An Equational Theory for a Region Calculus - Revised*

Simon Helsen
University of Freiburg[†], Germany

October 17, 2002

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 the region calculus. We show that our notion of 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 [57] 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, 48, 52]. Region annotations can be automatically inferred by means of a type and effect analysis [6, 49, 50, 53]. 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 [53], the ML-kit [51], a compiler for the functional programming language Standard ML [36], implements a practical implementation of region

*This is a revised and corrected version of Technical Report 172, Institut für Informatik, Universität Freiburg

[†]Institut für Informatik, Georges-Köhler-Allee 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, 30, 54]. In addition, many different flavors of the region calculus have been developed [4, 9, 12, 13, 20, 23, 55, 56], 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 the λ_{tt}^{region}-calculus.

Whereas type soundness properties for region calculi have been extensively examined [5, 8, 10, 53, 58], 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_{tt}^{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 annotation is guided by a type and effect analysis.

Using bisimulation, a proof method originating from work on CCS [35], we prove that the equational theory is sound with respect to Morris-style contextual equivalence [38, 44]. Following methods by Gordon, Abramsky and others [2, 11, 15–17, 19], we specify a notion of applicative bisimilarity for the region calculus. With a proof technique due to Howe [24,25], recasted to typed functional programming languages by Gordon [15], 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 [44] 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 [34] introduced his context lemma to reduce the need for full-blown contexts to evaluation contexts. The use of CCS-style bisimilarity [35] 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 [15–17] and Crole and Gordon [11] to other programming models. Notably, Gordon [17] proved that a form of bisimilarity for PCF with lazy streams is fully abstract with respect to contextual equivalence.

We closely follow Gordon's results on contextual equivalence for a polymorphically typed object calculus [16]. 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, 17] 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 sub-typing tells more programs apart than contextual equivalence. Experimental equivalence, a co-inductive variant of Milner's context lemma,

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

turns out to be complete for such object calculi [16]. 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 [31–33, 47]. Pitts [40] and Pitts and Stark [42] 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 [16, 39].

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 [58]. 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 λ_f^{region}-calculus, is a simple generalization of the λ_{tt}^{region}-calculus and introduced in section 2. We specify a dynamic semantics as a small-step reduction relation and define a type system based on an effect analysis. 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 the λ_f^{region}-calculus. 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.

1.3 Notation

For a function $f \in A \to B$, we define its *extension* $f+\{x \mapsto a\}$ as follows: $(f+\{x \mapsto a\})(x) = a$ and $(f + \{x \mapsto a\})(x') = f(x')$ whenever $x' \neq x$. A sequence of symbols a_i is written $\vec{a}$ and the set of the elements in a sequence $\vec{a}$ is denoted by $\{\vec{a}\}$. The symbol $=$ denotes equality of terms modulo α-conversion.

2 The λ_f^{region}-calculus

The λ_f^{region}-calculus, is a small variation of the λ_{tt}^{region}-calculus by Tofte and Talpin [53], similar in spirit to the computational extensions introduced in previous work [8, 21].

2.1 Syntax

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

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

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

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

Figure 1: The $\lambda_f^{\mathit{region}}$-calculus

Variables, function application and the let-construct are standard. The region allocation construct $\mathtt{new}\, \varrho.\, e$ 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. The dead region cannot be abstracted. 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 erroneous 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.

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 notation $e[x \mapsto e']$, stands for the term e with e' substituted for each free occurrence of x. As usual, substitution avoids capture of variables by α-renaming.

The grammar for the $\lambda_f^{\mathit{region}}$-calculus as described above actually defines *computational terms*. Such terms include intermediate syntactic forms that express computational state. They are needed for the small-step rewrite semantics presented below. 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 the presentation of an equational theory, but as in the Tofte-Talpin calculus, we may expect that compilers or programmers produce non-computational programs only.

2.2 Dynamic Semantics

The dynamic semantics of the $\lambda_f^{\mathit{region}}$-calculus is specified as a small-step reduction semantics in the style of Plotkin [45], 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

$$\lambda\, x.\, e \;\texttt{at}\; \varrho \overset{1}{\dashrightarrow} \langle \lambda\, x.\, e \rangle_{\varrho} \tag{1}$$

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

$$\texttt{new}\, \varrho.\, v \overset{3}{\dashrightarrow} v[\varrho \mapsto \bullet] \tag{3}$$

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

$$\texttt{let}\; x = v \;\texttt{in}\; e \overset{5}{\dashrightarrow} e[x \mapsto v] \tag{5}$$

$$\begin{gathered} \langle \texttt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e \rangle_{\varrho}\, [\rho'_1, \ldots, \rho'_n] \;\texttt{at}\; \varrho' \overset{6}{\dashrightarrow} \\ \langle \lambda\, x.\, e[\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n][f \mapsto \langle \texttt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e \rangle_{\varrho}] \rangle_{\varrho'} \end{gathered} \tag{6}$$

Figure 2: Reduction Rules for the λ_f^{region}-calculus

visible in the reduction rules. Intuitively, every `new`-abstracted 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 $\overset{n}{\dashrightarrow} \in \mathcal{P}(\mathsf{Term} \times \mathbb{N} \times \mathsf{Term})$ specifies beta-reduction rules for the λ_f^{region}-calculus and is defined in Figure 2. 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 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 the λ_f^{region}-calculus is defined by the relation $\rightharpoonup$ and its rules are listed in Figure 3. 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* ($e \uparrow$).

2.3 Static Semantics

The static semantics of the λ_f^{region}-calculus is a simple adaptation of the type system for the λ_{tt}^{region}-calculus found in [21, 53].

The categories $\mathsf{EffectVar}$ and $\mathsf{TypeVar}$ are disjoint denumerable sets of effect and type variables respectively. The type language of the λ_f^{region}-calculus is generated by the following grammar:

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

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

$$\mathsf{TypeWPlace} \ni \pi ::= \;\; (\overline{\sigma}, \rho) \qquad\qquad \mathsf{TypeScheme} \ni \sigma ::= \;\; \forall\, \vec{\alpha}.\forall\, \vec{\epsilon}.\overline{\sigma}$$

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

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

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

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

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

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

Figure 3: Small-step semantics for the λ_f^{region}-calculus

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 [50]. Effect variables can also occur in an effect set. A *result type* $\overline{\sigma}$ abstracts a type over region variables. Although this introduces rank-n region polymorphism (not type or effect polymorphism), it eases the presentation below.

As an extension to the Tofte-Talpin system, we take a *type with place* π to be a pair of a result type $\overline{\sigma}$ and a region ρ. Type variables, α, are taken from a denumerable set $\mathsf{TypeVar}$ and a *type scheme* σ abstracts a result type over type and effect variables.

A type substitution $S_t : \mathsf{TypeVar} \to \mathsf{Type}$ is a total function from type variables to types. An effect substitution $S_e : \mathsf{EffectVar} \to \mathsf{EffectVar} \times \mathsf{Effect}$ is a total function from effect variables onto arrow effects. Finally, a region substitution is a total function $S_r : \mathsf{RegionVar} \to \mathsf{Placeholder}$ from region variables to region placeholders. 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 onto $\epsilon.\{\}$.

