymbol{\epsilon}\}) \cap (\mathit{fv}\,(TE) \cup \{\rho\}) = \emptyset \\ TE + \{f \mapsto (\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \rho)\} \vdash_f \langle \lambda\, x.e \rangle_\rho : (\tau, \rho), \emptyset \end{array}}{TE \vdash_f \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \rangle_\rho : (\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \rho), \emptyset}$$

(Regapp)
$$\frac{\begin{array}{c} TE \vdash_f e : (\hat{\sigma}, \rho), \varphi \quad \hat{\sigma} = \forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau \\ \tau' \prec \hat{\sigma} \ \mathit{via}\ (I_t, S_e, S_r) \quad S_r = \{\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n\} \end{array}}{TE \vdash_f e\, [\rho'_1, \ldots, \rho'_n]\, \mathtt{at}\, \rho' : (\tau', \rho'), \varphi \cup \{\rho, \rho'\}}$$

(Effect)
$$\frac{TE \vdash_f e : \pi, \varphi \quad \epsilon \notin \mathit{fev}\,(TE, \pi)}{TE \vdash_f e : \pi, \varphi \setminus \{\epsilon\}}$$

Fig. 2. Type System for $\lambda^{\text{f-region}}$

difficult or even undecidable. However, $\lambda^{\text{f-region}}$ allows for an elegant equational theory and we illustrate later on how λ^{region} can be treated as a special case of $\lambda^{\text{f-region}}$.

Finally, we list some elementary properties of the type system. The following two lemmas relate type judgments and their substitutions. They are both proven by induction on the type derivation:

Lemma 1. *If* $TE \vdash_f e : \pi, \varphi$ *then, for any type substitution* S_s*, we have that* $S_s(TE) \vdash_f S_s(e) : S_s(\pi), S_s(\varphi)$.

Lemma 2. *If for some* S_s*, we have* $S_s(TE) \vdash_f S_s(e) : S_s(\pi), S_s(\varphi)$*, then the judgment* $TE \vdash_f e : \pi, \varphi$ *holds as well.*

An expression substitution, $e[x \mapsto e']$, substitutes all free occurrences of x in e for e', avoiding name capture as usual. This leads to a variable substitution lemma:

Lemma 3. *Suppose* $TE \vdash_f v : (\hat{\sigma}, \rho), \emptyset$ *and take* $\boldsymbol{\alpha}$ *such that* $\{\boldsymbol{\alpha}\} \cap ftv\,(TE) = \emptyset$*. Additionally, assume* $TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}, \rho)\} \vdash_f e : \pi, \varphi$*. Then we have* $TE \vdash_f e[x \mapsto v] : \pi, \varphi$

Proof. Similar to the proof of the two substitution lemmas in [21]:

The following lemma allows for abstraction of variables in expressions.

Lemma 4. *Suppose* $TE \vdash_f v : (\hat{\sigma}, \rho), \emptyset$ *with* $\hat{\sigma} \prec \sigma$ *via* S_t *and that* $TE \vdash_f e[x \mapsto v] : \pi, \varphi$*. Then, we also have that* $TE + \{x \mapsto (\sigma, \rho)\} \vdash_f e : \pi, \varphi$.

Proof. A relatively straightforward induction proof on $TE \vdash_f e[x \mapsto v] : \pi, \varphi$, similar to the proof of Lemma 3.

2.3 Dynamic Semantics

The dynamic semantics of $\lambda^{\text{f-region}}$ is specified as a small-step reduction semantics in the style of Plotkin [42], similar in spirit to previous work [8, 21]. It requires no store because we do not handle destructive updates. Region allocation is implicit and only deallocation is visible in the reduction rules. Intuitively, every region variable, $\varrho \in \mathsf{RegionVar}$, in a term can be understood as some actual part of the heap, either local and scoped by a binder, or otherwise initial and global. Since region variables denote some region in the heap, renaming a region variable actually implies that we consider some different part in memory. This can happen by α-renaming a bound region variable or by substituting one region for another.

The relation $\overset{n}{\rightarrowtail} \in \mathcal{P}(\mathsf{Term} \times \mathbb{N} \times \mathsf{Term})$ specifies beta-reduction for $\lambda^{\text{f-region}}$ and is defined in Figure 3. Rules (1) and (2) allocate a lambda and a recursive function value respectively. Rule (3) deallocates a region after the enclosed expression is reduced to a value. Region deallocation is modeled by renaming the region variable ϱ to the dead region $\bullet$. All objects, previously in ϱ, are inaccessible now and the variable ϱ can be re-used. Beta-value reduction is specified by

$$\lambda\, x.e\, \mathtt{at}\, \varrho \overset{1}{\rightarrowtail} \langle \lambda\, x.e \rangle_{\varrho} \tag{1}$$

$$\mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\, \mathtt{at}\, \varrho \overset{2}{\rightarrowtail} \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \rangle_{\varrho} \tag{2}$$

$$\Lambda\, \varrho.v \overset{3}{\rightarrowtail} v[\varrho \mapsto \bullet] \tag{3}$$

$$\langle \lambda\, x.e \rangle_{\varrho} @\, v \overset{4}{\rightarrowtail} e[x \mapsto v] \tag{4}$$

$$\mathtt{let}\ x = v\ \mathtt{in}\ e \overset{5}{\rightarrowtail} e[x \mapsto v] \tag{5}$$

$$\begin{array}{c} \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \rangle_{\varrho}\, [\rho'_1, \ldots, \rho'_n]\, \mathtt{at}\, \varrho' \overset{6}{\rightarrowtail} \\ \langle \lambda\, x.e[\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n][f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \rangle_{\varrho}] \rangle_{\varrho'} \end{array} \tag{6}$$

Fig. 3. Beta rules

rules (4) and (5). Rule (6) implements region application, where the recursive function is bound to f *simultaneously* with the region substitutions.

A small-step semantics for $\lambda^{\text{f-region}}$ is defined by the relation $\rightharpoonup$ and its rules are listed in Figure 4. Rule (7) lifts beta-reduction into the small-step semantics, where $n \in \{1, \ldots, 6\}$. The rules (8) to (12) are standard for a left-to-right call-by-value language. The relation $\overset{*}{\rightharpoonup}$ is the reflexive and transitive closure of the one-step reduction relation $\rightharpoonup$.

We say that a term e *converges* ($e \downarrow$) iff there exists a value v such that $e \overset{*}{\rightharpoonup} v$. If for each $e \overset{*}{\rightharpoonup} e'$, there exists an e'' such that $e' \rightharpoonup e''$, then we say that e *diverges* ($e \uparrow$). As an example for a well-typed diverging expression, take for arbitrary $\hat{\sigma}$ and ϱ, the term

$$\Omega^{\hat{\sigma}}_{\varrho} = ((\mathtt{fix}\, f.\Lambda\, \varrho.\lambda\, x.(f\ [\varrho]\, \mathtt{at}\, \varrho) @\, x\, \mathtt{at}\, \varrho)\ [\varrho]\, \mathtt{at}\, \varrho) @\, (\lambda\, x.x\, \mathtt{at}\, \varrho)$$

We always have that $\Omega^{\hat{\sigma}}_{\varrho} \uparrow$ and $\{\ \} \vdash_f \Omega^{\hat{\sigma}}_{\varrho} : (\hat{\sigma}, \varrho), \{\varrho\}$.

The reduction relation obeys a type preservation property:

Proposition 1. *Suppose* $TE \vdash_f e : \pi, \varphi$. *If* $e \rightharpoonup e'$ *then there exist effects* φ' *such that* $TE \vdash_f e' : \pi, \varphi'$ *with* $\varphi' \subseteq \varphi$.

Proof. The proof is almost identical to the type preservation proof in [21].

The canonical forms lemma says something about the form of a value.

Lemma 5. *Suppose* $TE \vdash_f v : \pi, \emptyset$.

- *If* $\pi = (\pi_1 \xrightarrow{\epsilon.\varphi} \pi_2, \rho)$, *then* v *is of the form* $\langle \lambda\, x.e \rangle_{\rho}$
- *If* $\pi = \forall\, \boldsymbol{\epsilon}.\forall\, \varrho_1, \ldots, \varrho_n.\tau$ *and* $n \geq 1$,
  then v *is of the form* $\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \rangle_{\varrho}$

We also have a progress property:

$$\frac{e \overset{n}{\rightarrowtail} e'}{e \rightharpoonup e'} \qquad n \in \{1, \ldots, 6\} \tag{7}$$

$$\frac{e \rightharpoonup e'}{e @ e'' \rightharpoonup e' @ e''} \tag{8}$$

$$\frac{e \rightharpoonup e'}{\langle \lambda\, x.e'' \rangle_\rho @ e \rightharpoonup \langle \lambda\, x.e'' \rangle_\rho @ e'} \tag{9}$$

$$\frac{e \rightharpoonup e'}{\mathtt{let}\ x = e\ \mathtt{in}\ e'' \rightharpoonup \mathtt{let}\ x = e'\ \mathtt{in}\ e''} \tag{10}$$

$$\frac{e \rightharpoonup e'}{e\ [\rho_1, \ldots, \rho_n]\ \mathtt{at}\ \rho \rightharpoonup e'\ [\rho_1, \ldots, \rho_n]\ \mathtt{at}\ \rho} \tag{11}$$

$$\frac{e \rightharpoonup e'}{\Lambda\, \varrho.e \rightharpoonup \Lambda\, \varrho.e'} \tag{12}$$

Fig. 4. Reduction Semantics

Proposition 2. *Suppose* $\{\ \} \vdash_f e : \pi, \varphi$ *and* $Clean(\varphi)$*, then either* e *is a value* v *or there exists an* e' *such that* $e \rightharpoonup e'$*.*

Type soundness for $\lambda^{\text{f-region}}$ is as follows:

Theorem 1. *Suppose* $\{\ \} \vdash_f e : \pi, \varphi$ *and* $Clean(\varphi)$*. Then,* $e \downarrow$ *or* $e \uparrow$*.*

Proof. Simple induction on the reduction of e using Proposition 1 and 2.

Our small-step reduction is deterministic:

Lemma 6. *Suppose* $e \rightharpoonup e'$ *and* $e \rightharpoonup e''$*, then* $e' = e''$*.*

Reduction is invariant under expression substitution:

Lemma 7. *Suppose* $e \rightharpoonup e'$*, then* $e[x \mapsto v] \rightharpoonup e'[x \mapsto v]$*.*

Similarly, the reduction relation is invariant under region substitution, provided the substitution does not cause a dereference of dangling pointers.

Lemma 8. *Suppose* $TE \vdash_f e : \pi, \varphi$ *and for a substitution* S_s *it holds that* $Clean(S_s(\varphi))$*. Then* $e \rightharpoonup e'$ *implies* $S_s(e) \rightharpoonup S_s(e')$*.*

2.4 Relation to λ^{region}

As mentioned earlier, the use $\lambda^{\text{f-region}}$ instead of λ^{region} facilitates the formulation of an equational theory. We now illustrate that $\lambda^{\text{f-region}}$ is a proper generalization of the computational λ^{region}-calculus [21], defined by the following grammar:

$$\begin{aligned} \mathsf{Term}' \ni \mathbf{e} ::= {} & x \mid \mathbf{e} @ \mathbf{e} \mid \mathtt{let}\ x = \mathbf{e}\ \mathtt{in}\ \mathbf{e} \mid \Lambda\, \varrho.\mathbf{e} \mid \lambda\, x.\mathbf{e}\ \mathtt{at}\ \rho \mid \langle \lambda\, x.\mathbf{e} \rangle_\rho \mid \\ & \mathtt{letrec}\ f = \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\mathbf{e}\ \mathtt{at}\ \rho\ \mathtt{in}\ \mathbf{e} \mid \\ & \mathtt{letrec}\ f = \langle \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\mathbf{e} \rangle_\rho\ \mathtt{in}\ \mathbf{e} \mid f\ [\rho_1, \ldots, \rho_n]\ \mathtt{at}\ \rho \mid \\ & \langle \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\mathbf{e} \rangle_\rho\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \rho' \end{aligned}$$

We define a translation function $\|\cdot\| \in \mathsf{Term'} \to \mathsf{Term}$ inductively on $\mathsf{Term'}$-expressions, where we omit the cases that only propagate the translation to sub-terms:

$$\begin{array}{l}
\|\langle \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\mathbf{e}\rangle_\rho\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \rho'\| = \\
\qquad \langle \mathtt{fix}\, f''.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\|\mathbf{e}\|\rangle_\rho\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \rho' \quad \textit{where } f'' \notin \mathit{fv}\ (\mathbf{e}) \\
\|\mathtt{letrec}\ f = \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\mathbf{e}_1\ \mathtt{at}\ \rho\ \mathtt{in}\ \mathbf{e}_2\| = \\
\qquad \mathtt{let}\ f = \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\|\mathbf{e}_1\|\ \mathtt{at}\ \rho\ \mathtt{in}\ \|\mathbf{e}_2\| \\
\|\mathtt{letrec}\ f = \langle \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\mathbf{e}_1\rangle_\rho\ \mathtt{in}\ \mathbf{e}_2\| = \\
\qquad \mathtt{let}\ f = \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\|\mathbf{e}_1\|\rangle_\rho\ \mathtt{in}\ \|\mathbf{e}_2\|
\end{array}$$

The type system $\vdash^t_t$ and small-step reduction semantics $\to$ of λ^{region} are defined elsewhere [21] and we do not repeat them here. We have the following properties, which are proven by straightforward rule induction:

Proposition 3. *Suppose* $TE \vdash^t_t \mathbf{e} : (\tau, \rho), \varphi$, *then* $TE \vdash_f \|\mathbf{e}\| : (\tau, \rho), \varphi$.

Proposition 4. *Suppose* $\mathbf{e} \to \mathbf{e}'$, *then* $\|\mathbf{e}\| \rightharpoonup \|\mathbf{e}'\|$ *too.*

3 Typed Relations and Open extensions

Define a *typed relation* $\mathcal{R} \in \mathsf{TypedRel}$ as a relation on quintuples $(TE, e_1, e_2, \pi, \varphi)$ where $\mathsf{TypedRel} = \mathcal{P}(\mathsf{TypeEnv} \times \mathsf{Term} \times \mathsf{Term} \times \mathsf{TypeWPlace} \times \mathsf{Effect})$. We write $TE \rhd e_1\, \mathcal{R}\, e_2 : \pi, \varphi$ to say that $(TE, e_1, e_2, \pi, \varphi) \in \mathcal{R}$.

Definition 1. *A typed relation* $\mathcal{R}$ *is* well-formed *if the following two conditions hold simultaneously:*

1. *if* $TE \rhd e_1\, \mathcal{R}\, e_2 : \pi, \varphi$ *then* $TE \vdash_f e_1 : \pi, \varphi_1$ *and* $TE \vdash_f e_2 : \pi, \varphi_2$ *such that* $\varphi_1 \cup \varphi_2 \subseteq \varphi$ *with* $\mathit{Clean}(\varphi)$ *and* $TE \rhd e_1\, \mathcal{R}\, e_2 : \pi, \varphi_1 \cup \varphi_2$.
2. *if* $TE \rhd e_1\, \mathcal{R}\, e_2 : \pi, \varphi$ *then for all sets of effects* φ' *with* $\mathit{Clean}(\varphi')$, *also* $TE \rhd e_1\, \mathcal{R}\, e_2 : \pi, \varphi \cup \varphi'$ *holds.*

A typed relation $\mathcal{R}$ is reflexive if $TE \rhd e\, \mathcal{R}\, e : \pi, \varphi$, whenever $TE \vdash_f e : \pi, \varphi'$ and $\varphi' \subseteq \varphi$ with $\mathit{Clean}(\varphi)$. The inverse of a typed relation $\mathcal{R}$ as follows: $\mathcal{R}^{-1} = \{(TE, e_2, e_1, \pi, \varphi) \mid TE \rhd e_1\, \mathcal{R}\, e_2 : \pi, \varphi\}$. A typed relation $\mathcal{R}$ is *closed* ($\mathcal{R} \in \mathsf{ClosedTypedRel}$) iff $TE \rhd e_1\, \mathcal{R}\, e_2 : \pi, \varphi$ implies $TE = \{\ \}$.

It is sometimes necessary to extend a closed typed relation to a typed relation on non-closed terms. This leads to the notion of *open extension.*

First, define an *iterated substitution*, ξ, as a sequence of value substitutions, written $(x_1 \mapsto v_1, \ldots, x_n \mapsto v_n)$. We denote the empty sequence by ε. An iterated substitution works over $\lambda^{\text{f-region}}$-terms as follows:

$$\begin{array}{c}
\varepsilon(e) = e \\
(\xi, x \mapsto v)(e) = \xi(e[x \mapsto v])
\end{array}$$

An iterated substitution, ξ, is *a completion* of a type environment, TE, whenever $TE \succ \xi$ holds and $\succ$ is the least relation closed under the rules:

$$\{\ \} \succ \varepsilon \qquad \frac{TE(x) = (\sigma, \rho) \quad \hat{\sigma} \prec \sigma\ \textit{via}\ S_t \quad TE \vdash_f v : (\hat{\sigma}, \rho), \emptyset \quad TE - x \succ \xi}{TE \succ (\xi, x \mapsto v)}$$

The *open extension*, $\mathcal{R}^\circ$, of a closed typed relation $\mathcal{R}$ is now defined as follows:

$$\mathcal{R}^\circ = \{(TE, e_1, e_2, \pi, \varphi) \mid \begin{array}{l} \textit{for all } \xi \textit{ where } TE \succ \xi \\ \qquad \textit{it holds that } \{\,\} \rhd \xi(e_1)\,\mathcal{R}\,\xi(e_2) : \pi, \varphi\} \end{array}$$

We will often use that $\{\,\} \rhd e_1\,\mathcal{R}\,e_2 : \pi, \varphi$ whenever $\{\,\} \rhd e_1\,\mathcal{R}^\circ\,e_2 : \pi, \varphi$, which is trivially verified.