The *support* of a type substitution, $\mathit{Supp}\,(S_t)$, is the set $\{\alpha \mid S_t(\alpha) \neq \alpha\}$. We sometimes 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 $\mathit{Supp}\,(S_r)$ and $\mathit{Supp}\,(S_e)$ are defined analogously. Similarly, we define the *image* of a type substitution S_t by $\mathit{Img}\,(S_t) = \big\{S_t(\alpha) \mid \alpha \in \mathit{Supp}\,(S_t)\big\}$ and analogously for S_e and S_r.

We define $S_t - \big\{\vec{\alpha}\big\}$ as the type substitution for which $(S_t - \big\{\vec{\alpha}\big\})(\alpha) = S_t(\alpha)$ whenever $\alpha \in (\mathit{Supp}\,(S_t) \setminus \big\{\vec{\alpha}\big\})$. Otherwise, we have $(S_t - \big\{\vec{\alpha}\big\})(\alpha) = \alpha$ and analogously for S_e and S_r. For an object O, we may 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 substitution $S_s = (S_t, S_e, S_r)$ is a triple of a type substitution S_r, an effect substitution S_e and a region substitution S_t. The substitution S_s distributes operations component-wise over its three substitution. We write S_r as a shorthand for (I_t, I_e, S_r) and similarly for the other substitutions. The definition of S_s on types and effects is now defined as follows:

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

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

$$S_s(\alpha) = S_t(\alpha) \quad S_s(\varrho) = S_r(\varrho) \quad S_s(\bullet) = \bullet \quad S_s(\forall\, \vec{\varrho}.\tau) = \forall\, \vec{\varrho}.(S_s - \{\vec{\varrho}\})(\tau)$$

$$S_s(\overline{\sigma}, \rho) = (S_s(\overline{\sigma}), S_s(\rho)) \quad S_s(\forall\, \vec{\alpha}.\forall\, \vec{\epsilon}.\overline{\sigma}) = \forall\, \vec{\alpha}.\forall\, \vec{\epsilon}.(S_s - (\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}))(\overline{\sigma})$$

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

A type environment, $TE \in \mathsf{TypeEnv}$, is a partial function mapping 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 $ftv\,(O)$, $frv\,(O)$, and $fev\,(O)$ for O's free type, region (placeholder) and effect variables respectively. Also, $fv\,(O) = ftv\,(O) \cup frv\,(O) \cup fev\,(O)$.

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

A typing judgment $TE \vdash_f e : (\overline{\sigma}, \rho), \varphi$ assigns type with place $(\overline{\sigma}, \rho)$ and effect φ to expression e given a type environment TE. The type rules are listed in Figure 4.

Since the type rules for the λ_f^{region}-calculus are very similar to those of the λ_{tt}^{region}-calculus [21, 53], we only discuss some peculiarities specific to the λ_f^{region}-calculus.

Because recursive functions are first-class objects, we have separated region polymorphism from type polymorphism. Rules *(F-Fix)* and *(F-Fixv)* type a recursive function and its value counterpart respectively. The result type of the recursive function generalizes over region variables. The recursive binding additionally generalizes over effect variables, but not over type variables. Type polymorphism is introduced by type rule *(F-Let)*, which specifies standard Hindley-Milner `let`-polymorphism. The *(F-Let)* rule also generalizes over effect variables.

Type rule *(F-Var)* now only instantiates a type scheme to a result type. Further instantiation of a result type into a "normal" type is done upon region application, as formulated in type rule *(F-Regapp)*.

As for the λ_{tt}^{region}-calculus, we have polymorphic recursion for region and effect variables, but not for type variables. The use of result types in type judgments introduces rank-n region polymorphism, which may make region reconstruction more difficult or even undecidable. However, this is irrelevant for the development of the equational theory.

2.4 Properties of the λ_f^{region}-calculus

Similar to the λ_{tt}^{region}-calculus, the λ_f^{region}-calculus exhibits some elementary properties, which we present without proof. The proofs are either straightforward inductions or can be found elsewhere [8, 21].

We have a type substitution lemma as usual:

Lemma 2.1 *If $TE \vdash_f e : \pi, \varphi$ then, for any type substitution S_s, we have that $S_s(TE) \vdash_f S_s(e) : S_s(\pi), S_s(\varphi)$.*

We have a variable substitution lemma as well:

(F-Var)
$$\frac{TE(x) = (\sigma, \rho) \quad \overline{\sigma} \prec \sigma \;\; via \;\; (S_t, S_e, I_r)}{TE \vdash_f x : (\overline{\sigma}, \rho), \emptyset}$$

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

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

(F-Reglam)
$$\frac{TE \vdash_f e : \pi, \varphi \quad \varrho \notin frv\,(TE, \pi)}{TE \vdash_f \texttt{new}\, \varrho.\, e : \pi, \varphi \setminus \{\varrho\}}$$

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

(F-Let)
$$\frac{\begin{array}{c} TE \vdash_f e_1 : (\overline{\sigma}', \rho'), \varphi' \quad (\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}) \cap fv\,(TE) = \emptyset \\ TE + \{x \mapsto (\forall\, \vec{\alpha}\,.\forall\, \vec{\epsilon}.\overline{\sigma}', \rho')\} \vdash_f e_2 : \pi, \varphi \end{array}}{TE \vdash_f \texttt{let}\, x = e_1 \,\texttt{in}\, e_2 : \pi, \varphi \cup \varphi'}$$

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

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

(F-Regapp)
$$\frac{\begin{array}{c} TE \vdash_f e : (\forall\, \varrho_1, \ldots, \varrho_n\,.\tau, \rho), \varphi \\ S_r = \{\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n\} \end{array}}{TE \vdash_f e\,[\rho'_1, \ldots, \rho'_n] \; \texttt{at}\, \rho' : (S_r(\tau), \rho'), \varphi \cup \{\rho, \rho'\}}$$

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

Figure 4: The Type System for the λ_f^{region}-calculus

Lemma 2.2 (Substitution Lemma) *Suppose* $TE \vdash_f v : (\overline{\sigma}, \rho), \emptyset$ *and take* $\vec{\alpha}$ *and* $\vec{\epsilon}$ *such that* $(\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}) \cap fv\,(TE) = \emptyset$*. Additionally, assume* $TE + \{x \mapsto (\forall\,\vec{\alpha}\,.\forall\,\vec{\epsilon}.\overline{\sigma}, \rho)\} \vdash_f e : \pi, \varphi$*. Then we have* $TE \vdash_f e[x \mapsto v] : \pi, \varphi$

The λ_f^{region}-calculus is obviously not strongly normalizing. We can define a canonical well-typed diverging program. For arbitrary ϱ, we define

$$\Omega_\varrho = ((\texttt{fix}\, f.\Lambda\, \varrho.\lambda\, x.\, (f\ [\varrho]\ \texttt{at}\ \varrho) \,@\, x\ \texttt{at}\ \varrho)\ [\varrho]\ \texttt{at}\ \varrho) \,@\, (\lambda\, x.\, x\ \texttt{at}\ \varrho)$$

We always have that $\Omega_\varrho \Uparrow$ and for any $\overline{\sigma}$ also that $\{\ \} \vdash_f \Omega_\varrho : (\overline{\sigma}, \varrho), \{\varrho\}$.