Lemma 9. *If $TE \vdash_f e : \pi, \varphi$ and $TE \succ \xi$, then $\{\,\} \vdash_f \xi(e) : \pi, \varphi$.*

Proof. This lemma is proven by a simple induction on ξ, using Lemma 3 for the induction step.

We also have the following useful lemma:

Lemma 10. *If $S_s(TE) \succ \xi'$, then there exists an iterated substitution ξ for which $TE \succ \xi$ and $\xi' = S_s(\xi)$.*

Proof. By induction on the length of ξ', using Lemma 2 in the induction step.

An iterated substitution is conveniently written as a sequence of nested let-expressions:

$$\begin{array}{l} \mathcal{T}(\varepsilon, e) = e \\ \mathcal{T}((\xi, x \mapsto v), e) = \texttt{let}\ x = v\ \texttt{in}\ \mathcal{T}(\xi, e) \end{array}$$

Open extensions do not affect well-formedness:

Lemma 11. *If $\mathcal{R}$ is a well-formed closed typed relation, then $\mathcal{R}^\circ$ is also well-formed.*

Proof. Proven by induction on the length of domain of the type environment and using Lemma 4 for the induction step.

We have a substitution lemma for open extensions of well-formed relations:

Lemma 12. *Suppose a well-formed closed typed relation $\mathcal{R}$. If $TE \vdash_f v : (\hat{\sigma}, \rho), \emptyset$ and $TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\,.\hat{\sigma}, \rho)\} \rhd e_1\,\mathcal{R}^\circ\,e_2 : \pi, \varphi$ where $\{\boldsymbol{\alpha}\} \cap \mathit{frv}\,(TE) = \emptyset$, then $TE \rhd e_1[x \mapsto v]\,\mathcal{R}^\circ\,e_2[x \mapsto v] : \pi, \varphi$.*

Proof. By definition of open extension, take ξ such that $TE \succ \xi$. So, we have to show that $\{\,\} \rhd \xi(e_1[x \mapsto v])\,\mathcal{R}\,\xi(e_2[x \mapsto v]) : \pi, \varphi$, which is by definition of iterated substitution the same as $\{\,\} \rhd (\xi, x \mapsto v)(e_1)\,\mathcal{R}\,(\xi, x \mapsto v)(e_1) : \pi, \varphi$.

Since $TE \succ \xi$, we have by definition of completion that for $\{\,\} \vdash_f v : (\hat{\sigma}', \rho), \emptyset$ where $\hat{\sigma}' \prec \forall\, \boldsymbol{\alpha}\,.\hat{\sigma}$ *via* S_t, it holds that $TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\,.\hat{\sigma}, \rho)\} \succ (\xi, x \mapsto v)$. By assumption and Lemma 1, we have $TE \vdash_f v : (\hat{\sigma}', \rho), \emptyset$. So, since it is given that $TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\,.\hat{\sigma}, \rho)\} \rhd e_1\,\mathcal{R}^\circ\,e_2 : \pi, \varphi$, the claim $\{\,\} \rhd (\xi, x \mapsto v)(e_1)\,\mathcal{R}\,(\xi, x \mapsto v)(e_1) : \pi, \varphi$ follows.

4 Contextual Equivalence

In a semantic framework based on evaluation, we are interested in a notion of equivalence that equates two expressions whenever they are *indistinguishable* in some sense to be defined. *Contextual equivalence* is such a notion of equality due to Morris [35] and popularized by Plotkin [41]. Intuitively, two expressions are contextual equivalent whenever an arbitrary context exhibits the same termination behavior within the operational semantics. Similar to Gordon [17], we formalize this notion for $\lambda^{\text{f-region}}$ with a style introduced by Lassen [44].

4.1 Compatible Refinement

Given a typed relation $\mathcal{R}$, we define its *compatible refinement* $\widehat{\mathcal{R}} \in \mathsf{TypedRel}$ as the least relation closed under the rules of Figure 5. The refinement rules are pretty much binary extensions of the type rules in Figure 2 without rule *(Effect)*. We say that a typed relation $\mathcal{R}$ is *compatible* iff $\widehat{\mathcal{R}} \subseteq \mathcal{R}$.

The main peculiarity of the compatible refinement definition is that the effects are always clean. This restriction is not unreasonable: for instance, a region annotated program produced by a compiler will obey this property. Additionally, we found the requirement necessary to have a usable notion of contextual equivalence in the first place.

For example, suppose the type with place $\pi = ((\alpha, \varrho') \xrightarrow{\epsilon.\{\}} (\alpha, \varrho'), \varrho')$ and consider the following assertion, relating two expressions of type π:

$$\{\,\} \rhd (\langle \lambda\, x.x \rangle_{\varrho} @ (\lambda\, w.w\ \mathtt{at}\ \varrho'))\ \mathcal{R}\ (\lambda\, w.w\ \mathtt{at}\ \varrho') : \pi, \varphi \tag{13}$$

Assume now that we do not require clean effects for contexts. For a compatible $\mathcal{R}$, we would then be able to derive the following context assertion of (13):

$$\begin{aligned}\{\,\} \rhd (\langle \mathtt{fix}\ f.\Lambda\, \varrho.\lambda\, z.(\underline{\langle \lambda\, x.x \rangle_{\varrho} @ (\lambda\, w.w\ \mathtt{at}\ \varrho')}) \rangle_{\varrho''}\ [\bullet]\ \mathtt{at}\ \varrho'') @ \langle \lambda\, z.z \rangle_{\varrho''}\ \mathcal{R} \\ (\langle \mathtt{fix}\ f.\Lambda\, \varrho.\lambda\, z.(\underline{\lambda\, w.w\ \mathtt{at}\ \varrho'}) \rangle_{\varrho''}\ [\bullet]\ \mathtt{at}\ \varrho'') @ \langle \lambda\, z.z \rangle_{\varrho''} : \pi, \varphi'\end{aligned} \tag{14}$$

Although it is quite natural to expect the expressions in (13) to be *equivalent*, for instance, they both evaluate to $\langle \lambda\, w.w \rangle_{\varrho'}$, it is easy to see that the two expressions in (14) are not. Indeed, the left-hand side expression does not reduce to any value - it tries to dereference $\bullet$, whereas the right-hand side evaluates to $\langle \lambda\, w.w \rangle_{\varrho'}$. The actual reason for this perhaps somewhat restrictive view of compatibility is that well typedness alone does not guarantee reduction as witnessed by Theorem 1. Clean effects *belong* in some sense to well typedness.

A compatible refinement has two useful properties, which are proven by rule induction on its definition.

Lemma 13. *Suppose a typed relation $\mathcal{R}$.*

1. *If $\mathcal{R}$ is compatible, then $\mathcal{R}$ is reflexive.*
2. *If $\mathcal{R}$ is well-formed, then $\widehat{\mathcal{R}}$ is well-formed as well.*

(Comp-var)

$$\frac{TE(x) = (\sigma, \rho) \quad \hat{\sigma} \prec \sigma \ \mathit{via}\ S_t \quad \mathit{Clean}(\varphi)}{TE \rhd x\, \widehat{\mathcal{R}}\, x : (\hat{\sigma}, \rho), \varphi}$$

(Comp-lam)

$$\frac{TE + \{x \mapsto \pi_1\} \rhd e\, \mathcal{R}\, e' : \pi_2, \varphi' \quad \mathit{Clean}(\varphi)}{TE \rhd \lambda\, x.e\ \mathtt{at}\ \varrho\, \widehat{\mathcal{R}}\, \lambda\, x.e'\ \mathtt{at}\ \varrho : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \varrho), \{\varrho\} \cup \varphi}$$

(Comp-lamv)

$$\frac{TE + \{x \mapsto \pi_1\} \rhd e\, \mathcal{R}\, e' : \pi_2, \varphi' \quad \mathit{Clean}(\varphi)}{TE \rhd \langle \lambda\, x.e \rangle_\rho\, \widehat{\mathcal{R}}\, \langle \lambda\, x.e' \rangle_\rho : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \rho), \varphi}$$

(Comp-reglam)

$$\frac{TE \rhd e\, \mathcal{R}\, e' : \pi, \varphi' \quad \varrho \notin \mathit{frv}\,(TE, \pi) \quad \mathit{Clean}(\varphi)}{TE \rhd \Lambda\, \varrho.e\, \widehat{\mathcal{R}}\, \Lambda\, \varrho.e' : \pi, (\varphi' \setminus \{\varrho\}) \cup \varphi}$$

(Comp-app)

$$\frac{\begin{array}{c} TE \rhd e_1\, \mathcal{R}\, e_1' : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \varrho), \varphi_1 \\ \mathit{Clean}(\varphi) \quad TE \rhd e_2\, \mathcal{R}\, e_2' : \pi_1, \varphi_2 \end{array}}{TE \rhd e_1 @ e_2\, \widehat{\mathcal{R}}\, e_1' @ e_2' : \pi_2, \varphi' \cup \varphi_1 \cup \varphi_2 \cup \{\epsilon, \varrho\} \cup \varphi}$$

(Comp-let)

$$\frac{\begin{array}{c} TE \rhd e_1\, \mathcal{R}\, e_1' : (\hat{\sigma}', \rho'), \varphi_1 \quad \{\boldsymbol{\alpha}\} \cap \mathit{ftv}\,(TE) = \emptyset \\ \mathit{Clean}(\varphi) \quad TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}', \rho')\} \rhd e_2\, \mathcal{R}\, e_2' : \pi, \varphi_2 \end{array}}{TE \rhd \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2\, \widehat{\mathcal{R}}\, \mathtt{let}\ x = e_1'\ \mathtt{in}\ e_2' : \pi, \varphi_1 \cup \varphi_2 \cup \varphi}$$

(Comp-fix)

$$\frac{\begin{array}{c} (\{\varrho_1, \ldots, \varrho_n\} \cup \{\boldsymbol{\epsilon}\}) \cap (\mathit{fv}\,(TE) \cup \{\rho\}) = \emptyset \\ \hat{\sigma} = \forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau \quad \mathit{Clean}(\varphi) \\ TE + \{f \mapsto (\hat{\sigma}, \rho)\} \rhd \lambda\, x.e\ \mathtt{at}\ \rho\, \mathcal{R}\, \lambda\, x.e'\ \mathtt{at}\ \rho : (\tau, \rho), \varphi' \end{array}}{\begin{array}{l} TE \rhd \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\ \mathtt{at}\ \rho\, \widehat{\mathcal{R}} \\ \quad \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\ \mathtt{at}\ \rho : (\hat{\sigma}, \rho), \varphi' \cup \varphi \end{array}}$$

(Comp-fixv)

$$\frac{\begin{array}{c} (\{\varrho_1, \ldots, \varrho_n\} \cup \{\boldsymbol{\epsilon}\}) \cap (\mathit{fv}\,(TE) \cup \{\rho\}) = \emptyset \\ \hat{\sigma} = \forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau \quad \mathit{Clean}(\varphi) \\ TE + \{f \mapsto (\hat{\sigma}, \rho)\} \rhd \langle \lambda\, x.e \rangle_\rho\, \mathcal{R}\, \langle \lambda\, x.e' \rangle_\rho : (\tau, \rho), \varphi' \end{array}}{\begin{array}{l} TE \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \rangle_\rho\, \widehat{\mathcal{R}} \\ \quad \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e' \rangle_\rho : (\hat{\sigma}, \rho), \varphi' \cup \varphi \end{array}}$$

(Comp-regapp)

$$\frac{\begin{array}{c} TE \rhd e\, \mathcal{R}\, e' : (\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \varrho), \varphi' \quad \mathit{Clean}(\varphi) \\ S_s = (I_t, S_e, \{\varrho_1 \mapsto \rho_1', \ldots, \varrho_n \mapsto \rho_n'\}) \quad \mathit{Supp}(S_e) \subseteq \{\boldsymbol{\epsilon}\} \end{array}}{\begin{array}{l} TE \rhd e\, [\rho_1', \ldots, \rho_n']\ \mathtt{at}\ \varrho'\, \widehat{\mathcal{R}} \\ \quad e'\, [\rho_1', \ldots, \rho_n']\ \mathtt{at}\ \varrho' : (S_s(\tau), \varrho'), \varphi' \cup \{\varrho, \varrho'\} \cup \varphi \end{array}}$$

Fig. 5. Compatible Refinement

Additionally, we have that the open extension of a compatible $\mathcal{R}$ can be encoded within $\mathcal{R}$ itself:

Lemma 14. *Suppose a typed relation $\mathcal{R}^\circ$ which is well-formed and compatible. Then if $TE \rhd e_1 \,\mathcal{R}^\circ\, e_2 : \pi, \varphi$ and $TE \succ \xi$, we also have that $\{\,\} \rhd \mathcal{T}(\xi, e_1)\,\mathcal{R}\,\mathcal{T}(\xi, e_2) : \pi, \varphi$*

Proof. Proven by simple induction on the length of ξ and using rule *(Comp-let)* in the induction step.

4.2 Context Closure

Define the *context closure* $\mathcal{R}^c$ of a typed relation $\mathcal{R}$ as the least relation, closed under the rules

$$\textit{(Closure-1)}\quad \frac{TE \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi}{TE \rhd e_1 \,\mathcal{R}^c\, e_2 : \pi, \varphi} \qquad\qquad \textit{(Closure-2)}\quad \frac{TE \rhd e_1 \,\widehat{\mathcal{R}^c}\, e_2 : \pi, \varphi}{TE \rhd e_1 \,\mathcal{R}^c\, e_2 : \pi, \varphi}$$

Intuitively, a context closure relates all typed contexts of the elements in a typed relation, where *(Closure-1)* defines the empty context. Therefore, the context closure of the one-element relation $\{(TE, e_1, e_2, \pi, \varphi)\}$ relates all typed contexts of the expressions e_1 and e_2.

Lemma 15. *Suppose a typed relation $\mathcal{R}$,*

1. *if $\mathcal{R}$ is well-formed, then so is $\mathcal{R}^c$.*
2. *the context closure $\mathcal{R}^c$ is compatible, and therefore, also reflexive.*
3. *compatible refinement and context closure are monotone operators in the lattice $(\mathsf{TypedRel}, \subseteq)$.*
4. *$\mathcal{R}$ is compatible iff $\mathcal{R}^c \subseteq \mathcal{R}$.*

One-element context closures have the following useful properties, all shown by induction on the derivations:

Lemma 16. *1. If $TE' \rhd e' \,\{(TE, e, e, \pi, \varphi)\}^c\, e'' : \pi', \varphi'$ then $e' = e''$.*

2. *If $TE' \rhd e'_1 \,\{(TE, e_1, e_2, \pi, \varphi)\}^c\, e'_2 : \pi', \varphi'$, then $TE' \rhd e'_2 \,\{(TE, e_2, e_1, \pi, \varphi)\}^c\, e'_1 : \pi', \varphi'$*
3. *Suppose $TE' \rhd e'_1 \,\{(TE, e_1, e_3, \pi, \varphi)\}^c\, e'_3 : \pi', \varphi'$, then for all e_2, there exists e'_2 with $TE' \rhd e'_1 \,\{(TE, e_1, e_2, \pi, \varphi)\}^c\, e'_2 : \pi', \varphi'$ and $TE' \rhd e'_2 \,\{(TE, e_2, e_3, \pi, \varphi)\}^c\, e'_3 : \pi', \varphi'$*
4. *If $TE' \rhd e'_1 \,\{(TE, e_1, e_2, \pi, \varphi)\}^c\, e'_2 : \pi', \varphi'$ and $TE'' \rhd e''_1 \,\{(TE', e'_1, e'_2, \pi', \varphi')\}^c\, e''_2 : \pi'', \varphi''$, then $TE'' \rhd e''_1 \,\{(TE, e_1, e_2, \pi, \varphi)\}^c\, e''_2 : \pi'', \varphi''$*

We can use the context closure of a typed relation to give an alternative characterization of compatibility provided the relation is a pre-order and well-formed:

Proposition 5. *Suppose a typed relation $\mathcal{R}$ which is reflexive, transitive and well-formed. Then, $\mathcal{R}$ is compatible iff $TE' \rhd e'_1 \,\mathcal{R}\, e'_2 : \pi', \varphi'$ whenever $TE \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ and $TE' \rhd e'_1 \,\{(TE, e_1, e_2, \pi, \varphi)\}^c\, e'_2 : \pi', \varphi'$.*

Proof. Case $\Rightarrow$. Take $TE \rhd e_1 \mathcal{R}\, e_2 : \pi, \varphi$, so $\{(TE, e_1, e_2, \pi, \varphi)\} \subseteq \mathcal{R}$. By Lemma 15.3, we have that $\{(TE, e_1, e_2, \pi, \varphi)\}^c \subseteq \mathcal{R}^c$. And since $\mathcal{R}$ is compatible, by Lemma 15.4, we also have $\mathcal{R}^c \subseteq \mathcal{R}$. This implies $\{(TE, e_1, e_2, \pi, \varphi)\}^c \subseteq \mathcal{R}$ proving the case.

Case $\Leftarrow$. For the other direction, we have to prove that $TE \rhd e_1 \,\widehat{\mathcal{R}}\, e_2 : \pi, \varphi$ implies $TE \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$, which is shown by case analysis:

Case *(Comp-var).* Immediate since rule *(Var)* implies $TE \vdash_f x : (\hat{\sigma}, \rho), \emptyset$ and we are given that $\mathcal{R}$ is reflexive.

Case *(Comp-let).* Suppose $TE \rhd \mathtt{let}\, x = e_1\, \mathtt{in}\, e_2 \,\widehat{\mathcal{R}}\, \mathtt{let}\, x = e_1'\, \mathtt{in}\, e_2' : \pi, \varphi$. Then $TE \rhd e_1 \,\mathcal{R}\, e_1' : (\hat{\sigma}', \rho'), \varphi_1$ and $TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}', \rho')\} \rhd e_2 \,\mathcal{R}\, e_2' : \pi, \varphi_2$ with $\{\boldsymbol{\alpha}\} \cap \mathit{ftv}\,(TE) = \emptyset$ and $\varphi_1 \cup \varphi_2 \subseteq \varphi$.