The reduction relation $\rightharpoonup$ for the λ_f^{region}-calculus obeys a type preservation property:

Proposition 2.3 (Type Preservation) *Suppose* $TE \vdash_f e : \pi, \varphi$*. If* $e \rightharpoonup e'$ *then there exists an effect* φ' *such that* $TE \vdash_f e' : \pi, \varphi'$ *with* $\varphi' \subseteq \varphi$*.*

We have a canonical forms lemma for Values:

Lemma 2.4 (Canonical Forms Lemma) *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* $\pi = (\forall\, \varrho_1, \ldots, \varrho_n\,.\tau, \rho)$ *and* $n \geq 1$*,*
  then v *is of the form* $\langle \texttt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e \rangle_\rho$*.*

A progress property is also standard:

Proposition 2.5 (Progress) *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 the λ_f^{region}-calculus is now as follows:

Theorem 2.6 (Type Soundness of the λ_f^{region}-calculus) *Suppose* $\{\ \} \vdash_f e : \pi, \varphi$ *and* $Clean\,(\varphi)$*. Then,* $e \Downarrow$ *or* $e \Uparrow$*.*

The small-step reduction $\rightharpoonup$ is deterministic:

Lemma 2.7 *Suppose* $e \rightharpoonup e'$ *and* $e \rightharpoonup e''$*, then* $e' = e''$*.*

The reduction relation is invariant under expression substitution:

Lemma 2.8 *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 2.9 *Suppose* $TE \vdash_f e : \pi, \varphi$ *and for an* S_r *it holds that* $Clean\,(S_r(\varphi))$*. Then* $e \rightharpoonup e'$ *implies* $S_r(e) \rightharpoonup S_r(e')$*.*

As mentioned earlier, the use of the λ_f^{region}-calculus instead of the λ_{tt}^{region}-calculus facilitates the formulation of an equational theory. We now illustrate that the λ_f^{region}-calculus is a proper generalization of the λ_{tt}^{region}-calculus [21], defined by the following grammar:

$$\begin{array}{rcl}
\lambda_f^{region}\textsf{-Term} \ni \mathbf{e} & ::= & x \mid \mathbf{e} \,@\, \mathbf{e} \mid \texttt{let}\ x = \mathbf{e}\ \texttt{in}\ \mathbf{e} \mid \texttt{new}\ \varrho.\, \mathbf{e} \mid \lambda\, x.\, \mathbf{e}\ \texttt{at}\ \rho \mid \langle \lambda\, x.\, \mathbf{e} \rangle_\rho \mid \\
& & \texttt{letrec}\ f = \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, \mathbf{e}\ \texttt{at}\ \rho\ \texttt{in}\ \mathbf{e} \mid \\
& & \texttt{letrec}\ f = \langle \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, \mathbf{e} \rangle_\rho\ \texttt{in}\ \mathbf{e} \mid f\ [\rho_1, \ldots, \rho_n]\ \texttt{at}\ \rho \mid \\
& & \langle \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, \mathbf{e} \rangle_\rho\ [\rho'_1, \ldots, \rho'_n]\ \texttt{at}\ \rho'
\end{array}$$

Define a translation function $[\![\cdot]\!] \in \lambda_{tt}^{region}\text{-}\mathsf{Term} \to \lambda_{f}^{region}\text{-}\mathsf{Term}$ inductively on $\lambda_{tt}^{region}\text{-}\mathsf{Terms}$, where we omit the cases that only propagate the translation to sub-terms:

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

$$
\begin{array}{l}
[\![\mathtt{letrec}\ f = \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, \mathbf{e}_1\ \mathtt{at}\ \rho\ \mathtt{in}\ \mathbf{e}_2]\!] = \\
\qquad\qquad \mathtt{let}\ f = \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, [\![\mathbf{e}_1]\!]\ \mathtt{at}\ \rho\ \mathtt{in}\ [\![\mathbf{e}_2]\!]
\end{array}
$$

$$
\begin{array}{l}
[\![\mathtt{letrec}\ f = \langle \Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, \mathbf{e}_1\rangle_\rho\ \mathtt{in}\ \mathbf{e}_2]\!] = \\
\qquad\qquad \begin{cases}
[\![\mathbf{e}_1]\!][f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, [\![\mathbf{e}_1]\!]\rangle_\rho] & \textit{if}\ \mathbf{e}_1 = \mathbf{e}_2 \\
\mathtt{let}\ f = \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, [\![\mathbf{e}_1]\!]\rangle_\rho\ \mathtt{in}\ [\![\mathbf{e}_2]\!] & \textit{if}\ \mathbf{e}_1 \neq \mathbf{e}_2
\end{cases}
\end{array}
$$

The type system specified by the entailment $\vdash_t^t$ and a small-step reduction semantics $\to$ for the λ_{tt}^{region}-calculus are defined elsewhere [21] and we do not repeat them here.

We have the following two properties, proven by a straightforward rule induction:

Proposition 2.10 *Suppose* $TE \vdash_t^t \mathbf{e} : (\tau, \rho), \varphi$, *then* $TE \vdash_f [\![\mathbf{e}]\!] : (\tau, \rho), \varphi$.

Define $\rightharpoonup^{0,1}$ as the relation making one or no $\rightharpoonup$ reduction steps.

Proposition 2.11 *Suppose* $\mathbf{e} \to \mathbf{e}'$, *then* $[\![\mathbf{e}]\!] \rightharpoonup^{0,1} [\![\mathbf{e}']\!]$ *too.*

3 Typed Relations and Open extensions

Let $\mathsf{TypedRel} = \mathcal{P}(\mathsf{TypeEnv} \times \mathsf{Term} \times \mathsf{Term} \times \mathsf{TypeWPlace} \times \mathsf{Effect})$. A *typed relation* $\mathcal{R} \in \mathsf{TypedRel}$ is a set of quintuples $(TE, e_1, e_2, \pi, \varphi)$. 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 3.1 *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.*

Well-formedness implies a type invariant together with a subsumption property for effect sets. 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 $\mathit{Clean}\,(\varphi)$. Note that the empty relation is *not* reflexive. The inverse of a typed relation $\mathcal{R}$ is now as follows: $\mathcal{R}^{-1} = \big\{(TE, e_2, e_1, \pi, \varphi) \mid TE \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi\big\}$. 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, *i.e.* a finite function from Var to Value: $(x_1 \mapsto v_1, \ldots, x_n \mapsto v_n)$. We denote the empty sequence by ε. An iterated substitution works over λ_f^{region}-terms as follows:

$$\varepsilon(e) = e$$
$$(\xi, x \mapsto v)(e) = \xi(e[x \mapsto v])$$

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:

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

We have the following straightforward property for completions:

Lemma 3.1 *If $TE \vdash_f e : \pi, \varphi$ and $S_r(TE) \succ \xi$ for any region substitution S_r, then $\{\ \} \vdash_f \xi(S_r(e)) : S_r(\pi), S_r(\varphi)$.*

Proof. First, apply Lemma 2.1 on the type judgment and then prove by induction on ξ, using Lemma 2.2 for the induction step. $\square$

The *open extension* of a well-formed closed typed relation $\mathcal{R}$ is a well-formed typed relation $\mathcal{R}^\circ$ such that

$$\begin{array}{ll} TE \rhd e_1 \,\mathcal{R}^\circ\, e_2 : \pi, \varphi \ \textit{iff} & \textit{for all } \xi \textit{ and for all } S_r \textit{ where } S_r(TE) \succ \xi \\ & \textit{and Clean}\,(S_r(\varphi)) \textit{ it holds that} \\ & \{\ \} \rhd \xi(S_r(e_1)) \,\mathcal{R}\, \xi(S_r(e_2)) : S_r(\pi), S_r(\varphi) \end{array}$$

This definition may be a little surprising because of the quantified region substitution S_r. However, this somewhat stronger condition is related to the presence of region polymorphism in the λ_f^{region}-calculus. The condition implies that $\{\ \} \rhd e_1 \,\mathcal{R}^\circ\, e_2 : \pi, \varphi$ is not necessarily equivalent to $\{\ \} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ because of the substitution S_r. We will come back to this restricted view of open extensions in Section 6.2, where we introduce bisimilarity.

An iterated substitution can be formulated as a sequence of nested let-expressions:

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

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

Lemma 3.2 *Suppose a well-formed closed typed relation $\mathcal{R}$. If $TE \vdash_f v : (\overline{\sigma}, \rho), \emptyset$ and $TE + \{x \mapsto (\forall \vec{\alpha}.\forall \vec{\epsilon}.\overline{\sigma}, \rho)\} \rhd e_1 \,\mathcal{R}^\circ\, e_2 : \pi, \varphi$ where $(\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}) \cap fv\,(TE) = \emptyset$, then $TE \rhd e_1[x \mapsto v] \,\mathcal{R}^\circ\, e_2[x \mapsto v] : \pi, \varphi$.*

Proof. First note that without loss of generality, we can assume that $x \notin Dom\,(TE)$. By definition of open extension, take ξ and S_r such that $S_r(TE) \succ \xi$ with $Clean\,(S_r(\varphi))$. So, we have to show that $\{\ \} \rhd \xi(S_r(e_1[x \mapsto v])) \,\mathcal{R}\, \xi(S_r(e_2[x \mapsto v])) : S_r(\pi), S_r(\varphi)$, which is by definition of iterated substitution the same as $\{\ \} \rhd (\xi, x \mapsto S_r(v))(S_r(e_1)) \,\mathcal{R}\, (\xi, x \mapsto S_r(v))(S_r(e_1)) : S_r(\pi), S_r(\varphi)$.

We have by assumption that $TE \vdash_f v : (\overline{\sigma}, \rho), \emptyset$ and therefore by Lemma 2.1 that $S_r(TE) \vdash_f S_r(v) : (S_r(\overline{\sigma}), S_r(\rho)), \emptyset$. Take S_t and S_e such that $Supp\,(S_t) = \vec{\alpha}$, $Supp\,(S_e) = \vec{\epsilon}$ and $S_r(\overline{\sigma}) \prec S_r(\sigma)\ via\ (S_t, S_e, I_r)$. Since we also have $S_r(TE) \succ \xi$, by definition of completion, also $S_r(TE + \{x \mapsto (\forall \vec{\alpha}.\forall \vec{\epsilon}.\overline{\sigma}, \rho)\}) \succ (\xi, x \mapsto S_r(v))$.

Therefore, since we also have by assumption that $TE + \{x \mapsto (\forall \vec{\alpha}.\forall \vec{\epsilon}.\overline{\sigma}, \rho)\} \rhd e_1 \,\mathcal{R}^\circ\, e_2 : \pi, \varphi$, the claim $\{\ \} \rhd (\xi, x \mapsto S_r(v))(S_r(e_1)) \,\mathcal{R}\, (\xi, x \mapsto S_r(v))(S_r(e_1)) : S_r(\pi), S_r(\varphi)$ follows. $\square$

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 [38] and popularized by Plotkin [44]. Intuitively, two expressions are contextual equivalent whenever an arbitrary context exhibits the same termination behavior within the operational semantics. Similar to Gordon [16], we formalize this notion for the λ_f^{region}-calculus with a style introduced by Lassen [28].

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 4 without rule *(F-Effect)*. However, to assure that the compatible extension does not change the typing, we must guarantee that superfluous effect variables can be eliminated in the rules *(Comp-Reglam)*, *(Comp-App)*, *(Comp-Let)* and *(Comp-Regapp)*. We say that a typed relation $\mathcal{R}$ is *compatible* iff $\widehat{\mathcal{R}} \subseteq \mathcal{R}$. Note that the empty relation is not compatible due to rule *(Comp-Var)*.

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. Let us explain this with an 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):

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

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 our definition of compatibility (by only considering clean effects) is that well typedness alone does not guarantee progress as witnessed by the soundness Theorem 2.6. 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 4.1 *Consider 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.*

$$\textit{(Comp-Var)} \quad \frac{TE(x) = (\sigma, \rho) \quad \overline{\sigma} \prec \sigma \ \textit{via}\ (S_t, S_e, I_r) \quad \textit{Clean}\,(\varphi)}{TE \rhd x \,\widehat{\mathcal{R}}\, x : (\overline{\sigma}, \rho), \varphi}$$

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

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

$$\textit{(Comp-Reglam)} \quad \frac{TE \rhd e \,\mathcal{R}\, e' : \pi, \varphi' \quad \varrho \notin \textit{frv}\,(TE, \pi) \quad \textit{Clean}\,(\varphi) \quad \epsilon \notin \textit{fev}\,(TE, \pi)}{TE \rhd \texttt{new}\, \varrho.\, e \,\widehat{\mathcal{R}}\, \texttt{new}\, \varrho.\, e' : \pi, (\varphi' \setminus \{\varrho, \epsilon\}) \cup \varphi}$$

$$\textit{(Comp-App)} \quad \frac{\begin{array}{c} TE \rhd e_1 \,\mathcal{R}\, e_1' : (\pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2, \varrho), \varphi_1 \\ \textit{Clean}\,(\varphi) \quad TE \rhd e_2 \,\mathcal{R}\, e_2' : \pi_1, \varphi_2 \quad \epsilon' \notin \textit{fev}\,(TE, \pi_2) \end{array}}{TE \rhd e_1 \mathbin{@} e_2 \,\widehat{\mathcal{R}}\, e_1' \mathbin{@} e_2' : \pi_2, (\varphi' \cup \varphi_1 \cup \varphi_2 \cup \{\epsilon, \varrho\} \cup \varphi) \setminus \{\epsilon'\}}$$

$$\textit{(Comp-Let)} \quad \frac{\begin{array}{c} TE \rhd e_1 \,\mathcal{R}\, e_1' : (\overline{\sigma}', \rho'), \varphi_1 \\ (\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}) \cap \textit{fv}\,(TE) = \emptyset \quad \textit{Clean}\,(\varphi) \quad \epsilon' \notin \textit{fev}\,(TE, \pi) \\ TE + \{x \mapsto (\forall\, \vec{\alpha}.\forall\, \vec{\epsilon}.\overline{\sigma}', \rho')\} \rhd e_2 \,\mathcal{R}\, e_2' : \pi, \varphi_2 \end{array}}{TE \rhd \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \,\widehat{\mathcal{R}}\, \texttt{let}\ x = e_1'\ \texttt{in}\ e_2' : \pi, (\varphi_1 \cup \varphi_2 \cup \varphi) \setminus \{\epsilon'\}}$$