- Take $\mathcal{R}' = \{(TE, e_1, e_1', (\hat{\sigma}', \rho'), \varphi_1)\}$. By *(Closure-1)* we have that $TE \rhd e_1 \,\mathcal{R}'^c\, e_1' : (\hat{\sigma}', \rho'), \varphi_1$. Since $\mathcal{R}$ is well-formed and using Lemma 15.2, we obtain $TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}', \rho')\} \rhd e_2 \,\mathcal{R}'^c\, e_2 : \pi, \varphi_2$. Hence, using rule *(Comp-let)* we have $TE \rhd \mathtt{let}\, x = e_1\, \mathtt{in}\, e_2 \,\widehat{\mathcal{R}'^c}\, \mathtt{let}\, x = e_1'\, \mathtt{in}\, e_2 : \pi, \varphi$. Rule *(Closure-2)* and the assumption give $TE \rhd \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \,\mathcal{R}\, \mathtt{let}\ x = e_1'\ \mathtt{in}\ e_2 : \pi, \varphi$.
- By analogous reasoning, we also derive $TE \rhd \mathtt{let}\, x = e_1'\, \mathtt{in}\, e_2 \,\mathcal{R}\, \mathtt{let}\, x = e_1'\ \mathtt{in}\ e_2' : \pi, \varphi$.

By transitivity of $\mathcal{R}$, we conclude that $TE \rhd \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \,\mathcal{R}\, \mathtt{let}\ x = e_1'\ \mathtt{in}\ e_2' : \pi, \varphi$, which proves the case.

All other cases are proven by similar reasoning.

4.3 Contextual Equivalence

A typed relation $\mathcal{R}$ is *adequate* iff for all e_1, e_2, π and φ and $\{\ \} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ it holds that $e_1 \downarrow\!\Leftrightarrow e_2 \downarrow$.

Requiring that the context closure of a one-element relation is adequate yields the following definition of contextual equivalence:

Definition 2 (Contextual Equivalence). *Let contextual equivalence, $\simeq$, be a well-formed typed relation on terms such that $TE \rhd e_1 \simeq e_2 : \pi, \varphi$ iff $\{(TE, e_1, e_2, \pi, \varphi)\}^c$ is adequate.*

Although this definition of contextual equivalence uses the small-step semantics of section 2.3, we could have used the big-step evaluation style formulation of Tofte and Talpin [51] as well. In previous work [8], we prove that this choice does not make a fundamental difference.

Lemma 17. *Contextual equivalence is well-defined,* i.e. *an equivalence relation.*

Proof. Reflexivity follows from Lemma 16.1, symmetry from Lemma 16.2. For transitivity, suppose $TE \rhd e_1 \simeq e_2 : \pi, \varphi$ and $TE \rhd e_2 \simeq e_3 : \pi, \varphi$. By showing that for $\{\ \} \rhd e_1' \,\{(TE, e_1, e_3, \pi, \varphi)\}^c\, e_3' : \pi', \varphi'$ it holds that $e_1' \downarrow\!\Leftrightarrow e_3' \downarrow$, we have that $TE \rhd e_1 \simeq e_3 : \pi, \varphi$, *i.e.* transitivity.

Suppose $e_1' \downarrow$. Using Lemma 16.3, we have for e_2, an e_2' such that $\{\,\} \rhd e_1' \{(TE, e_1, e_2, \pi, \varphi)\}^c e_2' : \pi', \varphi'$ and $\{\,\} \rhd e_2' \{(TE, e_2, e_3, \pi, \varphi)\}^c e_3' : \pi', \varphi'$. Because $e_1' \downarrow$ and $TE \rhd e_1 \simeq e_2 : \pi, \varphi$, it must be that $e_2' \downarrow$. This, together with $TE \rhd e_2 \simeq e_3 : \pi, \varphi$ implies $e_3' \downarrow$. Symmetrically, we show that $e_3' \downarrow \Rightarrow e_1' \downarrow$, proving the case.

Proposition 6. *Contextual equivalence is the greatest typed relation that is well-formed, compatible and adequate.*

Proof. 1. By rule *(Closure-1)*, we immediately have that contextual equivalence is adequate and by definition well-formed.
Using Proposition 5 and Lemma 17, we show that contextual equivalence is compatible by proving that $TE' \rhd e_1' \simeq e_2' : \pi', \varphi'$ whenever $TE \rhd e_1 \simeq e_2 : \pi, \varphi$ and $TE' \rhd e_1' \{(TE, e_1, e_2, \pi, \varphi)\}^c e_2' : \pi', \varphi'$.
Since $TE \rhd e_1 \simeq e_2 : \pi, \varphi$, we obviously have that $\{(TE, e_1, e_2, \pi, \varphi)\}$ is well-formed and, by Lemma 15.1, $\{(TE, e_1, e_2, \pi, \varphi)\}^c$ is also well-formed.
For adequacy of $\{(TE', e_1', e_2', \pi', \varphi')\}^c$, assume $\{\,\} \rhd e_1'' \{(TE', e_1', e_2', \pi', \varphi')\}^c e_2'' : \pi'', \varphi''$. By Lemma 16.4 and the assumptions, we have $\{\,\} \rhd e_1'' \{(TE, e_1, e_2, \pi, \varphi)\}^c e_2'' : \pi'', \varphi''$. Also by assumption, we have $TE \rhd e_1 \simeq e_2 : \pi, \varphi$ and therefore $e_1'' \downarrow \Leftrightarrow e_2'' \downarrow$, showing that $\{(TE', e_1', e_2', \pi', \varphi')\}^c$ is adequate.
2. To show that $\simeq$ is the greatest well-formed, compatible and adequate typed relation, assume a typed relation $\mathcal{R}$, which is also well-formed, compatible and adequate. Take $TE \rhd e_1 \mathcal{R} e_2 : \pi, \varphi$.
Since $\mathcal{R}$ is compatible, we have from Lemma 15.3 and 15.4 that $\{(TE, e_1, e_2, \pi, \varphi)\}^c \subseteq \mathcal{R}$. Because $\mathcal{R}$ is also adequate, we derive that $\{(TE, e_1, e_2, \pi, \varphi)\}^c$ is adequate, from which we can conclude that $TE \rhd e_1 \simeq e_2 : \pi, \varphi$.

5 A Syntactic Equational Theory

Although contextual equivalence is a very powerful notion for term-equality, it is not very practical to use. In this section, we present a purely syntactic theory built on a set of typed equational rules. The rest of the paper shows that the theory is sound with respect to contextual equivalence. Define the relation $\doteq$ as the least typed relation closed under the rules in Figure 6.

Rule *(Eq-comp)* guarantees that the region theory is compatible. By Lemma 13.1, $\doteq$ is automatically reflexive as well. The rules *(Eq-symm)* and *(Eq-trans)* make the theory an equivalence. The beta-reduction rules from Figure 3 become typed and bidirectional in rule *(Eq-beta-n)*. The index n refers to the corresponding rules in Figure 3. Rule *(Eq-app-cbv)* equals call-by-value function application with a monomorphic `let`-expression

The rules *(Eq-app-let)*, *(Eq-let-let)* and *(Eq-drop)* express useful transformations we exploit in work on program specialization for ML-like languages [22]. Rule *(Eq-app-let)* is a let-insertion rule for function application, which makes the left-to-right evaluation of $\lambda^{\text{f-region}}$ more explicit. Rule *(Eq-let-let)* is standard let-flattening and allows for re-ordering of let-expressions. The rule *(Eq-drop)*,

$$\textit{(Eq-comp)}\quad \frac{TE \rhd e \mathrel{\widehat{=}} e' : \pi, \varphi}{TE \rhd e \doteq e' : \pi, \varphi} \qquad \textit{(Eq-symm)}\quad \frac{TE \rhd e' \doteq e : \pi, \varphi}{TE \rhd e \doteq e' : \pi, \varphi}$$

$$\textit{(Eq-trans)}\quad \frac{TE \rhd e \doteq e' : \pi, \varphi \quad TE \rhd e' \doteq e'' : \pi, \varphi}{TE \rhd e \doteq e'' : \pi, \varphi}$$

$$\textit{(Eq-beta-n)}\quad \frac{TE \vdash_f e : \pi, \varphi' \quad e \overset{n}{\rightarrowtail} e' \quad \varphi' \subseteq \varphi \quad \mathit{Clean}(\varphi)}{TE \rhd e \doteq e' : \pi, \varphi} \quad n \in \{1, 2, 5, 6\}$$

$$\textit{(Eq-app-cbv)}\quad \frac{\begin{array}{c} TE \vdash_f \langle \lambda\, x.e_1 \rangle_\varrho : (\pi_1 \xrightarrow{\epsilon.\varphi''} \pi, \varrho), \emptyset \\ TE \vdash_f e_2 : \pi_1, \varphi' \quad \varphi' \cup \varphi'' \cup \{\epsilon, \varrho\} \subseteq \varphi \quad \mathit{Clean}(\varphi) \end{array}}{TE \rhd \langle \lambda\, x.e_1 \rangle_\varrho @\, e_2 \doteq \mathtt{let}\ x = e_2\ \mathtt{in}\ e_1 : \pi, \varphi}$$

$$\textit{(Eq-app-let)}\quad \frac{\begin{array}{c} TE \vdash_f e_1 : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi, \varrho), \varphi_1 \quad f \notin fv\,(e_2) \\ TE \vdash_f e_2 : \pi_1, \varphi_2 \quad \varphi_1 \cup \varphi_2 \cup \varphi' \cup \{\epsilon, \varrho\} \subseteq \varphi \quad \mathit{Clean}(\varphi) \end{array}}{TE \rhd e_1 @\, e_2 \doteq \mathtt{let}\ f = e_1\ \mathtt{in}\ f @\, e_2 : \pi, \varphi}$$

$$\textit{(Eq-let-let)}\quad \frac{\begin{array}{c} TE \vdash_f e_1 : (\hat{\sigma}_1, \rho_1), \varphi_1 \quad \{\boldsymbol{\alpha}^1\} \cap ftv\,(TE) = \emptyset \quad \varphi_1 \cup \varphi_2 \cup \varphi_3 \subseteq \varphi \\ TE + \{x \mapsto (\forall\, \boldsymbol{\alpha}^1\, .\hat{\sigma}_1, \rho_1)\} \vdash_f e_2 : (\hat{\sigma}_2, \rho_2), \varphi_2 \quad x \notin frv\,(e_3) \quad \mathit{Clean}(\varphi) \\ \{\boldsymbol{\alpha}^2\} \cap ftv\,(TE) = \emptyset \quad TE + \{y \mapsto (\forall\, \boldsymbol{\alpha}^2\, .\hat{\sigma}_2, \rho_2)\} \vdash_f e_3 : \pi, \varphi_3 \end{array}}{\begin{array}{l} TE \rhd \mathtt{let}\ x = e_1\ \mathtt{in}\ (\mathtt{let}\ y = e_2\ \mathtt{in}\ e_3) \doteq \\ \qquad \mathtt{let}\ y = (\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2)\ \mathtt{in}\ e_3 : \pi, \varphi \end{array}}$$

$$\textit{(Eq-drop)}\quad \frac{TE \vdash_f e : \pi, \varphi' \quad \varrho \notin frv\,(TE, \pi) \quad \varphi' \subseteq \varphi \quad \mathit{Clean}(\varphi)}{TE \rhd e \doteq \Lambda\, \varrho.e : \pi, \varphi}$$

$$\textit{(Eq-dealloc)}\quad \frac{TE \vdash_f e : \pi, \varphi' \quad \varrho \notin frv\,(\pi) \cup \varphi' \quad \varphi' \subseteq \varphi \quad \mathit{Clean}(\varphi)}{TE \rhd e \doteq e[\varrho \mapsto \bullet] : \pi, \varphi}$$

Fig. 6. Equational Theory

specifies that the allocation of a region in a safe place (guaranteed by the condition $\varrho \notin frv\,(TE, \pi)$) does not change the meaning of an expression. Of course, it is hoped that region inference inserts such allocations as much and as early as possible. In fact, rule *(Eq-drop)* optimizes memory use when used from left to right, and deteriorates it when used from right to left. Finally, rule *(Eq-dealloc)* expresses that deallocation, when it is safe to do so, *does not matter*.

Note that we do not have the equational rules *(Eq-beta-3)* and *(Eq-beta-4)*. The former is covered by the rules *(Eq-drop)* and *(Eq-dealloc)* and the latter by the rules *(Eq-app-cbv)* and *(Eq-beta-5)*.

The typed equational theory by Dal Zilio and Gordon [56] is monomorphic and is subtly different due their presentation of the type system, which records allocated region variables. As a consequence, the rule *(Eq-drop)* from Figure 6 would not be well-formed. This is not the case in our type system and allows us

to derive the following rules:

$$\textit{(Eq-swap)}\quad \frac{\begin{array}{c} TE \vdash_f e : \pi, \varphi' \quad \{\varrho, \varrho'\} \cap \mathit{frv}\,(TE, \pi) = \emptyset \\ \varphi' \setminus \{\varrho, \varrho'\} \subseteq \varphi \quad \mathit{Clean}(\varphi) \end{array}}{TE \rhd \Lambda\,\varrho.\Lambda\,\varrho'.e \;\simeq\; \Lambda\,\varrho'.\Lambda\,\varrho.e : \pi, \varphi}$$

$$\textit{(Eq-reg-let)}\quad \frac{\begin{array}{c} TE \vdash_f e_1 : (\hat{\sigma}', \rho'), \varphi_1 \quad \{\boldsymbol{\alpha}\} \cap \mathit{ftv}\,(TE) = \emptyset \\ (\varphi_1 \cup \varphi_2) \setminus \{\varrho\} \subseteq \varphi \quad \mathit{Clean}(\varphi) \quad \varrho \notin \mathit{frv}\,(TE, (\hat{\sigma}', \rho'), \pi) \\ TE + \{x \mapsto (\forall\,\boldsymbol{\alpha}\,.\hat{\sigma}', \rho')\} \vdash_f e_2 : \pi, \varphi_2 \end{array}}{TE \rhd \Lambda\,\varrho.(\mathtt{let}\; x = e_1\; \mathtt{in}\; e_2) \;\simeq\; \mathtt{let}\; x = (\Lambda\,\varrho.e_1)\; \mathtt{in}\; (\Lambda\,\varrho.e_2) : \pi, \varphi}$$

$$\textit{(Eq-reg-lam)}\quad \frac{\begin{array}{c} TE + \{x \mapsto \pi_1\} \vdash_f e : \pi_2, \varphi' \quad \varphi' \subseteq \varphi'' \\ \varrho \notin \mathit{frv}\,(TE, (\pi_1 \xrightarrow{\epsilon.\varphi''} \pi_2, \varrho')) \quad \varrho' \in \varphi \quad \mathit{Clean}(\varphi) \end{array}}{TE \rhd \Lambda\,\varrho.(\lambda\,x.e\;\mathtt{at}\;\varrho') \;\simeq\; \lambda\,x.(\Lambda\,\varrho.e)\;\mathtt{at}\;\varrho' : (\pi_1 \xrightarrow{\epsilon.\varphi''} \pi_2, \varrho'), \varphi}$$

Dal Zilio and Gordon also specify an equational rule (Eq Drop). However, due to their formulation of the type system, it is weaker than *(Eq-drop)* and hence, they have to specify variants of the rules *(Eq-swap)* and *(Eq-reg-let)* as axioms in their equational theory.

Lemma 18. *The equational theory $\simeq$ is well-formed.*

Proof. Induction on the derivation, using Lemma 13.2, Proposition 1 and inspecting the premises.

We also have that the encoding of a completion is equal to the application of the completion on an expression:

Lemma 19. *If $TE \vdash_f e : \pi, \varphi$ and $TE \succ \xi$ then $\{\,\} \rhd \xi(e) \simeq \mathcal{T}(\xi, e) : \pi, \varphi$.*

Proof. Also a simple induction on the length of ξ using equational rule *(Eq-beta-5)*.

The equational theory as specified above is tied to computational terms. In practice, it is often desirable to prove program transformations on non-computational terms, for instance when verifying a compiler optimization. Such rules are somewhat clumsier, but easy to derive from the equational theory. We omit the details.

6 Bisimilarity for $\lambda^{\text{f-region}}$

The abstract property that two programs are contextual equivalent if no context can distinguish them, is intuitive, but does not yield a practical proof principle. We specify a form of *applicative bisimilarity* to prove program equivalences in $\lambda^{\text{f-region}}$ and use it to prove soundness of $\simeq$.

6.1 A Labeled Transition System

Following Crole and Gordon [11, 18], we use a labeled transition system to describe the observations one can make of a program. Observations are defined by the following grammar:

$$\mathsf{Observation} \ni \gamma ::= @\, v \mid [\rho_1, \ldots, \rho_n]\, \mathtt{at}\, \rho \mid \bullet$$

Except for $\bullet$, observations in $\lambda^{\text{f-region}}$ are applicative. The first kind of observation is a value application, the second applies regions. The last observation caters for deallocated code as we shall explain below. Note that the juxtaposition of an expression e with an observation γ, which is not $\bullet$, yields an new expression in $\lambda^{\text{f-region}}$, *i.e.* $e\gamma \in \mathsf{Term}$. However, the observation $[\rho_1, \ldots, \rho_n]\, \mathtt{at}\, \bullet$ is not very useful either since it always leads to an expressions that has a dirty effect. Therefore, we define *deadly* observations as follows: $\mathit{Deadly}(\gamma)$ is true iff $\gamma = \bullet$ or $\gamma = [\rho_1, \ldots, \rho_n]\, \mathtt{at}\, \bullet$. The predicate $\mathit{Safe}(\gamma)$ holds iff $\mathit{Deadly}(\gamma)$ is false.

Define a labeled transition system $\longrightarrow \in \mathcal{P}(\mathsf{Term} \times \mathsf{TypeWPlace} \times \mathsf{Effect} \times \mathsf{Observation} \times \mathsf{Term} \times \mathsf{TypeWPlace} \times \mathsf{Effect})$ inductively over the rules in Figure 7.

In a call-by-value language like $\lambda^{\text{f-region}}$, an expression is essentially characterized by its value. Hence, rule *(Obs-red)* says that the observations of a closed $\lambda^{\text{f-region}}$-term are the observations one can make of its value, provided it exists.

The rules *(Obs-fun)* and *(Obs-fix)* observe function application and region application respectively, provided the objects are not deallocated. In *(Obs-fun)*, we apply a value, not an expression. This decision is not of great importance since expressions are characterized by their values, but it facilitates proofs. Also, we require the arrow effect to be clean as we want to avoid that the resulting effect of an observation is dirty.

The condition $\mathit{frv}\,(v) \subseteq \mathit{frv}\,(\pi'')$ implies that dead code within v has to use regions from its type. In rule *(Obs-fix)* we have a similar restriction on the applied regions. We basically say that it is sufficient to look at applicative observations that happen within the same part of memory as the observed value uses. This leads to a useful notion of bisimilarity for a region calculus: free regions occurring in observations are determinable from the type of the observed value. We will exploit this feature to prove the equational theory a bisimilarity.

Since we want to make an observation of *every* well-typed value, we have the rules *(Obs-deallocated)* and *(Obs-dangerous)*, which cater for values not covered by *(Obs-fun)* and *(Obs-fix)*. The transition rule *(Obs-deadalloc)* allows the applicative observation $[\rho_1, \ldots, \rho_n]\, \mathtt{at}\, \bullet$ and is only present for proof-technical reasons.

Transition rules with deadly observations are designed to catch values that have a dirty effect when applicatively observed. Therefore, these transitions go to the non-terminating program $\Omega^{\alpha}_{\varrho_\star}$ which disallows further observations. The region variable $\varrho_\star$ is fixed throughout the rest of the paper to guarantee a deterministic transition system.

Note that the two observation rules *(Obs-fun)* and *(Obs-fix)* apply the evaluation contexts (9) and (11) from Figure 4 respectively - except for the value in *(Obs-fun)*. Continuing along these lines, one could include labeled transitions

$$\textit{(Obs-red)}\quad \frac{\{\,\}\vdash_f e:\pi,\varphi'' \quad \varphi''\subseteq\varphi \qquad e \rightharpoonup e'' \quad (e'':\pi,\varphi)\xrightarrow{\gamma}(e':\pi',\varphi')}{(e:\pi,\varphi)\xrightarrow{\gamma}(e':\pi',\varphi')}$$

$$\textit{(Obs-fun)}\quad \frac{\{\,\}\vdash_f \langle\lambda\, x.e\rangle_\varrho : (\pi''\xrightarrow{\epsilon.\varphi'}\pi',\varrho),\emptyset \qquad \mathit{Clean}(\varphi') \quad \{\,\}\vdash_f v:\pi'',\emptyset \quad \mathit{frv}\,(v)\subseteq\mathit{frv}\,(\pi'')}{(\langle\lambda\, x.e\rangle_\varrho : (\pi''\xrightarrow{\epsilon.\varphi'}\pi',\varrho),\varphi_0)\xrightarrow{@\,v}(\langle\lambda\, x.e\rangle_\varrho @\, v:\pi',\varphi'\cup\{\epsilon,\varrho\})}$$

$$\textit{(Obs-fix)}\quad \frac{\begin{array}{c}\{\,\}\vdash_f \langle \mathtt{fix}\, f.\Lambda\,\varrho_1,\ldots,\varrho_n.\lambda\, x.e\rangle_\varrho : (\forall\,\boldsymbol{\epsilon}.\forall\,\varrho_1,\ldots,\varrho_n.\tau,\varrho),\emptyset \\ S_s=(I_t,S_e,\{\varrho_1\mapsto\rho_1',\ldots,\varrho_n\mapsto\rho_n'\}) \quad \mathit{Supp}(S_e)\subseteq\{\boldsymbol{\epsilon}\} \\ \{\rho_1',\ldots,\rho_n',\varrho'\}\subseteq \mathit{frv}\,(\forall\,\boldsymbol{\epsilon}.\forall\,\varrho_1,\ldots,\varrho_n.\tau,\varrho)\cup\{\varrho_1,\ldots,\varrho_n\}\end{array}}{\begin{array}{l}(\langle \mathtt{fix}\, f.\Lambda\,\varrho_1,\ldots,\varrho_n.\lambda\, x.e\rangle_\varrho : (\forall\,\boldsymbol{\epsilon}.\forall\,\varrho_1,\ldots,\varrho_n.\tau,\varrho),\varphi_0)\xrightarrow{[\rho_1',\ldots,\rho_n']\,\mathtt{at}\,\varrho'} \\ \qquad (\langle \mathtt{fix}\, f.\Lambda\,\varrho_1,\ldots,\varrho_n.\lambda\, x.e\rangle_\varrho\,[\rho_1',\ldots,\rho_n']\,\mathtt{at}\,\varrho' : (S_s(\tau),\varrho'),\{\varrho,\varrho'\})\end{array}}$$

$$\textit{(Obs-deallocated)}\quad \frac{\{\,\}\vdash_f \langle o\rangle_\bullet : \pi,\emptyset}{(\langle o\rangle_\bullet:\pi,\varphi_0)\xrightarrow{\bullet}(\Omega^\alpha_{\varrho_\star}:(\alpha,\varrho_\star),\{\varrho_\star\})}$$

$$\textit{(Obs-dangerous)}\quad \frac{\{\,\}\vdash_f \langle\lambda\, x.e\rangle_\varrho : (\pi\xrightarrow{\epsilon.\varphi}\pi',\varrho),\emptyset \quad \mathit{Dirty}(\varphi)}{(\langle\lambda\, x.e\rangle_\varrho : (\pi\xrightarrow{\epsilon.\varphi}\pi',\varrho),\varphi_0)\xrightarrow{\bullet}(\Omega^\alpha_{\varrho_\star}:(\alpha,\varrho_\star),\{\varrho_\star\})}$$

$$\textit{(Obs-deadalloc)}\quad \frac{\{\,\}\vdash_f \langle \mathtt{fix}\, f.\Lambda\,\varrho_1,\ldots,\varrho_n.\lambda\, x.e\rangle_\varrho : (\hat\sigma,\varrho),\emptyset}{\begin{array}{l}(\langle \mathtt{fix}\, f.\Lambda\,\varrho_1,\ldots,\varrho_n.\lambda\, x.e\rangle_\varrho : (\hat\sigma,\varrho),\varphi_0)\xrightarrow{[\rho_1',\ldots,\rho_n']\,\mathtt{at}\,\bullet} \\ \qquad (\Omega^\alpha_{\varrho_\star}:(\alpha,\varrho_\star),\{\varrho_\star\})\end{array}}$$

Fig. 7. Observations in $\lambda^{\text{f-region}}$

which correspond to the other evaluation contexts (8), (10), and (12). However, these observations do not allow us to observe more. In particular, the region deallocation rule - context rule (12) - is not a distinguishing context since region allocations and deallocations do not influence the termination semantics of the region calculus. A notion of bisimilarity based on all evaluation contexts is called experimental equivalence by Gordon [17, 18] and would provide an alternative formulation of Milner's context Lemma [32].

The labeled transition system described above is a superset of the system that is required to define bisimilarity for a polymorphically typed call-by-value lambda calculus. This suggests that the region calculus enriches a Hindley-Milner polymorphic lambda calculus without affecting observational properties. Such a connection has been investigated before, see for instance [5].

The design of the transition system has some very useful properties. The following lemma expresses that a well-typed converging expression always allows a transition:

Lemma 20. *Suppose* $\{\ \} \vdash_f e : \pi, \varphi''$ *where* $\varphi'' \subseteq \varphi$ *and* $e \downarrow$. *Then the transition* $(e : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$ *exists. Moreover, if* $Safe(\gamma)$, *we have for* $e \xrightarrow{*} v$ *that* $e' = v\gamma$.

Proof. By induction on the number of reductions in $e \xrightarrow{*} v$. If e is the value v, then one of the rules *(Obs-fun)*, *(Obs-fix)*, *(Obs-deallocated)* or *(Obs-dangerous)* applies, depending on the actual form of π. Whenever $Safe(\gamma)$, we also have $e' = v\gamma$ for $e \xrightarrow{*} v$ by the rules *(Obs-fun)* and *(Obs-fix)*. Alternatively, we have $e \rightharpoonup e''$ where $e'' \xrightarrow{*} v$. By Proposition 1, we also have $\{\ \} \vdash_f e'' : \pi, \varphi'''$ with $\varphi''' \subseteq \varphi'' \subseteq \varphi$. By induction, this yields the transition $(e'' : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$. Hence, rule *(Obs-red)* yields $(e : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$.

The opposite property also holds:

Lemma 21. *Suppose a transition* $(e : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$. *Then* $\{\ \} \vdash_f e : \pi, \varphi''$ *and* $\varphi'' \subseteq \varphi$ *with* $e \downarrow$. *Whenever* $Safe(\gamma)$, *it also holds that* $e' = v\gamma$ *for* $e \xrightarrow{*} v$.

Proof. First, see by inspection of the rules in Figure 7 that always $\{\ \} \vdash_f e : \pi, \varphi''$ with $\varphi'' \subseteq \varphi$.

Whenever the transition is due to one of the rules *(Obs-fun)*, *(Obs-fix)*, *(Obs-deallocated)*, *(Obs-dangerous)* or *(Obs-deadalloc)*, we immediately have $e \downarrow$. If $Safe(\gamma)$, it also holds that $e' = v\gamma$ for $e \xrightarrow{*} v$ by the rules *(Obs-fun)* and *(Obs-fix)*. In case of *(Obs-red)*, we have $e \rightharpoonup e''$ and $(e'' : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$. By induction we have $e'' \downarrow$ and hence also $e \downarrow$.

A transition preserves typing and guarantees a clean result effect:

Lemma 22. *Suppose a transition* $(e : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$, *then* $\{\ \} \vdash_f e' : \pi', \varphi'$ *with* $Clean(\varphi')$.

Proof. Simple proof by rule induction on the transition system and inspection of the premises.

The transition system is deterministic:

Lemma 23. *If* $(e_1 : \pi_1, \varphi_1) \xrightarrow{\gamma} (e_2 : \pi_2, \varphi_2)$ *and* $(e_1 : \pi_1, \varphi_1) \xrightarrow{\gamma} (e_2' : \pi_2, \varphi_2)$, *then* $e_2 = e_2'$.

Proof. By induction on the derivations and use of Lemma 6.

Convergence does not only imply a transition. We can also choose it to coincide with the observation of another expression of the same type.

Lemma 24. *Suppose* $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$ *and for* $\{\,\} \vdash_f e_2 : \pi, \varphi''$ *with* $\varphi'' \subseteq \varphi$ *it holds that* $e_2 \downarrow$. *Then there exists a transition* $(e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$. *Moreover, if* $Safe(\gamma)$, *then* $e_2' = v_2\gamma$ *for* $e_2 \xrightarrow{*} v_2$.

Proof. Straightforward by rule induction on the labeled transition system, where we note that the choice of observation is guided by the form of the type π.

We have a substitution lemma for transitions, but we need an extended notion of substitution since it is generally possible that an observation is *flushed* into a deadly observation. Therefore, define the following substitution auxiliaries, where we assume that they are implicitly parameterized over the substitution S_s:

$$S_s^o(\pi,\gamma) = \begin{cases} \bullet & \text{whenever } \pi = (\hat{\sigma}, \rho) \text{ and } S_s(\rho) = \bullet \\ \bullet & \text{whenever } \pi = (\pi_1 \xrightarrow{\epsilon.\varphi} \pi_2, \rho),\ S_s(\rho) \neq \bullet \text{ and } Dirty(S_s(\varphi)) \\ S_s(\gamma) & \text{otherwise} \end{cases}$$

$$S_s^e(\gamma, e) = \begin{cases} \Omega^{\alpha}_{\varrho_\star} & \text{if } Deadly(\gamma) \\ S_s(e) & \text{if } Safe(\gamma) \end{cases} \qquad S_s^t(\gamma, \pi) = \begin{cases} (\alpha, \varrho_\star) & \text{if } Deadly(\gamma) \\ S_s(\pi) & \text{if } Safe(\gamma) \end{cases}$$

$$S_s^f(\gamma, \varphi) = \begin{cases} \{\varrho_\star\} & \text{if } Deadly(\gamma) \\ S_s(\varphi) & \text{if } Safe(\gamma) \end{cases}$$

The substitution lemma is now as follows

Lemma 25. *Suppose a transition* $(e : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$ *and* $Clean(S_s(\varphi))$. *Then* $(S_s(e) : S_s(\pi), S_s(\varphi)) \xrightarrow{\gamma'} (S_s^e(\gamma', e') : S_s^t(\gamma', \pi'), S_s^f(\gamma', \varphi'))$ *exists too where* $\gamma' = S_s^o(\pi, \gamma)$.

Proof. The proof follows from a simple induction argument on the labeled transition system, using Lemma 8 for *(Obs-red)* and Lemma 1 for the other transitions.

The following lemma is useful to be able to use the above substitution lemma:

Lemma 26. *Suppose* $(S_s(e) : S_s(\pi), S_s(\varphi)) \xrightarrow{\gamma'} (e' : \pi', \varphi')$. *Then, the transition* $(e : \pi, \varphi) \xrightarrow{\gamma''} (e'' : \pi'', \varphi'')$ *exists such that* $S_s^o(\pi, \gamma'') = \gamma'$, $S_s^e(\gamma', e'') = e'$, $S_s^t(\gamma', \pi'') = \pi'$ *and* $S_s^f(\gamma', \varphi'') = \varphi'$.

Proof. Relatively straightforward induction argument, where we use that for an object O, we always can choose an O' such that $S_s(O') = O$.

6.2 Simulation and Bisimulation

For every well-typed $\lambda^{\text{f-region}}$-program, the labeled transition system produces a (potentially infinite) tree with terms in its nodes and observations on the edges. Intuitively, we expect two programs to be equivalent iff their corresponding observation trees are identical modulo the terms occurring in their nodes. Bisimulation makes this notion of equivalence exact.

Definition 3. *The functions* $\lceil . \rceil, [.] \in \mathsf{ClosedTypedRel} \to \mathsf{ClosedTypedRel}$ *are defined as follows:*

$$\begin{aligned}
\lceil \mathcal{R} \rceil = \{(\{\,\}, e_1, e_2, \pi, \varphi) \mid\ & \textit{whenever } (e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi') \textit{ implies an} \\
& e_2' \textit{ such that } (e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi') \textit{ and} \\
& \{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi' \cup \varphi'' \textit{ with } \mathit{Clean}(\varphi'')\} \\
[\mathcal{R}] = \lceil \mathcal{R} \rceil \cap \lceil \mathcal{R}^{-1} \rceil^{-1} & \\
= \lceil \mathcal{R} \rceil \cap \{(\{\,\}, e_1, e_2, \pi, \varphi) \mid\ & \textit{whenever } (e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi') \textit{ implies} \\
& \textit{an } e_1' \textit{ such that } (e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi') \\
& \textit{and } \{\,\} \rhd e_2' \,\mathcal{R}\, e_1' : \pi', \varphi' \cup \varphi'' \textit{ with } \mathit{Clean}(\varphi'')\}
\end{aligned}$$

Observe that the operators $\lceil . \rceil$ and $[.]$ are monotone in the complete lattice $(\mathsf{ClosedTypedRel}, \subseteq)$. They lead to the notions of similarity and bisimilarity:

Definition 4. *1. A well-formed closed typed relation* $\mathcal{R} \in \mathsf{ClosedTypedRel}$*, which is* $\lceil . \rceil$*-dense,* i.e. $\mathcal{R} \subseteq \lceil \mathcal{R} \rceil$*, is a* simulation. Similarity, $\lesssim$*, is defined as the union of all simulations.*

2. A well-formed closed typed relation $\mathcal{R} \in \mathsf{ClosedTypedRel}$*, which is* $[.]$*-dense,* i.e. $\mathcal{R} \subseteq [\mathcal{R}]$*, is a* bisimulation. *The union of all bisimulations is the* Bisimilarity *relation,* $\approx$*.*

Note that by Lemma 22 our notion of (bi)similarity is well-defined. A more manageable definition of simulation is as follows: For a well-formed relation $\mathcal{R}$ we have that $\mathcal{R}$ is a simulation iff $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ implies that if $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$ then $(e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$ exists such that $\{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi' \cup \varphi''$ with $\mathit{Clean}(\varphi'')$.

Because of the Knaster-Tarski fix-point theorem, we have that similarity is the greatest simulation and bisimilarity the greatest bisimulation, amounting to the following co-inductive proof principles [18], where we assume that $\mathcal{R}$ is well-formed:

Co-induction	$\mathcal{R} \subseteq [\mathcal{R}]$ *implies* $\mathcal{R} \subseteq \approx$
Strong Co-induction	$\mathcal{R} \subseteq [\mathcal{R} \cup \approx] \cup \approx$ *implies* $\mathcal{R} \subseteq \approx$

Additionally, we have that similarity and bisimilarity are fixed points for $\lceil . \rceil$ and $[.]$ respectively, *i.e.* $\lceil \lesssim \rceil = \lesssim$ and $[\approx] = \approx$.

Since both similarity and bisimilarity are defined as closed typed relations, we will often be interested in their open extensions, *i.e.* the relations $\lesssim^\circ$ and $\approx^\circ$ respectively.