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

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

$$\textit{(Comp-Regapp)} \quad \frac{\begin{array}{c} TE \rhd e \,\mathcal{R}\, e' : (\forall\, \varrho_1, \ldots, \varrho_n\,.\tau, \varrho), \varphi' \\ \textit{Clean}\,(\varphi) \quad S_r = \{\varrho_1 \mapsto \rho_1', \ldots, \varrho_n \mapsto \rho_n'\} \quad \epsilon \notin \textit{fev}\,(TE, \tau) \end{array}}{\begin{array}{l} TE \rhd e\ [\rho_1', \ldots, \rho_n']\ \texttt{at}\ \varrho' \,\widehat{\mathcal{R}} \\ \qquad e'\ [\rho_1', \ldots, \rho_n']\ \texttt{at}\ \varrho' : (S_r(\tau), \varrho'), (\varphi' \cup \{\varrho, \varrho'\} \cup \varphi) \setminus \{\epsilon\} \end{array}}$$

Figure 5: Compatible refinement of typed relations

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

Lemma 4.2 *Suppose a typed relation $\mathcal{R}^\circ$ which is well-formed and compatible. If for all S_r and ξ we have $S_r(TE) \succ \xi$ with $Clean\,(S_r(\varphi))$ and $TE \rhd e_1\, \mathcal{R}^\circ\, e_2 : \pi, \varphi$, then we also have that $\{\ \} \rhd \mathcal{T}(\xi, S_r(e_1))\, \mathcal{R}\, \mathcal{T}(\xi, S_r(e_2)) : S_r(\pi), S_r(\varphi)$*

Proof. Proven by simple induction on the length of ξ and using rule *(Comp-Let)* and Lemma 2.1 in the induction step. $\square$

4.2 Context Closure

The compatible refinement operator builds arbitrary non-empty contexts. However, since we also need empty contexts, we define the *context closure* $\mathcal{R}^c$ of a typed relation $\mathcal{R}$ as the least relation, closed under the rules

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

Intuitively, a context closure of a typed relation includes all well-typed contexts of the related λ_f^{region}-expressions. Therefore, the context closure of the relation $\{(TE, e_1, e_2, \pi, \varphi)\}$ relates all typed contexts of the expressions e_1 and e_2. The following properties trivially hold:

Lemma 4.3 *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 complete lattice* $(\mathsf{TypedRel}, \subseteq)$.
4. *The relation $\mathcal{R}$ 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 4.4

1. *$TE' \rhd e'\ \{(TE, e, e, \pi, \varphi)\}^c\ e'' : \pi', \varphi'$ implies $e' = e''$.*
2. *$TE' \rhd e_1'\ \{(TE, e_1, e_2, \pi, \varphi)\}^c\ e_2' : \pi', \varphi'$ implies $TE' \rhd e_2'\ \{(TE, e_2, e_1, \pi, \varphi)\}^c\ e_1' : \pi', \varphi'$.*
3. *$TE' \rhd e_1'\ \{(TE, e_1, e_3, \pi, \varphi)\}^c\ e_3' : \pi', \varphi'$ implies that for all e_2, there exist 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. *$TE' \rhd e_1'\ \{(TE, e_1, e_2, \pi, \varphi)\}^c\ e_2' : \pi', \varphi'$ and $TE'' \rhd e_1''\ \{(TE', e_1', e_2', \pi', \varphi')\}^c\ e_2'' : \pi'', \varphi''$ imply that $TE'' \rhd e_1''\ \{(TE, e_1, e_2, \pi, \varphi)\}^c\ e_2'' : \pi'', \varphi''$.*

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 4.5 *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 $TE' \rhd e_1' \,\big\{(TE, e_1, e_2, \pi, \varphi)\big\}^c\, e_2' : \pi', \varphi'$.*

Proof.

Case $\Rightarrow$. Take $TE \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$, so $\big\{(TE, e_1, e_2, \pi, \varphi)\big\} \subseteq \mathcal{R}$. By Lemma 4.3.3, we have that $\big\{(TE, e_1, e_2, \pi, \varphi)\big\}^c \subseteq \mathcal{R}^c$. And since $\mathcal{R}$ is compatible, we also have $\mathcal{R}^c \subseteq \mathcal{R}$ by Lemma 4.3.4. This implies $\big\{(TE, e_1, e_2, \pi, \varphi)\big\}^c \subseteq \mathcal{R}$ 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 *(F-Var)* implies $TE \vdash_f x : (\hat{\sigma}, \rho), \emptyset$ and we are given that $\mathcal{R}$ is reflexive.

Case *(Comp-Let)*. Suppose $TE \rhd \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \,\widehat{\mathcal{R}}\, \texttt{let}\ x = e_1'\ \texttt{in}\ e_2' : \pi, \varphi$. Then $TE \rhd e_1 \,\mathcal{R}\, e_1' : (\overline{\sigma}', \rho'), \varphi_1$ and $TE + \{x \mapsto (\forall\, \vec{\alpha}.\forall\, \vec{\epsilon}.\overline{\sigma}', \rho')\} \rhd e_2 \,\mathcal{R}\, e_2' : \pi, \varphi_2$ with $(\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}) \cap fv\,(TE) = \emptyset$ and $\varphi_1 \cup \varphi_2 \subseteq \varphi$.

- Take $\mathcal{R}' = \big\{(TE, e_1, e_1', (\overline{\sigma}', \rho'), \varphi_1)\big\}$. By *(Closure-Empty)* we have that $TE \rhd e_1 \,\mathcal{R}'^c\, e_1' : (\overline{\sigma}', \rho'), \varphi_1$. Since $\mathcal{R}$ is well-formed and by Lemma 4.3.2, we have $TE + \{x \mapsto (\forall\, \vec{\alpha}.\forall\, \vec{\epsilon}.\overline{\sigma}', \rho')\} \rhd e_2 \,\mathcal{R}'^c\, e_2 : \pi, \varphi_2$. Hence, using rule *(Comp-Let)* we have $TE \rhd \texttt{let}\, x = e_1\, \texttt{in}\, e_2 \,\widehat{\mathcal{R}'^c}\, \texttt{let}\, x = e_1'\, \texttt{in}\, e_2 : \pi, \varphi$. Rule *(Closure-Comp)* and the assumption give $TE \rhd \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \,\mathcal{R}\, \texttt{let}\ x = e_1'\ \texttt{in}\ e_2 : \pi, \varphi$.
- By analogous reasoning, we also derive $TE \rhd \texttt{let}\ x = e_1'\ \texttt{in}\ e_2 \,\mathcal{R}\, \texttt{let}\ x = e_1'\ \texttt{in}\, e_2' : \pi, \varphi$.

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

All other cases are proven by similar reasoning.

□

4.3 Contextual Equivalence

We now have enough material to introduce a general notion of contextual equivalence for the λ_f^{region}-calculus.

We say that 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 $e_1 \downharpoonleft \Leftrightarrow e_2 \downharpoonleft$.