6.3 Properties

We show that the open extension of bisimilarity is a well-formed equivalence relation:

Proposition 7. *1.* $\lesssim^\circ$ *is a pre-order,* i.e. *reflexive and transitive*

2. $\approx^\circ$ *is an equivalence,* i.e. *reflexive, symmetric and transitive*
3. $\approx^\circ$ *is well-formed*

Proof. 1. To show reflexivity, it is easy to see by definition of simulation that $\{(\{\,\}, e, e, \pi, \varphi) \mid \{\,\} \vdash_f e : \pi, \varphi' \textit{ with } \varphi' \subseteq \varphi \textit{ and } \mathit{Clean}(\varphi)\}$ is well-formed and $\lceil . \rceil$-dense, using Lemma 22. Hence, by co-induction, $\lesssim$ is reflexive and trivially also $\lesssim^\circ$.
For transitivity, assume $\mathcal{R} = \{(\{\,\}, e_1, e_3, \pi, \varphi) \mid$ *there exists* e_2 *such that* $\{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi$ *and* $\{\,\} \rhd e_2 \lesssim e_3 : \pi, \varphi\}$. The relation $\lesssim$ is transitive iff we have that $\mathcal{R} \subseteq \lesssim$. By co-induction, it is sufficient to show that $\mathcal{R} \subseteq \lceil \mathcal{R} \rceil$. So, suppose $\{\,\} \rhd e_1 \,\mathcal{R}\, e_3 : \pi, \varphi$. Then $\{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi$ and $\{\,\} \rhd e_2 \lesssim e_3 : \pi, \varphi$ for some e_2. By simulation, this means that $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$ implies $(e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$ such that $\{\,\} \rhd e_1' \lesssim e_2' : \pi', \varphi' \cup \varphi''$ with $\mathit{Clean}(\varphi'')$. Therefore also, $(e_3 : \pi, \varphi) \xrightarrow{\gamma} (e_3' : \pi', \varphi')$ and $\{\,\} \rhd e_2' \lesssim e_3' : \pi', \varphi' \cup \varphi''$. Hence, $\{\,\} \rhd e_1' \,\mathcal{R}\, e_3' : \pi', \varphi' \cup \varphi''$, which proves transitivity of $\lesssim$. Since type environments remain invariant under transitivity, the property holds trivially for $\lesssim^\circ$.
2. Reflexivity and transitivity follow by item 1. Symmetry is a similar co-induction argument using the definition of bisimilarity.
3. Since bisimilarity is by definition closed and well-formed, we have the property immediately by Lemma 11.

Determinism of the transition system makes bisimilarity equivalent to mutual similarity:

Proposition 8. *TE* $\rhd e_1 \approx^\circ e_2 : \pi, \varphi$ *iff TE* $\rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$ *and TE* $\rhd e_2 \lesssim^\circ e_1 : \pi, \varphi$ *(*$\approx^\circ = \lesssim^\circ \cap \lesssim^{\circ -1}$*).*

Proof. It is immediate that when the property holds for similarity (respectively bisimilarity), it also holds for their open extensions. Hence, we prove the proposition for similarity and bisimilarity.

We observe that $\approx = [\approx] = \lceil \approx \rceil \cap \lceil \approx^{-1} \rceil^{-1}$. Hence, we have $\approx \subseteq \lceil \approx \rceil$ and by co-induction that $\approx \subseteq \lesssim$. So, by Proposition 7.2, also $\approx \subseteq \lesssim^{-1}$, implying $\approx \subseteq \lesssim \cap \lesssim^{-1}$.

For the other direction, take the well-formed typed relation $\mathcal{R} = \lesssim \cap \lesssim^{-1} = \{(\{\,\}, e_1, e_2, \pi, \varphi) \mid \{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi \textit{ and } \{\,\} \rhd e_2 \lesssim e_1 : \pi, \varphi\}$. By co-induction, it suffices to show that $\mathcal{R} \subseteq [\mathcal{R}]$. Since $\mathcal{R}$ is symmetric and $[\mathcal{R}] = \lceil \mathcal{R} \rceil \cap \lceil \mathcal{R}^{-1} \rceil^{-1}$, we have to prove $\mathcal{R} \subseteq \lceil \mathcal{R} \rceil$. So, take $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ and suppose $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$. By definition of $\mathcal{R}$, there exists an e_2' such that $(e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$ and $\{\,\} \rhd e_1' \lesssim e_2' : \pi', \varphi' \cup \varphi''$ with $\mathit{Clean}(\varphi'')$. Also, we have that $\{\,\} \rhd e_2 \lesssim e_1 : \pi, \varphi$, so there exists an e_1'' such that $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1'' : \pi', \varphi')$ and $\{\,\} \rhd e_2' \lesssim e_1'' : \pi', \varphi' \cup \varphi''$. By Lemma 23, it must be that $e_1' = e_1''$, so $\{\,\} \rhd e_2' \lesssim e_1' : \pi', \varphi' \cup \varphi''$, which implies $\{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi, \varphi' \cup \varphi''$, showing that $\mathcal{R}$ is a simulation.

Mutual similarity allows the following practical formulation of bisimilarity: $\{\,\} \rhd e_1 \approx e_2 : \pi, \varphi$ iff $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi') \Leftrightarrow (e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$

where $\{\,\} \rhd e_1' \approx e_2' : \pi', \varphi' \cup \varphi''$ with *Clean*(φ''). Henceforth, we will assume that application of Proposition 8 is implicit.

When a converging expression is similar to an expression, then that expression must also converge:

Lemma 27. *Suppose* $TE \rhd e \lesssim^\circ e' : \pi, \varphi$. *If* $e \downarrow$ *then* $e' \downarrow$.

Proof. By open extension, we take ξ such that $TE \succ \xi$ and have $\{\,\} \rhd \xi(e) \lesssim \xi(e') : \pi, \varphi$. By Lemma 7 we now have that $\xi(e) \downarrow$, so using Lemma 9 and 20, we have a transition from $\xi(e)$. By the definition of similarity therefore also from $\xi(e')$. Lemma 9 and 21 prove the claim.

A well-typed clean diverging expression is immediately similar to any other well-typed clean expression.

Lemma 28. *Suppose* $\{\,\} \vdash_f e_1 : \pi, \varphi_1$ *and* $\{\,\} \vdash_f e_2 : \pi, \varphi_2$ *where* $\varphi_1 \cup \varphi_2 \subseteq \varphi$ *and* *Clean*(φ). *If* $e_1 \uparrow$ *then* $\{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi$.

Proof. Simple consequence of the definition of similarity and Lemma 21.

The left and right-hand side of a $\lambda^{\text{f-region}}$-reduction are bisimilar:

Lemma 29. *Suppose* $TE \vdash_f e : \pi, \varphi'$ *and* $e \rightharpoonup e'$ *where* $\varphi' \subseteq \varphi$ *and* *Clean*(φ). *Then* $TE \rhd e \approx^\circ e' : \pi, \varphi$.

Proof. Whenever the lemma holds for (bi)similarity, it trivially holds for the open extensions too, so define $\mathcal{R} = \{(\{\,\}, e, e', \pi, \varphi) \mid \{\,\} \vdash_f e : \pi, \varphi'$ *and* $e \rightharpoonup e'$ *where* $\varphi' \subseteq \varphi$ *with* *Clean*$(\varphi)\}$. By Proposition 1, we also have $\{\,\} \vdash_f e' : \pi, \varphi''$ with $\varphi'' \subseteq \varphi'$. Hence $\mathcal{R}$ is well-formed and we have to show that $\mathcal{R} \subseteq \approx$, where $\approx = [\approx] = \lceil \approx \rceil \cap \lceil \approx^{-1} \rceil^{-1}$. So, suppose $\{\,\} \rhd e \,\mathcal{R}\, e' : \pi, \varphi$.

First, we assume that $(e : \pi, \varphi) \xrightarrow{\gamma} (e_0 : \pi', \varphi')$. Since e cannot be a value, the transition must be due to rule *(Obs-red)* and therefore $(e' : \pi, \varphi) \xrightarrow{\gamma} (e_0 : \pi', \varphi')$. By Lemma 22, we have that $\{\,\} \vdash_f e_0 : \pi', \varphi'$ with *Clean*(φ') and hence by reflexivity of $\approx$, also that $\{\,\} \rhd e_0 \approx e_0 : \pi', \varphi' \cup \varphi''$ with *Clean*(φ''). Hence, $\mathcal{R} \subseteq \lceil \approx \rceil$.

The other way around, assume $(e' : \pi, \varphi) \xrightarrow{\gamma} (e_0' : \pi', \varphi')$. Then, immediately by rule *(Obs-red)*, we have that $(e : \pi, \varphi) \xrightarrow{\gamma} (e_0' : \pi', \varphi')$. Because of Lemma 22, we also have $\{\,\} \vdash_f e_0' : \pi', \varphi'$ with *Clean*(φ') and therefore by reflexivity that $\{\,\} \rhd e_0' \approx e_0' : \pi', \varphi' \cup \varphi''$ with *Clean*(φ''), implying $\mathcal{R} \subseteq \lceil \approx^{-1} \rceil^{-1}$.

The open extension of a simulation also has a type substitution property. We first show the property for simulation.

Lemma 30. *Suppose* $\{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi$ *where* *Clean*$(S_s(\varphi))$. *Then* $\{\,\} \rhd S_s(e_1) \lesssim S_s(e_2) : S_s(\pi), S_s(\varphi)$.

Proof. We prove by co-induction, so take $\mathcal{R} = \{(\{\,\}, S_s(e_1), S_s(e_2), S_s(\pi), S_s(\varphi)) \mid \{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi \textit{ with } \textit{Clean}(S_s(\varphi))\}$. By Lemma 1, we obviously have that $\mathcal{R}$ is well-formed, so we only need to prove that $\mathcal{R} \subseteq \lceil \mathcal{R} \cup \lesssim \rceil$.

Take $\{\,\} \rhd S_s(e_1)\, \mathcal{R}\, S_s(e_2) : S_s(\pi)$, and $(S_s(e_1) : S_s(\pi), S_s(\varphi)) \xrightarrow{\gamma'} (e_1' : \pi', \varphi')$. Now, by Lemma 26, this implies the transition

$$(e_1 : \pi, \varphi) \xrightarrow{\gamma''} (e_1'' : \pi'', \varphi'') \tag{15}$$

where $\gamma' = S_s^o(\pi, \gamma'')$, $e_1' = S_s^e(\gamma', e_1'')$, $\pi' = S_s^t(\gamma', \pi'')$ and $\varphi' = S_s^f(\gamma', \varphi'')$. So, we have the transition

$$(S_s(e_1) : S_s(\pi), S_s(\varphi)) \xrightarrow{\gamma'} (S_s^e(\gamma', e_1'') : S_s^t(\gamma', \pi''), S_s^f(\gamma', \varphi'')) \tag{16}$$

Since we are given that $\{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi$, by transition (15) we also have

$$(e_2 : \pi, \varphi) \xrightarrow{\gamma''} (e_2'' : \pi'', \varphi'') \tag{17}$$

and for arbitrary effect φ_0 with $\textit{Clean}(\varphi_0)$ that

$$\{\,\} \rhd e_1'' \lesssim e_2'' : \pi'', \varphi'' \cup \varphi_0 \tag{18}$$

Applying Lemma 25 on transition (17) yields

$$(S_s(e_2) : S_s(\pi), S_s(\varphi)) \xrightarrow{\gamma'} (S_s^e(\gamma', e_2'') : S_s^t(\gamma', \pi''), S_s^f(\gamma', \varphi''))$$

From this and the transition (16) it is sufficient to prove that

$$(\{\,\}, S_s^e(\gamma', e_1''), S_s^e(\gamma', e_2''), S_s^t(\gamma', \pi''), S_s^f(\gamma', \varphi'') \cup \varphi_0') \in (\mathcal{R} \cup \lesssim) \tag{19}$$

holds with $\textit{Clean}(\varphi_0')$. First, observe that the similarity (18) also holds for all φ_0'' such that $\textit{Clean}(\varphi_0'')$ and $\textit{Clean}(S_s(\varphi_0''))$. Since by general assumption we have that $\textit{Clean}(\textit{Supp}(S_s))$, we know that $\textit{Clean}(S_s(\varphi_0''))$ implies $\textit{Clean}(\varphi_0'')$, so we can take $\varphi_0' = S_s(\varphi_0'')$ to prove (19) without affecting generality.

Consider the case where we have $\textit{Deadly}(\gamma')$. Then, by reflexivity of $\lesssim$, we also have $\{\,\} \rhd S_s^e(\gamma', e_1'') \lesssim S_s^e(\gamma', e_2'') : S_s^t(\gamma', \pi''), S_s^f(\gamma', \varphi'') \cup \varphi_0'$. Whenever $\textit{Safe}(\gamma')$ holds, the similarity (18) and Lemma 22 imply that $\{\,\} \rhd S_s(e_1'')\, \mathcal{R}\, S_s(e_2'') : S_s(\pi''), S_s(\varphi'') \cup \varphi_0'$, which proves the case.

With this, we can prove the following type substitution property for open extensions of simulating expressions:

Proposition 9. *Suppose* $TE \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$ *with* $\textit{Clean}(S_s(\varphi))$. *Then* $S_s(TE) \rhd S_s(e_1) \lesssim^\circ S_s(e_2) : S_s(\pi), S_s(\varphi)$.

Proof. Take an ξ' such that $S_s(TE) \succ \xi'$. So, we have to show that $\{\,\} \rhd \xi'(S_s(e_1)) \lesssim \xi'(S_s(e_1)) : S_s(\pi), S_s(\varphi)$. Now, by Lemma 10, there exists an ξ such that $TE \succ \xi$ and $\xi' = S_s(\xi)$. Hence, it remains to be shown that $\{\,\} \rhd S_s(\xi(e_1)) \lesssim S_s(\xi(e_1)) : S_s(\pi), S_s(\varphi)$. However, since $TE \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$, it holds that $\{\,\} \rhd \xi(e_1) \lesssim \xi(e_1) : \pi, \varphi$. Since $\textit{Clean}(S_s(\varphi))$, we can apply Lemma 30, from which the claim follows.

7 Bisimilarity is Compatible

In this section, we prove that bisimilarity obeys a congruence property, more specifically, we show that the bisimilarity relation is compatible. We use a technique due to Howe [24, 25], which has been re-casted to typed functional languages by Gordon [16–18]. The key idea is to define the notion of *compatible extension*, a compatible relation which we prove to coincide with the open extension of similarity.

The compatible extension of $\lesssim$ is inductively defined as the least well-formed typed relation closed under the rule:

$$\textit{(Def-comp-ext)} \quad \frac{TE \rhd e_1 \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi \quad TE \rhd e_3 \mathrel{\lesssim^\circ} e_2 : \pi, \varphi}{TE \rhd e_1 \mathrel{\lesssim^*} e_2 : \pi, \varphi}$$

We have the following auxiliary result:

Lemma 31. *If $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ and $TE \rhd e_2 \lesssim^\circ e_3 : \pi, \varphi$, then $TE \rhd e_1 \lesssim^* e_3 : \pi, \varphi$*

Proof. By *(Def-comp-ext)* there exists an e_2' such that $TE \rhd e_1 \mathrel{\widehat{\lesssim^*}} e_2' : \pi, \varphi$ and $TE \rhd e_2' \lesssim^\circ e_2 : \pi, \varphi$. By transitivity of $\lesssim^\circ$, we have that $TE \rhd e_2' \lesssim^\circ e_3 : \pi, \varphi$. Hence, by *(Def-comp-ext)*, it follows that $TE \rhd e_1 \lesssim^* e_3 : \pi, \varphi$.

The compatible extension relation $\lesssim^*$ has the following basic properties:

Proposition 10. *1. $\lesssim^*$ is compatible*
2. $\lesssim^*$ is reflexive
3. $\lesssim^\circ \subseteq \lesssim^*$

Proof. 1. Take $TE \rhd e_1 \mathrel{\widehat{\lesssim^*}} e_2 : \pi, \varphi$. Since $\lesssim^*$ is well-formed by definition, we have by Lemma 13.2 that $\mathrel{\widehat{\lesssim^*}}$ is well-formed and hence, $TE \vdash_f e_2 : \pi, \varphi'$ with $\varphi' \subseteq \varphi$ and $\mathit{Clean}(\varphi)$. Since $\lesssim^\circ$ is reflexive, this gives us that $TE \rhd e_2 \lesssim^\circ e_2 : \pi, \varphi$. Therefore, by *(Def-comp-ext)*, we conclude that $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$.

2. Immediate by Lemma 13.1 and the previous item.

3. Assume $TE \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$. Since $\lesssim^\circ$ is well-formed, we have $TE \vdash_f e_2 : \pi, \varphi'$ with $\varphi' \subseteq \varphi$ and $\mathit{Clean}(\varphi)$. So by item 2 this implies that $TE \rhd e_2 \lesssim^* e_2 : \pi, \varphi$. Using Lemma 31 this yields $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$, proving the claim.

The compatible extension $\lesssim^*$ obeys a variable substitution lemma:

Lemma 32. *If $TE \rhd v_1 \lesssim^* v_2 : (\hat{\sigma}, \rho), \emptyset$ and $TE + \{y \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}, \rho)\} \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ with $\{\boldsymbol{\alpha}\} \cap \mathit{ftv}\,(TE) = \emptyset$, then $TE \rhd e_1[y \mapsto v_1] \lesssim^* e_2[y \mapsto v_2] : \pi, \varphi$.*

Proof. So we have $TE + \{y \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}, \rho)\} \rhd e_1 \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$ and $TE + \{y \mapsto (\forall\, \boldsymbol{\alpha}\, .\hat{\sigma}, \rho)\} \rhd e_3 \lesssim^\circ e_2 : \pi, \varphi$. By Lemma 12 we already have $TE \rhd e_3[y \mapsto v_2] \lesssim^\circ e_2[y \mapsto v_2] : \pi, \varphi$.

Using an inductive case analysis on the compatible refinement we prove that $TE \rhd e_1[y \mapsto v_1] \lesssim^* e_3[y \mapsto v_2] : \pi, \varphi$ and conclude the case by applying Lemma 31. The only interesting case is *(Comp-var)* where $x = y$ and we have to prove that $TE \rhd v_1 \lesssim^* v_2 : \pi, \varphi$. This, however, is given by assumption. All other cases are shown by application of the induction hypothesis and use of Proposition 10.1.

Additionally, $\lesssim^*$ obeys a type substitution lemma:

Lemma 33. *Suppose* $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ *with* $Clean(S_s(\varphi))$. *Then* $S_s(TE) \rhd S_s(e_1) \lesssim^* S_s(e_2) : S_s(\pi), S_s(\varphi)$.

Proof. So, we have by *(Def-comp-ext)* that

$$TE \rhd e_1 \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi \tag{20}$$

and $TE \rhd e_3 \lesssim^\circ e_2 : \pi, \varphi$. Applying Proposition 9 on the latter yields $S_s(TE) \rhd S_s(e_3) \lesssim^\circ S_s(e_2) : S_s(\pi), S_s(\varphi)$. It remains to show that $S_s(TE) \rhd S_s(e_1) \lesssim^* S_s(e_3) : S_s(\pi), S_s(\varphi)$, since we can then use Lemma 31 to obtain the desired result. Except for the cases *(Comp-lamv)* and *(Comp-lam)*, the claim is proven by a straightforward induction on the compatible refinement, using Proposition 10.1.

Suppose we have by *(Comp-lam)* (the case *(Comp-lamv)* is almost identical) that $TE \rhd \lambda\, x.e_1' \,\mathtt{at}\, \varrho \mathrel{\widehat{\lesssim^*}} \lambda\, x.e_3' \,\mathtt{at}\, \varrho : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \varrho), \varphi$. Hence, we also have $TE + \{x \mapsto \pi_1\} \rhd e_1' \lesssim^* e_3' : \pi_2, \varphi'$. Whenever we have that $Clean(S_s(\varphi'))$, we can use the induction argument and Proposition 10.1.

So, assume we have $Dirty(S_s(\varphi'))$. Since $\varrho \in \varphi$, we already have that $S_s(\varrho) \neq \bullet$. So, by Lemma 29, the transition rule *(Obs-dangerous)*, the fact that $\lesssim$ is a preorder and the definition of $\lesssim$, we also have that $S_s(TE) \rhd S_s(\lambda\, x.e_1' \,\mathtt{at}\, \varrho) \lesssim^\circ S_s(\lambda\, x.e_3' \,\mathtt{at}\, \varrho) : S_s(\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \varrho), S_s(\varphi)$. The case now follows from Proposition 10.3.

When a converging expression is $\lesssim^*$-related to another expression, then, that expression converges too.

Lemma 34. *Suppose* $TE \rhd e \lesssim^* e' : \pi, \varphi$. *If* $e \downarrow$ *then* $e' \downarrow$.

Proof. By *(Def-comp-ext)* we have $TE \rhd e \mathrel{\widehat{\lesssim^*}} e'' : \pi, \varphi$ and $TE \rhd e'' \lesssim^\circ e' : \pi, \varphi$. For the cases *(Comp-lam)*, *(Comp-lamv)*, *(Comp-fix)* and *(Comp-fixv)*, it is immediate that $e'' \downarrow$. For the other cases we apply the induction hypothesis to obtain $e'' \downarrow$ too. The lemma is now proven by Lemma 27.

To show that the open extension of similarity is compatible, we need an important auxiliary lemma:

Lemma 35. *Suppose* $\{\,\} \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ *and* $e_1 \rightharpoonup e_1'$. *Then* $\{\,\} \rhd e_1' \lesssim^* e_2 : \pi, \varphi$.

Proof. By rule induction on $e_1 \rightharpoonup e_1'$, where we always use *(Def-comp-ext)* on $\{\,\} \rhd e_1 \lesssim^* e_2 : \pi, \varphi$.

Case $\lambda\, x.e\,\mathtt{at}\,\varrho \rightharpoonup \langle \lambda\, x.e\rangle_\varrho$**.** So we have that $\{\,\} \rhd \lambda\, x.e\,\mathtt{at}\,\varrho \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$ and

$$\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi \tag{21}$$

By rule *(Comp-lam)* we have that $\pi = \pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2$ and $e_3 = \lambda\, x.e'\,\mathtt{at}\,\varrho$. Additionally we have $\varrho \in \varphi$ and $\{x \mapsto \pi_1\} \rhd e \lesssim^* e' : \pi_2, \varphi'$. By Lemma 29 the similarity $\{\,\} \rhd \langle \lambda\, x.e'\rangle_\varrho \lesssim \lambda\, x.e'\,\mathtt{at}\,\varrho : \pi, \varphi$ holds, and hence, by (21) and transitivity of $\lesssim$ also that

$$\{\,\} \rhd \langle \lambda\, x.e'\rangle_\varrho \lesssim e_2 : \pi, \varphi \tag{22}$$

Since the premises of *(Comp-lam)* and *(Comp-lamv)* are identical, we also have $\{\,\} \rhd \langle \lambda\, x.e\rangle_\varrho \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$, which combined with (22) and *(Def-comp-ext)* makes $\{\,\} \rhd \langle \lambda\, x.e\rangle_\varrho \lesssim^* e_2 : \pi, \varphi$.

Case $\mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\,\mathtt{at}\,\varrho \rightharpoonup \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho$**.** Analogous to the previous case.

Case $\Lambda\, \varrho.v \rightharpoonup v[\varrho \mapsto \bullet]$**.** Hence, $\{\,\} \rhd \Lambda\, \varrho.v \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$ and $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$. By rule *(Comp-reglam)*, it must be that $e_3 = \Lambda\, \varrho.e_3''$ where $\{\,\} \rhd v \lesssim^* e_3'' : \pi, \varphi$ and $\varrho \notin \mathit{frv}\,(\pi)$. Using Lemma 34, we have that $e_3'' \xrightarrow{*} v_3$ and by reduction rule (3) also that $\Lambda\, \varrho.e_3'' \xrightarrow{*} v_3[\varrho \mapsto \bullet]$. Hence, by Lemma 29, we have $\{\,\} \rhd e_3'' \lesssim v_3 : \pi, \varphi$ and $\{\,\} \rhd v_3[\varrho \mapsto \bullet] \lesssim e_3 : \pi, \varphi$. Transitivity of similarity on the one hand and Lemma 31 on the other make

$$\{\,\} \rhd v_3[\varrho \mapsto \bullet] \lesssim e_2 : \pi, \varphi \tag{23}$$

and $\{\,\} \rhd v \lesssim^* v_3 : \pi, \emptyset$. Applying Lemma 33 on the latter gives $\{\,\} \rhd v[\varrho \mapsto \bullet] \lesssim^* v_3[\varrho \mapsto \bullet] : \pi, \emptyset$. By well-formedness we also have $\{\,\} \rhd v[\varrho \mapsto \bullet] \lesssim^* v_3[\varrho \mapsto \bullet] : \pi, \varphi$ and hence, we can combine with (23) using Lemma 31 to obtain $\{\,\} \rhd v[\varrho \mapsto \bullet] \lesssim^* e_2 : \pi, \emptyset$.

Case $\langle \lambda\, x.e\rangle_\varrho @\, v \rightharpoonup e[x \mapsto v]$**.** We are given that $\{\,\} \rhd \langle \lambda\, x.e\rangle_\varrho @\, v \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$ and $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$. By *(Comp-app)* we must have that $e_3 = \langle \lambda\, x.e''\rangle_\varrho @\, e'$ where $\{\,\} \rhd v \lesssim^* e' : \pi', \varphi_1$ and

$$\{x \mapsto \pi'\} \rhd e \lesssim^* e'' : \pi, \varphi_2 \tag{24}$$

Using Lemma 34 it also holds that $e' \xrightarrow{*} v'$ and by Lemma 29 we have $\{\,\} \rhd e' \lesssim v' : \pi', \varphi_1$. From Lemma 31 this implies $\{\,\} \rhd v \lesssim^* v' : \pi', \varphi_1$. The latter, (24) and Lemma 32 imply

$$\{\,\} \rhd e[x \mapsto v] \lesssim^* e''[x \mapsto v'] : \pi, \varphi_2 \tag{25}$$

Since we have that $\langle \lambda\, x.e''\rangle_\varrho @\, e' \xrightarrow{*} e''[x \mapsto v']$, it also holds that $\{\,\} \rhd e''[x \mapsto v'] \lesssim e_3 : \pi, \varphi$ and hence, by transitivity of $\lesssim$ also $\{\,\} \rhd e''[x \mapsto v'] \lesssim e_2 : \pi, \varphi$. This, in combination with (25) and Lemma 31 proves the case.

Case $\mathtt{let}\, x = v\, \mathtt{in}\, e \rightharpoonup e[x \mapsto v]$**.** Analogous to the previous case.

Case $\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho\ [\rho'_1, \ldots, \rho'_n]\,\mathtt{at}\,\varrho' \rightharpoonup \langle \lambda\, x.e[\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n][f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho]\rangle_{\varrho'}$.

Now, we have that $\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho\ [\rho'_1, \ldots, \rho'_n]\,\mathtt{at}\,\varrho' \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$ and $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$ hold. Hence, by *(Comp-regapp)*, the only possible form for e_3 is $e'_3\ [\rho'_1, \ldots, \rho'_n]\,\mathtt{at}\,\varrho'$ where $\pi = (S_s(\tau), \varrho')$; $S_s = (I_t, S_e, \{\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n\})$ and $\hat{\sigma} = \forall\, \boldsymbol{\epsilon}.\forall\, \varrho_1, \ldots, \varrho_n.\tau$. Additionally, we also have $\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho \lesssim^* e'_3 : (\hat{\sigma}, \varrho), \varphi'$ with $\varphi' \cup \{\varrho, \varrho'\} \subseteq \varphi$.

Using Lemma 34, we have that $e'_3 \stackrel{*}{\rightharpoonup} v_3$ and by the canonical forms Lemma 5 also that v_3 has the form $\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\rangle_\varrho$. Using Lemma 29 and 31 we therefore have

$$\begin{array}{l}\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho \lesssim^* \\ \qquad \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\rangle_\varrho : (\hat{\sigma}, \varrho), \varphi'\end{array} \tag{26}$$

Since $e_3 \stackrel{*}{\rightharpoonup} \langle \lambda\, x.S_s(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\rangle_\varrho]\rangle_{\varrho'}$ too, we also obtain with Lemma 29 that

$$\{\,\} \rhd \langle \lambda\, x.S_s(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\rangle_\varrho]\rangle_{\varrho'} \lesssim e_3 : \pi, \varphi$$

Since $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$, by transitivity of $\lesssim$ this yields

$$\{\,\} \rhd \langle \lambda\, x.S_s(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\rangle_\varrho]\rangle_{\varrho'} \lesssim e_2 : \pi, \varphi \tag{27}$$

By *(Def-comp-ext)*, we have from (26) that

$$\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho \mathrel{\widehat{\lesssim^*}} e_4 : (\hat{\sigma}, \varrho), \varphi' \tag{28}$$

and

$$\{\,\} \rhd e_4 \lesssim \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\rangle_\varrho : (\hat{\sigma}, \varrho), \varphi' \tag{29}$$

From *(Comp-fixv)*, the congruence (28) implies that e_4 must be of the form $\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e''\rangle_\varrho$ with $\varrho \notin \{\varrho_1, \ldots, \varrho_n\}$ and also $\{f \mapsto (\hat{\sigma}, \varrho)\} \rhd \langle \lambda\, x.e\rangle_\varrho \lesssim^* \langle \lambda\, x.e''\rangle_\varrho : (\tau, \varrho), \varphi'$. The latter on its turn yields with *(Def-comp-ext)* and *(Comp-lamv)* that $\{f \mapsto (\hat{\sigma}, \varrho)\} \rhd \langle \lambda\, x.e\rangle_\varrho \mathrel{\widehat{\lesssim^*}} \langle \lambda\, x.e'''\rangle_\varrho : (\tau, \varrho), \varphi'$ and $\{f \mapsto (\hat{\sigma}, \varrho)\} \rhd \langle \lambda\, x.e'''\rangle_\varrho \lesssim^\circ \langle \lambda\, x.e''\rangle_\varrho : (\tau, \varrho), \varphi'$. The compatibility rule *(Comp-lamv)*, the definition of open extension, the transition rule *(Obs-fun)* and the type rule *(Lamv)* make that $\{f \mapsto (\hat{\sigma}, \varrho)\} \rhd \langle \lambda\, x.e\rangle_{\varrho'} \mathrel{\widehat{\lesssim^*}} \langle \lambda\, x.e'''\rangle_{\varrho'} : (\tau, \varrho'), \varphi'$ and $\{f \mapsto (\hat{\sigma}, \varrho)\} \rhd \langle \lambda\, x.e'''\rangle_{\varrho'} \lesssim^\circ \langle \lambda\, x.e''\rangle_{\varrho'} : (\tau, \varrho'), \varphi'$ also holds. Recombining with *(Def-comp-ext)* yields $\{f \mapsto (\hat{\sigma}, \varrho)\} \rhd \langle \lambda\, x.e\rangle_{\varrho'} \lesssim^* \langle \lambda\, x.e''\rangle_{\varrho'} : (\tau, \varrho'), \varphi'$. Using Lemma 33 with S_s and Lemma 32 with (28) on this assertion, we obtain

$$\begin{array}{l}\{\,\} \rhd \langle \lambda\, x.S_s(e)[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e\rangle_\varrho]\rangle_{\varrho'} \lesssim^* \\ \qquad \langle \lambda\, x.S_s(e'')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e''\rangle_\varrho]\rangle_{\varrho'} : \pi, \emptyset\end{array} \tag{30}$$

But we also have by (29) and the definition of similarity (using *(Obs-fix)*) that

$$\begin{array}{l}\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e''\rangle_\varrho\ [\varrho_1, \ldots, \varrho_n]\,\mathtt{at}\,\varrho \lesssim \\ \qquad \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'\rangle_\varrho\ [\varrho_1, \ldots, \varrho_n]\,\mathtt{at}\,\varrho : (\tau, \varrho), \emptyset\end{array}$$

Using Lemma 29, transitivity of $\lesssim$ and Proposition 9 with S_s, we obtain

$$\{\,\} \rhd \langle \lambda\, x.S_s(e'')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'' \rangle_\varrho] \rangle_\varrho \lesssim \langle \lambda\, x.S_s(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e' \rangle_\varrho] \rangle_\varrho : (S_s(\tau), \varrho), \emptyset$$

By transition rule *(Obs-fun)*, type rule *(Lamv)* and well-formedness, we therefore also have:

$$\{\,\} \rhd \langle \lambda\, x.S_s(e'')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e'' \rangle_\varrho] \rangle_{\varrho'} \lesssim \langle \lambda\, x.S_s(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e' \rangle_\varrho] \rangle_{\varrho'} : \pi, \varphi$$

In combination with (30) via Lemma 31 this yields

$$\{\,\} \rhd \langle \lambda\, x.S_s(e)[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e \rangle_\varrho] \rangle_{\varrho'} \lesssim^* \langle \lambda\, x.S_s(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e' \rangle_\varrho] \rangle_{\varrho'} : \pi, \varphi$$

and, again combining with (27) using Lemma 31, we prove the case.

Case $e @ e'' \rightharpoonup e' @ e''$ where $e \rightharpoonup e'$**.** We have $\{\,\} \rhd e @ e'' \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$ and

$$\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi \tag{31}$$

By *(Comp-app)*, e_3 must be of the form $e_0 @ e_0''$ where $\{\,\} \rhd e \lesssim^* e_0 : (\pi' \xrightarrow{\epsilon.\varphi} \pi, \rho), \varphi_1$ and $\{\,\} \rhd e'' \lesssim^* e_0'' : \pi', \varphi_2$. Because of the induction hypothesis, we also have $\{\,\} \rhd e' \lesssim^* e_0 : (\pi' \xrightarrow{\epsilon.\varphi} \pi, \rho), \varphi_1$. Again, by rule *(Comp-app)*, we have that $\{\,\} \rhd e' @ e'' \mathrel{\widehat{\lesssim^*}} e_0 @ e_0'' : \pi, \varphi$ and combining this with (31) and *(Def-comp-ext)* the case follows.

The remaining context cases are all analogous to the case for $e @ e'' \rightharpoonup e' @ e''$.

With this lemma, the following inclusion result can be proven:

Lemma 36. *If* $\{\,\} \rhd e_1 \lesssim^* e_2 : \pi, \varphi$*, then* $\{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi$.

Proof. We prove by co-induction, hence, define $\mathcal{R} = \{(\{\,\}, e_1, e_2, \pi, \varphi) \mid \{\,\} \rhd e_1 \lesssim^* e_2 : \pi, \varphi\}$. Obviously, $\mathcal{R}$ is well-formed since $\lesssim^*$ is, so it is sufficient to prove that $\mathcal{R} \subseteq \lceil \mathcal{R} \cup \lesssim \rceil$. Take $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ and $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$.

We proceed by an inductive case analysis on the labeled transition system:

Case *(Obs-red)***.** So we have that $\{\,\} \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ and since $e_1 \rightharpoonup e_1''$ holds we can use Lemma 35 to obtain $\{\,\} \rhd e_1'' \lesssim^* e_2 : \pi, \varphi$. Hence, $\{\,\} \rhd e_1'' \,\mathcal{R}\, e_2 : \pi, \varphi$ and so, by induction, there is a transition $(e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$ such that $(\{\,\}, e_1', e_2', \pi', \varphi' \cup \varphi'') \in \mathcal{R} \cup \lesssim$ with $\mathit{Clean}(\varphi'')$, proving the case.