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

Definition 4.1 (Contextual Equivalence for the λ_f^{region}-calculus)
Let contextual equivalence, $\simeq\ \in$ $\mathsf{TypedRel}$, be a well-formed typed relation on λ_f^{region}-Terms such that $TE \rhd e_1 \simeq e_2 : \pi, \varphi$ iff $\big\{(TE, e_1, e_2, \pi, \varphi)\big\}^c$ is adequate.

This definition of contextual equivalence is based on the reduction semantics of the λ_f^{region}-calculus. We note that this choice is not fundamental, since the λ_f^{region}-calculus semantics is reminiscent of the λ_{tt}^{region}-calculus semantics, which we have proven equivalent in some sense to the big-step evaluation style formulation of Tofte and Talpin [8].

Lemma 4.6 *Contextual equivalence is well-defined,* i.e. *an equivalence relation.*

Proof. Reflexivity follows from Lemma 4.4.1, symmetry from Lemma 4.4.2. 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 $e_1' \downharpoonleft \Leftrightarrow e_3' \downharpoonleft$, we have that $TE \rhd e_1 \simeq e_3 : \pi, \varphi$, *i.e.* transitivity.

Suppose now that $e_1' \downharpoonleft$. Using Lemma 4.4.3, 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 $e_1' \downharpoonleft$ and $TE \rhd e_1 \simeq e_2 : \pi, \varphi$, it must be that $e_2' \downharpoonleft$. This, together with $TE \rhd e_2 \simeq e_3 : \pi, \varphi$ implies $e_3' \downharpoonleft$. Symmetrically, we show that $e_3' \downharpoonleft \Rightarrow e_1' \downharpoonleft$, proving the case. □

Proposition 4.7 *Contextual equivalence is the greatest typed relation that is well-formed, compatible and adequate.*

Proof.

1. By rule *(Closure-Empty)*, we immediately have that contextual equivalence is adequate and by definition well-formed.

   Using Proposition 4.5 and Lemma 4.6, 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 4.3.1, $\{(TE, e_1, e_2, \pi, \varphi)\}^c$ is also well-formed. For adequacy of the relation $\{(TE', e_1', e_2', \pi', \varphi')\}^c$, assume $\{\ \} \rhd e_1'' \ \{(TE', e_1', e_2', \pi', \varphi')\}^c \ e_2'' : \pi'', \varphi''$. By Lemma 4.4.4 and the assumptions, $\{\ \} \rhd e_1'' \ \{(TE, e_1, e_2, \pi, \varphi)\}^c \ e_2'' : \pi'', \varphi''$ holds. Also by assumption, we have $TE \rhd e_1 \simeq e_2 : \pi, \varphi$ and therefore $e_1'' \downharpoonleft \Leftrightarrow e_2'' \downharpoonleft$, 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 4.3.3 and Lemma 4.3.4 that the inclusion $\{(TE, e_1, e_2, \pi, \varphi)\}^c \subseteq \mathcal{R}$ holds. Because $\mathcal{R}$ is also adequate, we derive that the relation $\{(TE, e_1, e_2, \pi, \varphi)\}^c$ 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 $\triangleq$ as the least typed relation closed under the rules in Figure 6.

$$\textit{(Eq-Comp)}\quad \frac{TE \rhd e \,\widehat{\triangleq}\, e' : \pi, \varphi}{TE \rhd e \triangleq e' : \pi, \varphi}$$

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

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

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

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

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

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

$$\textit{(Eq-Drop)}\quad \frac{TE \vdash_f e : \pi, \varphi' \quad \varrho \notin \mathit{frv}\,(TE, \pi) \quad \varphi' \subseteq \varphi \quad \mathit{Clean}\,(\varphi)}{TE \rhd e \triangleq \mathtt{new}\, \varrho.\, e : \pi, \varphi}$$

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

Figure 6: An equational theory for the λ_f^{region}-calculus

Rule *(Eq-Comp)* guarantees that the relation Eqq is compatible. As a result of Lemma 4.1.1 it follows that the relation $\triangleq$ is reflexive as well. The rules *(Eq-Symm)* and *(Eq-Trans)* make the relation an equivalence.

The beta-reduction rules (1), (2), (5) and (6) become typed and bidirectional in rule *(Eq-Beta-n)*. The index n refers to the corresponding reduction rule in Figure 2.

Rule *(Eq-App-Cbv)* equates 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 use elsewhere [22] to prove the soundness of a specializer. Rule *(Eq-App-Let)* is a `let`-insertion rule for function application, which makes the left-to-right evaluation of the λ_f^{region}-calculus more explicit. Rule *(Eq-Let-Let)* is standard `let`-flattening [37] and allows for the un-nesting of `let`-expressions. The rule *(Eq-Drop)* specifies that the allocation of a region in a safe place (guaranteed by the condition $\varrho \notin \mathit{frv}\,(\mathit{TE}, \pi)$) 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)*.

In comparison, the typed equational theory for a region calculus by Dal Zilio and Gordon [59] 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 in their system. In contrast, in our system, the following rules from their theory are derivable:

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

$$\textit{(Eq-Rlet)}\quad \frac{\begin{array}{c}\mathit{TE} \vdash_f e_1 : (\overline{\sigma}', \rho'), \varphi_1 \quad (\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}) \cap \mathit{fv}\,(\mathit{TE}) = \emptyset \\ (\varphi_1 \cup \varphi_2) \setminus \{\varrho\} \subseteq \varphi \quad \mathit{Clean}\,(\varphi) \quad \varrho \notin \mathit{frv}\,(\mathit{TE}, (\overline{\sigma}', \rho'), \pi) \\ \mathit{TE} + \{x \mapsto (\forall\,\vec{\alpha}.\forall\,\vec{\epsilon}.\overline{\sigma}', \rho')\} \vdash_f e_2 : \pi, \varphi_2\end{array}}{\mathit{TE} \rhd \mathtt{new}\,\varrho.\,(\mathtt{let}\,x = e_1\,\mathtt{in}\,e_2) \triangleq \mathtt{let}\,x = (\mathtt{new}\,\varrho.\,e_1)\,\mathtt{in}\,(\mathtt{new}\,\varrho.\,e_2) : \pi, \varphi}$$

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

Dal Zilio and Gordon also specify an equational rule (Eq Drop). However, it is weaker than *(Eq-Drop)* and hence, they have to specify variants of the rules *(Eq-Swap)* and *(Eq-Rlet)* as axioms in their equational theory.

Lemma 5.1 *The relation $\triangleq$ is well-formed.*

Proof. By induction on the derivation, using Lemma 4.1.2, Proposition 2.3 and inspecting the premises. □

We also have that the encoding of a completion can be equated to the application of the completion to an expression:

Lemma 5.2 *If* $TE \vdash_f e : \pi, \varphi$ *and* $S_r(TE) \succ \xi$ *with* $Clean\,(S_r(\varphi))$ *then* $\{\ \} \rhd \xi(S_r(e)) \triangleq \mathcal{T}(\xi, S_r(e)) : S_r(\pi), S_r(\varphi)$.

Proof. A simple inductive argument on the length of ξ using equational rule *(Eq-Beta-5)* with Lemma 2.1 and Lemma 2.9. □

The equational theory as specified above is formulated for the λ_f^{region}-calculus and includes intermediate computational terms only required for the small-step transition semantics of Figure 2 and Figure 3. In practice, it is often desirable to prove program transformations on non-computational terms, *i.e.* terms which are used by a programmer or compiler. It is possible to formulate an equational theory directly for such terms, however, the rules would be clumsier and technically difficult to prove sound. Alternatively, such rules can be *derived* from the rules in Figure 6. We omit the details.