Case *(Obs-fun)***.** So we have that $e_1 = \langle \lambda\, x.e_1'' \rangle_\varrho$, the type $\pi = (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \varrho)$ with $\mathit{Clean}(\varphi')$, the observation $\gamma = @\, v$, $\mathit{frv}\,(v) \subseteq \mathit{frv}\,(\pi)$ and $e_1' = \langle \lambda\, x.e_1'' \rangle_\varrho @\, v$. But we also have $\{\,\} \rhd \langle \lambda\, x.e_1'' \rangle_\varrho \lesssim^* e_2 : \pi, \varphi$ and therefore, by *(Def-comp-ext)*, this implies $\{\,\} \rhd \langle \lambda\, x.e_1'' \rangle_\varrho \mathrel{\widehat{\lesssim^*}} \langle \lambda\, x.e_3'' \rangle_\varrho : \pi, \varphi$ and $\{\,\} \rhd \langle \lambda\, x.e_3'' \rangle_\varrho \lesssim e_2 : \pi, \varphi$. Hence, by Proposition 10.1, we also have $\{\,\} \rhd \langle \lambda\, x.e_1'' \rangle_\varrho \lesssim^*$

$\langle \lambda\, x.e_3'' \rangle_\varrho : \pi, \varphi$. Also, by Proposition 10.2, we have $\{\,\} \rhd v \lesssim^* v : \pi_1, \emptyset$. Combining this with *(Comp-app)* we obtain that

$$\{\,\} \rhd \langle \lambda\, x.e_1'' \rangle_\varrho @\, v \mathrel{\widehat{\lesssim^*}} \langle \lambda\, x.e_3'' \rangle_\varrho @\, v : \pi_2', \varphi'' \tag{32}$$

where $\varphi' \cup \{\epsilon, \varrho\} \subseteq \varphi''$ and $\mathit{Clean}(\varphi'')$. From Lemma 24, the transition $(\langle \lambda\, x.e_3'' \rangle_\varrho : \pi, \varphi) \xrightarrow{@\, v} (\langle \lambda\, x.e_3'' \rangle_\varrho @\, v : \pi_2', \varphi'')$ exists. Since $\{\,\} \rhd \langle \lambda\, x.e_3'' \rangle_\varrho \lesssim e_2 : \pi, \varphi$ holds and by the definition of $\lesssim$, we also have $\{\,\} \rhd \langle \lambda\, x.e_3'' \rangle_\varrho @\, v \lesssim e_2 @\, v : \pi_2', \varphi''$. This, together with (32) and *(Def-comp-ext)* proves the case, where we note that $\lesssim^*$ is already well-formed.

Case *(Obs-fix)*. We have that $e_1 = \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_1'' \rangle_\varrho$ and the type π is $(\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \varrho)$. So, the observation γ is $[\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho'$ and $e_1' = e_1\ [\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho'$. The type π' is $(S_s(\tau), \varrho')$ where we assume that the substitution $S_s = (I_t, S_e, \{\varrho_1 \mapsto \rho_1', \ldots, \varrho_n \mapsto \rho_n'\})$ such that $\mathit{Supp}(S_e) \subseteq \{\boldsymbol{\epsilon}\}$ and $\{\rho_1', \ldots, \rho_n', \varrho'\} \subseteq \mathit{frv}\,(\pi)$.
However, since we have $\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_1'' \rangle_\varrho \lesssim^* e_2 : \pi, \varphi$, by *(Def-comp-ext)*, we also have $\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_1'' \rangle_\varrho \mathrel{\widehat{\lesssim^*}} e_3 : \pi, \varphi$ and $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$ where $e_3 = \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_3'' \rangle_\varrho$. Due to Proposition 10.1 and *(Comp-regapp)* we have that

$$\begin{aligned} \{\,\} \rhd\ & \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_1'' \rangle_\varrho\ [\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho' \lesssim^* \\ & \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_3'' \rangle_\varrho\ [\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho' : (S_s(\tau), \varrho'), \varphi'' \end{aligned} \tag{33}$$

where $\{\varrho, \varrho'\} \subseteq \varphi''$ and $\mathit{Clean}(\varphi'')$. Since the transition

$$\begin{aligned} & (\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_3'' \rangle_\varrho : (\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \varrho), \varphi) \xrightarrow{\gamma} \\ & \quad (\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.e_3'' \rangle_\varrho\ [\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho' : (S_s(\tau), \varrho'), \varphi'') \end{aligned}$$

is possible by Lemma 24 and $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$ holds too, we have by definition of $\lesssim$ that

$$(e_2 : (\forall\, \boldsymbol{\epsilon}\, .\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \varrho), \varphi) \xrightarrow{\gamma} (e_2\ [\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho' : (S_s(\tau), \varrho'), \varphi'')$$

and $\{\,\} \rhd e_3\ [\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho' \lesssim e_2' : (S_s(\tau), \varrho'), \varphi''$ where $e_2' = e_2\ [\rho_1', \ldots, \rho_n']\, \mathtt{at}\, \varrho'$. Combining this with (33) and Lemma 31 proves that $\{\,\} \rhd e_1' \lesssim^* e_2' : \pi', \varphi''$ and hence the case.

Cases *(Obs-deallocated)*, *(Obs-dangerous)* and *(Obs-deadalloc)*. Because of Lemma 21 we have that $e_1 \downarrow$ and hence by Lemma 34 also $e_2 \downarrow$. The transition from e_1 goes to the expression $\Omega^\alpha_{\varrho_\star}$ and by Lemma 24 we therefore have a transition from e_2 to $\Omega^\alpha_{\varrho_\star}$ as well. Reflexivity of $\lesssim$ now proves the case.

The actual theorem now follows easily:

Theorem 2. *$\approx^\circ$ is compatible.*

Proof. We already have by Proposition 10.3 that $\lesssim^\circ \subseteq \lesssim^*$. For the other direction, assume $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ and take a ξ for which $TE \succ \xi$. Now, by Proposition 10.2 and Lemma 32, it is easy to prove that $\{\,\} \rhd \xi(e_1) \lesssim^* \xi(e_2) : \pi, \varphi$. So, from Lemma 36 we obtain $\{\,\} \rhd \xi(e_1) \lesssim \xi(e_2) : \pi, \varphi$, which by definition of completion implies $TE \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$.

Compatibility of $\approx^\circ$ follows immediately by Proposition 8.

8 The Equational Theory is a Bisimulation

Bisimulation provides a strong proof mechanism, which we now use to prove that the equational theory of section 5 is included in bisimilarity. To do so, we need one main auxiliary result which shows that a semantically dead region can be replaced by *the* dead region $\bullet$.

Lemma 37. *Suppose* $\{\,\} \vdash_f e : \pi, \varphi'$ *where* $\varphi' \subseteq \varphi$ *with* $\mathit{Clean}(\varphi)$ *and* $\varrho \notin \mathit{frv}(\pi) \cup \varphi$, *then* $\{\,\} \rhd e \approx e[\varrho \mapsto \bullet] : \pi, \varphi$.

Proof. Define $\mathcal{R} = \{(\{\,\}, e, e[\varrho \mapsto \bullet], \pi, \varphi) \mid \{\,\} \vdash_f e : \pi, \varphi' \text{ such that } \varphi' \subseteq \varphi \text{ with } \mathit{Clean}(\varphi) \text{ and } \varrho \notin \mathit{frv}(\pi) \cup \varphi\}$. First, observe by Lemma 1 that $\{\,\} \vdash_f e[\varrho \mapsto \bullet] : \pi, \varphi'$ and hence, $\mathcal{R}$ is well-formed. So, by co-induction we have to show that $\mathcal{R} \subseteq [\mathcal{R} \cup \approx]$ or $\mathcal{R} \subseteq \lceil \mathcal{R} \cup \approx \rceil \cap \lceil \mathcal{R}^{-1} \cup \approx \rceil^{-1}$. We show that $\mathcal{R} \subseteq \lceil \mathcal{R} \cup \approx \rceil$, the other case is by similar reasoning. Take $\{\,\} \rhd e \,\mathcal{R}\, e[\varrho \mapsto \bullet] : \pi, \varphi$ and a transition $(e : \pi, \varphi) \xrightarrow{\gamma} (e_0 : \pi_0, \varphi_0)$. From Lemma 21 we have that $e \downarrow$ and since $\varrho \notin \varphi$ we also have that $e[\varrho \mapsto \bullet] \downarrow$ by Lemma 8. Hence, by Lemma 20 we have $(e[\varrho \mapsto \bullet] : \pi, \varphi) \xrightarrow{\gamma} (e_0' : \pi_0, \varphi_0)$.

Whenever $\mathit{Deadly}(\gamma)$, then trivially $\{\,\} \rhd e_0 \lesssim e_0' : \pi_0, \varphi_0 \cup \varphi''$ for $\mathit{Clean}(\varphi'')$ by reflexivity of $\approx^\circ$. So, assume $\mathit{Safe}(\gamma)$. Then we have $e \xrightarrow{*} v$ and $e[\varrho \mapsto \bullet] \xrightarrow{*} v[\varrho \mapsto \bullet]$ where $e_0 = v\gamma$ and $e_0' = (v[\varrho \mapsto \bullet])\gamma$.

We first consider the case that γ is due to rule *(Obs-fun)*. Then $\gamma = @\,v'$ and we have that $\mathit{frv}(@\,v') \subseteq \mathit{frv}(\pi)$ and also that $\mathit{frv}(\pi_0, \varphi_0) \subseteq \mathit{frv}(\pi)$. Therefore, since $\varrho \notin \pi$, we have that $\{\,\} \rhd v@\,v' \,\mathcal{R}\, (v@\,v')[\varrho \mapsto \bullet] : \pi_0, \varphi_0 \cup \varphi''$ with $\mathit{Clean}(\varphi'')$.

Suppose that $\gamma = [\rho_1', \ldots, \rho_n']\,\mathtt{at}\,\varrho'$ is due to rule *(Obs-fix)*. Then, the value v has the form $\langle \mathtt{fix} f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda x.e \rangle_{\varrho_0}$ and we can assume by α-renaming that $\varrho \notin \{\varrho_1, \ldots, \varrho_n\}$. By inspection of the rule *(Obs-fix)* and since $\varrho \notin \mathit{frv}(\pi)$, we obtain that $e_0' = (v[\rho_1', \ldots, \rho_n']\,\mathtt{at}\,\varrho')[\varrho \mapsto \bullet]$ and $\{\,\} \rhd e_0 \,\mathcal{R}\, e_0' : \pi_0, \varphi_0 \cup \varphi''$ with $\mathit{Clean}(\varphi'')$. This proves the lemma.

Now, we can prove the main theorem:

Theorem 3. $\mathrel{\dot{=}} \subseteq \approx^\circ$.

Proof. We have to show that $\approx^\circ$ satisfies the rules in Figure 6. Then, because $\mathrel{\dot{=}}$ is the least relation satisfying these rules, we automatically have $\mathrel{\dot{=}} \subseteq \approx^\circ$. Observe that by Lemma 18, we already have that $\mathrel{\dot{=}}$ is well-formed.

(Eq-comp). Immediate by Theorem 2.
(Eq-symm) and *(Eq-trans).* Immediate from Proposition 7.2.
(Eq-beta-n) with $n \in \{1, 2, 5, 6\}$. Suppose $TE \vdash_f e : \pi, \varphi'$ where $\varphi' \subseteq \varphi$ with $\mathit{Clean}(\varphi)$ and $e \overset{n}{\rightarrowtail} e'$. For arbitrary ξ where $TE \succ \xi$, we have by Lemma 9 that $\{\,\} \vdash_f \xi(e) : \pi, \varphi'$. Also, by Lemma 7, we have that $\xi(e) \overset{n}{\rightarrowtail} \xi(e')$. So, by Lemma 29, it holds that $\{\,\} \rhd \xi(e) \approx \xi(e') : \pi, \varphi$ and this for arbitrary ξ with $TE \succ \xi$. Hence, by definition of open extension, we obtain that $TE \rhd e \approx^\circ e' : \pi, \varphi$.

(Eq-app-cbv). Given that $TE \vdash_f \langle \lambda\, x.e_1 \rangle_\varrho : (\pi_1 \xrightarrow{\epsilon.\varphi''} \pi, \varrho), \emptyset$ and $TE \vdash_f e_2 : \pi_1, \varphi'$ with $\varphi' \cup \varphi'' \cup \{\epsilon, \varrho\} \subseteq \varphi$ and $\mathit{Clean}(\varphi)$, we have to prove that $TE \rhd \langle \lambda\, x.e_1 \rangle_\varrho @\, e_2 \approx^\circ \mathtt{let}\ x = e_2\ \mathtt{in}\ e_1 : \pi, \varphi$. Without loss of generality, by α-renaming, let us assume that $x \notin \mathit{Dom}(TE)$. By definition of completion, this implies that x does not occur free in ξ either, where we take ξ such that $TE \succ \xi$. We have to show $\{\,\} \rhd \langle \lambda\, x.\xi(e_1) \rangle_\varrho @\, \xi(e_2) \approx \mathtt{let} x = \xi(e_2) \mathtt{in} \xi(e_1) : \pi, \varphi$.

First, we can easily see that if $\langle \lambda\, x.\xi(e_1) \rangle_\varrho @\, \xi(e_2) \downarrow$, then $(\mathtt{let} x = \xi(e_2) \mathtt{in} \xi(e_1)) \downarrow$ and the other way around. So, by Lemma 28, we have that the two terms are bisimilar.

Suppose on the other hand that $\langle \lambda\, x.\xi(e_1) \rangle_\varrho @\, \xi(e_2) \xrightarrow{*} \langle \lambda\, x.\xi(e_1) \rangle_\varrho @\, v_2 \xrightarrow{*} \xi(e_1)[x \mapsto v_2]$ where we assume $\xi(e_2) \xrightarrow{*} v_2$. Then, by Lemma 29, we obtain $\{\,\} \rhd \xi(e_1) @\, \xi(e_2) \approx \xi(e_1)[x \mapsto v_2] : \pi, \varphi$. We also have $\mathtt{let}\ x = \xi(e_2)\ \mathtt{in}\ \xi(e_1) \xrightarrow{*} \mathtt{let}\ x = v_2\ \mathtt{in}\ \xi(e_1) \xrightarrow{*} \xi(e_1)[x \mapsto v_2]$, so again by Lemma 29 we obtain $\{\,\} \rhd \mathtt{let}\ x = \xi(e_2)\ \mathtt{in}\ \xi(e_1) \approx \xi(e_1)[x \mapsto v_2] : \pi, \varphi$. Bisimilarity of the two terms now follows by transitivity and symmetry of the relation.

The fact that rule *(Eq-app-let)* is included in bisimilarity, is a consequence of a result for the weaker notion of *Kleene-equivalence* [17,38]: Given $\{\,\} \vdash_f e_1 : \pi, \varphi_1$ and $\{\,\} \vdash_f e_2 : \pi, \varphi_2$ with $\mathit{Clean}(\varphi_1 \cup \varphi_2)$, then e_1 is Kleene-equivalent to e_2 iff there exists a v such that $e_1 \xrightarrow{*} v$ and $e_2 \xrightarrow{*} v$. By similar reasoning as above, it is easy to prove that Kleene-equivalent terms are always bisimilar.

(Eq-app-let) and *(Eq-let-let).* It is easy to see that the two sides both the rules *(Eq-app-let)* and *(Eq-let-let)* are Kleene-equivalent after performing an arbitrary completion. So, analogous to the previous case, we can show that they are bisimilar.

(Eq-drop). To prove the two terms from *(Eq-drop)* bisimilar under open extension, assume an iterated substitution ξ with $TE \succ \xi$. We have to show that $\{\,\} \rhd \xi(e) \approx \Lambda\, \varrho.\xi(e) : \pi, \varphi$.

We obviously have that whenever $\xi(e) \downarrow$, also $\Lambda\, \varrho.\xi(e) \downarrow$ and the other way around. So by Lemma 28 the terms are bisimilar.

Otherwise, assume that $\xi(e) \xrightarrow{*} v$ and hence also $\Lambda\, \varrho.\xi(e) \xrightarrow{*} v[\varrho \mapsto \bullet]$. Using Lemma 29, these two reductions and the premises of *(Eq-drop)* make that $\{\,\} \rhd \xi(e) \approx v : \pi, \varphi$ and $\{\,\} \rhd \Lambda\, \varrho.\xi(e) \approx v[\varrho \mapsto \bullet] : \pi, \varphi$. Also, by Lemma 37, we have the bisimilarity $\{\,\} \rhd v \approx v[\varrho \mapsto \bullet] : \pi, \varphi$. Using Proposition 7.2, we can combine these to obtain $\{\,\} \rhd \xi(e) \approx \Lambda\, \varrho.\xi(e) : \pi, \varphi$, proving the case.

(Eq-dealloc). Immediate by Lemma 37.

9 Bisimilarity equals Contextual Equivalence

Now that we have shown that the equational theory $\doteq$ is included in the bisimilarity relation $\approx$, we prove that the latter is equivalent to contextual equivalence $\simeq$. As a corollary, we derive that the region theory is sound with respect to contextual equivalence.

First, we observe that bisimilarity is adequate:

Proposition 11. $\approx^\circ$ *is well-formed, compatible and adequate.*

Proof. From Proposition 7.3 we have that $\approx^\circ$ is well-formed and by Theorem 2, we also know it is compatible. Adequacy follows from Lemma 27.

We can now prove a context lemma, but instead of taking evaluation contexts, we only consider a sequence of safe observations:

Lemma 38. $\{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi$ *iff* $\{\,\} \vdash_f e_1 : \pi, \varphi_1$ *and* $\{\,\} \vdash_f e_2 : \pi, \varphi_2$ *where* $\varphi_1 \cup \varphi_2 \subseteq \varphi$ *with* $Clean(\varphi)$ *and for all observations* $\gamma_1, \ldots, \gamma_n$ *with* $Safe(\gamma_i)$*, it holds that* $e_1\gamma_1 \ldots \gamma_n \downarrow \Rightarrow e_2\gamma_1 \ldots \gamma_n \downarrow$.