While the rest of the paper deals with the soundness of the theory with respect to contextual equivalence, we note that the equational rules are not complete: they are not sufficient to show semantic equality of arbitrary programs. The latter would require a co-inductively defined equational system.

6 Bisimilarity for the λ_f^{region}-calculus

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 the λ_f^{region}-calculus and use it to prove soundness of $\triangleq$.

6.1 A Labeled Transition System

Following Crole and Gordon [11,17], 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 the λ_f^{region}-calculus 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 the λ_f^{region}-calculus, *i.e.* $e\gamma \in \lambda_f^{region}$-$\mathsf{Term}$. 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: *Deadly* (γ) is true iff $\gamma = \bullet$ or $\gamma = [\rho_1, \ldots, \rho_n]\ \mathtt{at} \bullet$. The predicate *Safe* (γ) holds iff *Deadly* (γ) is false, *i.e.* observations are *safe*.

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 by the rules in Figure 7.

In a call-by-value language like the λ_f^{region}-calculus, an expression is essentially characterized by its value. Hence, rule *(Obs-Red)* says that the observations of a closed λ_f^{region}-Term are the observations one can make of its value, provided it exists.

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

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

$$\textit{(Obs-Fix)}\quad \frac{S_r = \{\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n\} \qquad \{\,\} \vdash_f \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e\rangle_\varrho : (\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \varrho), \emptyset}{(\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e\rangle_\varrho : (\forall\, \varrho_1, \ldots, \varrho_n\, .\tau, \varrho), \varphi_0) \xrightarrow{[\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho'} (\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e\rangle_\varrho\, [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho' : (S_r(\tau), \varrho'), \{\varrho, \varrho'\})}$$

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

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

$$\textit{(Obs-Fixdead)}\quad \frac{\{\,\} \vdash_f \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e\rangle_\varrho : (\overline{\sigma}, \varrho), \emptyset}{(\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e\rangle_\varrho : (\overline{\sigma}, \varrho), \varphi_0) \xrightarrow{[\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \bullet} (\Omega_{\varrho_\star} : (\alpha, \varrho_\star), \{\varrho_\star\})}$$

Figure 7: Labeled transitions for the λ_f^{region}-calculus

The rules *(Obs-Lam)* and *(Obs-Fix)* observe function application and region application respectively, provided the objects are not deallocated. In *(Obs-Lam)*, 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.

Since we want to make an observation of *every* well-typed value, we have the rules *(Obs-Dealloc)* and *(Obs-Dang)*, which cater for values not covered by *(Obs-Lam)* and *(Obs-Fix)*. The transition rule *(Obs-Fixdead)* allows the observation $[\rho_1, \ldots, \rho_n]\ \mathtt{at}\ \bullet$.

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 $\Omega_{\varrho_\star}$ which disallows further observations. The region variable $\varrho_\star$ is arbitrary and not dead, but fixed throughout the rest of the paper to guarantee a deterministic transition system. The effect φ_0 in the rules which observe values is arbitrary since values do not have an effect.

Note that the two observation rules *(Obs-Lam)* and *(Obs-Fix)* apply the context rules (9) and (11) from Figure 3 respectively - except for the value in *(Obs-Lam)*. Continuing along these lines, one could include labeled transitions 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 [16,17] and would provide an alternative formulation of Milner's Context Lemma [34].

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

Lemma 6.1 *Suppose* $\{\,\} \vdash_f e : \pi, \varphi''$ *where* $\varphi'' \subseteq \varphi$ *and* $e \downarrow$. *Then there exists an observation* γ *such that the transition* $(e : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$ *exists. Moreover, if Safe* (γ), *we have for* $e \overset{*}{\rightharpoonup} v$ *that* $e' = v\gamma$.

Proof. By induction on the number of reductions in $e \overset{*}{\rightharpoonup} v$. If e is the value v, then one of the rules *(Obs-Lam)*, *(Obs-Fix)*, *(Obs-Dealloc)* or *(Obs-Dang)* applies, depending on the actual form of π. Whenever *Safe* (γ), we also have $e' = v\gamma$ for $e \overset{*}{\rightharpoonup} v$ by the rules *(Obs-Lam)* and *(Obs-Fix)*. Alternatively, we have $e \rightharpoonup e''$ where $e'' \overset{*}{\rightharpoonup} v$. By Proposition 2.3, 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 $(e : \pi, \varphi) \xrightarrow{\gamma} (e' : \pi', \varphi')$. $\square$

The opposite property also holds:

Lemma 6.2 *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* (γ), *it also holds that* $e' = v\gamma$ *for* $e \overset{*}{\rightharpoonup} v$.

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-Lam)*, *(Obs-Fix)*, *(Obs-Dealloc)*, *(Obs-Dang)* or *(Obs-Fixdead)*, we immediately have $e \downarrow$. If *Safe* (γ), it also holds that $e' = v\gamma$ for $e \overset{*}{\rightharpoonup} v$ by the rules *(Obs-Lam)* 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 $e \downarrow$. $\square$

A transition preserves typing and guarantees a clean result effect:

Lemma 6.3 *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, using type preservation for *(Obs-Red)*. □

The transition system is deterministic:

Lemma 6.4 *Whenever* $(e_1 : \pi_1, \varphi_1) \xrightarrow{\gamma} (e_2 : \pi_2, \varphi_2)$ *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 2.7. □

From Lemma 6.1, we already have that convergence implies a transition. We can also choose it to coincide with the observation of another expression of the same type.

Lemma 6.5 *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* $e_2 \stackrel{*}{\rightharpoonup} v_2$.

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. □

6.2 Simulation and Bisimulation

For every well-typed λ_f^{region}-program, the labeled transition system produces a derivation 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 6.1 *The functions* $\lceil . \rceil$, $[.] \in \mathsf{ClosedTypedRel} \rightarrow \mathsf{ClosedTypedRel}$ *are defined as follows:*

$$
\begin{aligned}
\lceil \mathcal{R} \rceil = \{(\{\,\}, e_1, e_2, \pi, \varphi) \mid\ & \textit{whenever for all } S_r \textit{ with } Clean\,(S_r(\varphi)) \textit{ the transition} \\
& (S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_1' : \pi', \varphi') \textit{ implies} \\
& \textit{the existence of an expression } e_2' \textit{ such that} \\
& (S_r(e_2) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_2' : \pi', \varphi') \textit{ and} \\
& \{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi' \cup \varphi'' \textit{ with } Clean\,(\varphi'')\}
\end{aligned}
$$

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

The operators $\lceil . \rceil$ and $\lfloor . \rfloor$ are monotone in the complete lattice $(\mathsf{ClosedTypedRel}, \subseteq)$ and therefore lead to the notions of similarity and bisimilarity:

Definition 6.2 *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 $\lfloor . \rfloor$-dense, i.e. $\mathcal{R} \subseteq \lfloor \mathcal{R} \rfloor$, *is a* bisimulation. *The union of all bisimulations is the* Bisimilarity *relation, $\approx$.*

By Lemma 6.3, (bi)similarity is well-defined. The similarity and bisimilarity relations are non-empty since they are reflexive as we shall see below.