Proof. **Left-to-right** Take a sequence of observations $\gamma_1, \ldots, \gamma_n$ with $Safe(\gamma_i)$. Since $e_1\gamma_1 \ldots \gamma_n \downarrow$, we have by reduction rules (8) and (11) the following reductions: $e_1 \xrightarrow{*} v_1^1$; $v_1^1\gamma_1 \xrightarrow{*} v_1^2$; $v_1^2\gamma_2 \xrightarrow{*} v_1^3$; ... and $v_1^n\gamma_n \downarrow$. By Lemma 20 and Proposition 1, this implies the existence of the transition $(e_1 : \pi, \varphi) \xrightarrow{\gamma_1} (v_1^1\gamma_1 : \pi^1, \varphi^1)$ and for $i \in \{1, \ldots, n-1\}$ also the transitions $(v_1^i\gamma_i : \pi^i, \varphi^i) \xrightarrow{\gamma_{i+1}} (v_1^{i+1}\gamma_{i+1} : \pi^{i+1}, \varphi^{i+1})$. Hence, by the similarity, we also have the transition $(e_2 : \pi, \varphi) \xrightarrow{\gamma_1} (v_2^1\gamma_1 : \pi^1, \varphi^1)$ and for $i \in \{1, \ldots, n-1\}$ the transitions $(v_2^i\gamma_i : \pi^i, \varphi^i) \xrightarrow{\gamma_{i+1}} (v_2^{i+1}\gamma_{i+1} : \pi^{i+1}, \varphi^{i+1})$ such that $\{\,\} \rhd v_1^i\gamma_i \lesssim v_2^i\gamma_i : \pi^i, \varphi^i$ and also $\{\,\} \rhd v_1^n\gamma_n \lesssim v_2^n\gamma_n : \pi^n, \varphi^n$. From the latter and since $v_1^n\gamma_n \downarrow$, we have by Lemma 20, that $v_2^n\gamma_n \downarrow$ too. Applying Lemma 21 subsequently on the transitions from $v_2^i\gamma_i$, makes that $e_2 \xrightarrow{*} v_2^1$; $v_2^1\gamma_1 \xrightarrow{*} v_2^2$; $v_2^2\gamma_2 \xrightarrow{*} v_2^3$; ... too. Therefore we have $e_2\gamma_1 \ldots \gamma_n \downarrow$, which proves the case.

Right-to-left We prove by co-induction, so define the set

$$\begin{aligned}\mathcal{R} = \{(\{\,\}, e_1, e_2, \pi, \varphi) \mid \{\,\} \vdash_f e_1 : \pi, \varphi_1 \textit{ and } \{\,\} \vdash_f e_2 : \pi, \varphi_2 \textit{ where} \\ \varphi_1 \cup \varphi_2 \subseteq \varphi \textit{ with } Clean(\varphi) \textit{ and for all } \gamma_1, \ldots, \gamma_n \textit{ with } Safe(\gamma_i) \\ \textit{it holds that } e_1\gamma_1 \ldots \gamma_n \downarrow \Rightarrow e_2\gamma_1 \ldots \gamma_n \downarrow\}\end{aligned}$$

Since $\mathcal{R}$ is well-formed, it is sufficient to show that $\mathcal{R} \subseteq \lceil \mathcal{R} \cup \lesssim \rceil$. Take $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ and suppose $(e_1 : \pi, \varphi) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$. This implies that $e_1 \downarrow$ and since $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ we therefore have $e_2 \downarrow$ (where we take the empty sequence of observations). By Lemma 24 the transition $(e_2 : \pi, \varphi) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$ exists.

Whenever $Deadly(\gamma)$, then the claim is proven by reflexivity of $\lesssim$. Therefore, assume $Safe(\gamma)$. So, we also have that $e_1 \xrightarrow{*} v_1$ with $e_1' = v_1\gamma_1$ and $e_2 \xrightarrow{*} v_2$ with $e_2' = v_2\gamma$.

Now, take an arbitrary sequence of observations $\gamma_1, \ldots, \gamma_n$ with $Safe(\gamma_i)$ and assume that $v_1\gamma\gamma_1 \ldots \gamma_n \downarrow$. Since $e_1 \xrightarrow{*} v_1$, we have because of the reductions (8) and (11) that $e_1\gamma\gamma_1 \ldots \gamma_n \downarrow$. Because $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$, this implies $e_2\gamma\gamma_1 \ldots \gamma_n \downarrow$. But, we also have $e_2 \xrightarrow{*} v_2$ and again using the reductions (8) and (11) that $v_2\gamma\gamma_1 \ldots \gamma_n \downarrow$. It now follows that $\{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi'$.

By definition of $\mathcal{R}$, we also have $\{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi' \cup \varphi''$ with $\mathit{Clean}(\varphi'')$, proving the claim.

Proposition 12. *If* $\{\,\} \rhd e_1 \simeq e_2 : \pi, \varphi$, *then* $\{\,\} \rhd e_1 \approx e_2 : \pi, \varphi$.

Proof. Take $\{\,\} \rhd e_1 \simeq e_2 : \pi, \varphi$ and hence, by definition, we have that $\{\,\} \vdash_f e_1 : \pi, \varphi_1$ and $\{\,\} \vdash_f e_2 : \pi, \varphi_2$ with $\varphi_1 \cup \varphi_2 \subseteq \varphi$ and $\mathit{Clean}(\varphi)$. Consecutively, take a sequence of observations $\gamma_1, \ldots, \gamma_n$ with $\mathit{Safe}(\gamma_i)$. By rules *(Comp-app)* and *(Comp-regapp)*, we can derive that $\{\,\} \rhd e_1\gamma_1 \ldots \gamma_n \{(\{\,\}, e_1, e_2, \pi, \varphi)\}^c\, e_2\gamma_1 \ldots \gamma_n : \pi^n, \varphi^n$. Since contextual equivalence makes $\{(\{\,\}, e_1, e_2, \pi, \varphi)\}^c$ adequate, we have $e_1\gamma_1 \ldots \gamma_n \lfloor \Leftrightarrow e_2\gamma_1 \ldots \gamma_n \rfloor$. Thus, by the context Lemma 38, we conclude that $\{\,\} \rhd e_1 \approx e_2 : \pi, \varphi$.

Our notion of applicative bisimilarity functions as an alternative characterization of contextual equivalence.

Theorem 4. $\approx^\circ = \simeq$

Proof. As an immediate consequence of Proposition 6 and 11, we have that $\approx^\circ \subseteq \simeq$. Remains to show that $\simeq \supseteq \approx^\circ$. Suppose $TE \rhd e_1 \simeq e_2 : \pi, \varphi$ and an iterated substitution ξ for which $TE \succ \xi$. To prove that $TE \rhd e_1 \approx^\circ e_2 : \pi, \varphi$, we have to show that $\{\,\} \rhd \xi(e_1) \approx \xi(e_2) : \pi, \varphi$.
By Proposition 6 and Lemma 14, we have that $\{\,\} \rhd \mathcal{T}(\xi, e_1) \simeq \mathcal{T}(\xi, e_2) : \pi, \varphi$. Also, by Theorem 3 and Lemma 19 we have that $\{\,\} \rhd \mathcal{T}(\xi, e_1) \simeq \xi(e_2) : \pi, \varphi$ and $\{\,\} \rhd \xi(e_1) \simeq \mathcal{T}(\xi, e_2) : \pi, \varphi$. So, by transitivity of $\simeq$, this implies that $\{\,\} \rhd \xi(e_1) \simeq \xi(e_2) : \pi, \varphi$ and hence, by Proposition 12, it follows that $\{\,\} \rhd \xi(e_1) \approx \xi(e_2) : \pi, \varphi$.

10 Conclusions and Future Work

We have defined a typed equational theory for a region calculus in the style of Tofte and Talpin [51]. By introducing an appropriate notion of bisimilarity based on a labeled transition system, we have shown that the theory is sound with respect to contextual equivalence, thereby providing an semantic framework to reason about region calculus transformations. The equational theory is used in other work to prove specialization soundness of offline partial evaluation for ML-like languages [22].

The proof methods in this paper are partially recycled from Gordon's work on bisimilarity in functional programming languages [17, 18]. Bisimilarity turns out to be a grateful proof mechanism for showing contextual equivalences in deterministic stateless program calculi. The addition of type, effect and region polymorphism does not hinder the use of bisimulation techniques.

Indirectly, the results of this paper suggest that region inference is *correct* in the sense that a region-annotated expression is semantically equivalent to the unannotated version. This can be understood as follows: for any region annotated program we use the equational theory to show that it is contextual equivalent to the program with only one global region and no local region allocations. We

conjecture that it is straightforward to show an equivalence (in some sense to be defined) between the program with the one global region and its unannotated version. Hence, our equational theory provides an alternative soundness proof for the translation correctness of region inference [51]. In that paper, Tofte and Talpin actually combine translation and type soundness leading to a monolithic, complex and un-intuitive proof. Perhaps more importantly, the equational theory explicitely shows that region operations do not affect the value semantics of an expression. In other words, region memory management is independent of the actual program semantics.

A challenging question for future work is to figure out whether bisimilarity is transferable to a region calculus with an explicit store and local state. In previous work [8], we already examined type-soundness results for a store-passing reduction semantics of the Tofte-Talpin region calculus, including operations for destructive updates. However, it is well-known that reasoning about contextual equivalences for stateful calculi is hard [39, 45].

Acknowledgments The author wishes to thank Silvano Dal Zilio, Andy Gordon, Matthias Neubauer and Peter Thiemann for suggestions, discussions and comments during the preparation of the draft.

References

1. Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, chapter 4, pages 65–116. Addison-Wesley, 1990.
2. Samson Abramsky and Luke Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267, 1993.
3. Alexander Aiken, Manuel Fähndrich, and Raph Levien. Better static memory management: Improving region-based analysis of higher-order languages. In *Proc. of the ACM SIGPLAN '95 Conference on Programming Language Design and Implementation*, pages 174–185, La Jolla, CA, USA, June 1995. ACM Press.
4. Alexander Aiken and David Gay. Memory management with explicit regions. In Keith D. Cooper, editor, *Proceedings of the 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 313–324, Montreal, Canada, June 1998. ACM. Volume 33(5) of SIGPLAN Notices.
5. Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. Region analysis and the polymorphic lambda calculus. In *Proc. of the 14th Annual IEEE Symposium on Logic in Computer Science*, pages 88–97, Trento, Italy, July 1999. IEEE Computer Society Press.
6. Lars Birkedal and Mads Tofte. A constraint-based region inference algorithm. *Theoretical Computer Science*, 58:299–392, 2001.
7. Lars Birkedal, Mads Tofte, and Magnus Vejlstrup. From region inference to von Neumann machines via region representation inference. In POPL1996 [43], pages 171–183.
8. Cristiano Calcagno, Simon Helsen, and Peter Thiemann. Syntactic type soundness results for the region calculus. *Information and Computation*, 173(2):199–221, 2002.

9. Morten V. Christiansen, Fritz Henglein, Henning Niss, and Per Velschow. Safe region-based memory management for objects. Technical report, DIKU, University of Copenhagen, October 1998.
10. Karl Crary, David Walker, and Greg Morrisett. Typed memory management in a calculus of capabilities. In Alex Aiken, editor, *Proc. 26th Annual ACM Symposium on Principles of Programming Languages*, pages 262–275, San Antonio, Texas, USA, January 1999. ACM Press.
11. Roy Crole and Andrew Gordon. A sound metalogical semantics for input/output. In *Computer Science Logic, CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 339–353, Kazimierz, Poland, 1995. Springer-Verlag.
12. Robert DeLine and Manuel Fähndrich. Enforcing high-level protocols in low-level software. In PLDI2001 [40], pages 59–69.
13. David Gay and Alexander Aiken. Language support for regions. In PLDI2001 [40], pages 70–80.
14. David K. Gifford and John M. Lucassen. Integrating functional and imperative programming. In *ACM Conference on Lisp and Functional Programming*, pages 28–38, 1986.
15. Andrew Gordon and Andrew Pitts, editors. *HOOTS '98 Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute. Cambridge University Press, 1998.
16. Andrew D. Gordon. *Functional Programming and Input/Output*. Cambridge University Press, 1994.
17. Andrew D. Gordon. Operational equivalences for untyped and polymorphic object calculi. In Gordon and Pitts [15], pages 9–54.
18. Andrew D. Gordon. Bisimilarity as a theory of functional programming. *Theoretical Computer Science*, 228(1-2):5–47, October 1999.
19. Andrew D. Gordon and Gareth D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In POPL1996 [43], pages 386–395.
20. Dan Grossman, Greg Morrisett, Trevor Jim, Mike Hicks, Yanling Wang, and James Cheney. Region-based memory management in cyclone. In *Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Berlin, Germany, June 2002.
21. Simon Helsen and Peter Thiemann. Syntactic type soundness for the region calculus. In Alan Jeffrey, editor, *ACM Workshop on Higher Order Operational Techniques in Semantics*, volume 41(3) of *Electronic Notes in Theoretical Computer Science*, pages 1–20, Montreal, Canada, September 2000. Elsevier Science.
22. Simon Helsen and Peter Thiemann. Polymorphic specialization for ML. Technical Report 173, Institut für Informatik, University of Freiburg, Germany, April 2002. Available from ftp://ftp.informatik.uni-freiburg.de/documents/reports/report173/report00173.ps.gz.
23. Fritz Henglein, Henning Makholm, and Henning Niss. A direct approach to control-flow sensitive region-based memory management. In Harald Søndergaard, editor, *Principles and Practice of Declarative Programming*, Firenze, Italy, September 2001.
24. Douglas J. Howe. Equality in lazy computation systems. In *Proc. of the 4th Annual Symposium on Logic in Computer Science*, pages 198–203, Pacific Grove, CA, June 1989. IEEE Computer Society Press.
25. Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

26. Trevor Jim, Greg Morrisett, Dan Grossman, Michael Hicks, James Cheney, and Yanling Wang. Cyclone: A safe dialect of c. In *USENIX Annual Technical Conference*, Monterey, CA, June 2002.
27. Pierre Jouvelot and David K. Gifford. Algebraic reconstruction of types and effects. In *Proc. 18th Annual ACM Symposium on Principles of Programming Languages*, pages 303–310, Orlando, Florida, January 1991. ACM Press.
28. Henning Makholm. Region-based memory management in prolog. Master's thesis, Department of Computer Science, University of Copenhagen, Denmark, March 2000.
29. Ian Mason and Carolyn Talcott. Programming, transforming, and proving with function abstractions and memories. In *Proc. Sixteenth International EATCS Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 574–588. Springer-Verlag, 1989.
30. Ian Mason and Carolyn Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):287–328, July 1991.
31. Ian Mason and Carolyn Talcott. Inferring the equivalence of functional programs that mutate data. In *Proc. of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 186–197. IEEE Computer Society Press, June 1992.
32. Robin Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1978.
33. Robin Milner. *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, NJ, 1989.
34. Robin Milner, Mads Tofte, Robert Harper, and Dave MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.
35. James H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT Press, December 1968.
36. Benjamin C. Pierce and Davide Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In Neil D. Jones, editor, *Proc. 24th Annual ACM Symposium on Principles of Programming Languages*, Paris, France, January 1997. ACM Press.
37. Andrew Pitts. Reasoning about local variables with operationlally-based logical relations. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.
38. Andrew Pitts. Operationally-based theories of program equivalence. In Andrew M. Pitts and Peter Dybjer, editors, *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.
39. Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In Gordon and Pitts [15], pages 309–326.
40. *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Snowbird, UT, June 2001.
41. Gordon Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.
42. Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Denmark, 1981.
43. *Proc. 23rd Annual ACM Symposium on Principles of Programming Languages*, St. Petersburg, Fla., January 1996. ACM Press.
44. S.B.Lassen. Relational reasoning about contexts. In Gordon and Pitts [15], pages 91–136.
45. Caroline Talcott. Reasoning about functions with effects. In Gordon and Pitts [15], pages 346–390.
46. Jean-Pierre Talpin and Pierre Jouvelot. Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3):245–272, July 1992.

47. Mads Tofte and Lars Birkedal. A region inference algorithm. *ACM Transactions on Programming Languages and Systems*, 20(5):724–767, 1998.
48. Mads Tofte and Lars Birkedal. Unification and polymorphism in region inference. In *Proof, Language, and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
49. Mads Tofte, Lars Birkedal, Martin Elsman, Niels Hallenberg, Tommy Olensen, and Peter Sestoft. *Programming with Regions in the ML Kit, Version 4.0.0*, September 2001. Available from http://www.it.edu/research/mlkit/dist/mlkit-4.0.0.pdf.
50. Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value λ-calculus using a stack of regions. In *Proc. 21st Annual ACM Symposium on Principles of Programming Languages*, pages 188–201, Portland, OR, January 1994. ACM Press.
51. Mads Tofte and Jean-Pierre Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.
52. Per Velschow and Morten Voetman. Region-based memory management in java. Master's thesis, DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark, May 1998.
53. David Walker, Carl Crary, and Greg Morrisett. Typed memory management via static capabilities. *ACM Transactions on Programming Languages and Systems*, 22(4):701–771, July 2000.
54. David Walker and Kevin Watkins. On regions and linear types. In Xavier Leroy, editor, *Proc. International Conference on Functional Programming 2001*, pages 181–192, Florence, Italy, September 2001. ACM Press, New York.
55. Paul R. Wilson. Uniprocessor garbage collection techniques. Technical report, University of Texas, 1999.
56. Silvano Dal Zilio and Andrew Gordon. Region analysis and a pi-calculus with groups. Technical Report MSR-TR-2000-57, Microsoft Research, August 2000.