Expanding the definition of simulation yields the following more manageable formulation: 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 for any S_r such that *Clean* $(S_r(\varphi))$ the transition $(S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\ \gamma\ } (e_1' : \pi', \varphi')$ exists, then $(S_r(e_2) : S_r(\pi), S_r(\varphi)) \xrightarrow{\ \gamma\ } (e_2' : \pi', \varphi')$ must exist such that $\{\ \} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi' \cup \varphi''$ with *Clean* (φ'').

In contrast with the more usual definition of bisimilarity [17], the operators $\lceil . \rceil$ and $\lfloor . \rfloor$ quantify over a region substitution S_r before considering a transition. This is due to the fact that the λ_f^{region}-calculus is region-polymorphic and therefore allows any free region variable ϱ to be abstracted by a context.

We explain this with an example. Consider the type $\pi = ((\pi_1 \xrightarrow{\epsilon.\emptyset} \pi_2, \varrho) \xrightarrow{\epsilon.\emptyset} (\alpha, \varrho), \varrho)$ such that

$$\pi_1 = ((\alpha, \varrho) \xrightarrow{\epsilon.\emptyset} ((\alpha, \varrho) \xrightarrow{\epsilon.\emptyset} (\alpha, \varrho), \varrho_3), \varrho_1)$$

and

$$\pi_2 = ((\alpha, \varrho) \xrightarrow{\epsilon.\emptyset} ((\alpha, \varrho) \xrightarrow{\epsilon.\emptyset} (\alpha, \varrho), \varrho_3), \varrho_2)$$

In a monomorphic subset of the λ_f^{region}-calculus, the expression

$$\lambda\, f.\, (f \mathbin{@} \langle \lambda\, w.\, \langle \lambda\, z.\, w \rangle_{\varrho_3} \rangle_{\varrho_1})\ \mathtt{at}\ \varrho$$

of type π is contextually equivalent to the expression

$$\lambda\, f.\, (f \mathbin{@} \langle \lambda\, w.\, \langle \lambda\, z.\, z \rangle_{\varrho_3} \rangle_{\varrho_1})\ \mathtt{at}\ \varrho$$

of type π. Intuitively, this is because the only values of type $(\pi_1 \xrightarrow{\epsilon.\emptyset} \pi_2, \varrho)$ have to be functions that forget their arguments. Indeed, it is not possible to move a value from one region (in this case ϱ_1) to another region (in this case ϱ_2) without effect (the latent effect is empty in our example).

However, in the polymorphic λ_f^{region}-calculus, it is possible that a context abstracts over the variables ϱ_1 and ϱ_2. At region application time, these variables may be mapped onto the same region variable, say ϱ'. But this would make it possible to apply the above expressions to the identity function $\langle \lambda\, x.\, x \rangle_{\varrho'}$, which yields different results. Therefore, a notion of bisimilarity which does not include a quantification over region substitutions renders the above expressions bisimilar, even though they are not contextual equivalent in a polymorphic region calculus.

Finally, we also note that in the definitions $\lceil . \rceil$ and $\lfloor . \rfloor$, we require *Clean* $(S_r(\varphi))$ for the effect of the transition. We need this condition to make sure that whenever a transition if possible at all, it is possible for all S_r too. Without *Clean* $(S_r(\varphi))$ we cannot guarantee that rule *(Obs-Red)* remains applicable for all S_r.

Knaster-Tarski now says that similarity is the greatest simulation and bisimilarity the greatest bisimulation, leading to the following co-inductive proof principles, 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 fixpoints for $\lceil . \rceil$ and $[.]$ respectively, *i.e.* $\lceil \lesssim \rceil = \lesssim$ and $[\approx] = \approx$.

Sometimes, it is helpful to use the following refined notion of bisimilarity, which is due to Milner [17,35]. The closed typed relation $\mathcal{R}$ is a bisimulation *up to* $\approx$, provided $\mathcal{R} \subseteq [\approx \mathcal{R} \approx]$, where we write $\mathcal{R}\mathcal{R}'$ for the composition of two typed relations $\mathcal{R}$ and $\mathcal{R}'$.

6.3 Properties of Simulations

A bisimulation up to $\approx$ is also a bisimulation:

Proposition 6.6 *Whenever $\mathcal{R}$ is a bisimulation up to $\approx$, then $\approx \mathcal{R} \approx \subseteq [\approx \mathcal{R} \approx]$*

Proof. See Milner [35] $\square$

Recall from Section 3 that we do not in general have $\{\ \} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ iff $\{\ \} \rhd e_1 \,\mathcal{R}^\circ\, e_2 : \pi, \varphi$. However, with the above notion of bisimilarity, this property does hold for similarity and bisimilarity. To prove this, consider the following substitution property:

Lemma 6.7 *Suppose $\{\ \} \rhd e_1 \lesssim e_2 : \pi, \varphi$ and $\mathit{Clean}\,(S_r(\varphi))$. Then $\{\ \} \rhd S_r(e_1) \lesssim S_r(e_2) : S_r(\pi), S_r(\varphi)$.*

Proof. We prove by co-induction, so take the relation

$$\mathcal{R} = \big\{(\{\ \}, S_r(e_1), S_r(e_2), S_r(\pi), S_r(\varphi)) \mid \{\ \} \rhd e_1 \lesssim e_2 : \pi, \varphi \textit{ with } \mathit{Clean}\,(S_r(\varphi))\big\}$$

By Lemma 2.1, we obviously have that $\mathcal{R}$ is well-formed, so it is sufficient to prove that $\mathcal{R} \subseteq \lceil \lesssim \rceil$.

So, we assume that $\{\ \} \rhd S_r(e_1) \,\mathcal{R}\, S_r(e_2) : S_r(\pi), S_r(\varphi)$ and consider the transition $((S'_r \circ S_r)(e_1) : (S'_r \circ S_r)(\pi), (S'_r \circ S_r)(\varphi)) \xrightarrow{\gamma'} (e'_1 : \pi', \varphi')$ where $\mathit{Clean}\,(S'_r(\varphi))$. Now, we obviously have $\mathit{Clean}\,((S'_r \circ S_r)(\varphi))$ as well, and since $\{\ \} \rhd e_1 \lesssim e_2 : \pi, \varphi$ holds, we have the transition $((S'_r \circ S_r)(e_2) : (S'_r \circ S_r)(\pi), (S'_r \circ S_r)(\varphi)) \xrightarrow{\gamma'} (e'_2 : \pi', \varphi')$ such that $\{\ \} \rhd e'_1 \lesssim e'_2 : \pi', \varphi'$ by the definition of simulation. $\square$

With this lemma, we also have the following property for (bi)similarity:

Proposition 6.8 $\{\ \} \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$ *iff* $\{\ \} \rhd e_1 \lesssim e_2 : \pi, \varphi$

Proof. From left-to-right, we can take the region substitution I_r in the definition of open extension. From right-to-left, we apply Lemma 6.7 for any S_r with $\mathit{Clean}\,(S_r(\varphi))$. We conclude by the definition of open extension. $\square$

We will often use this result implicitly in the rest of the paper. The substitution lemma also holds for open extensions.

Proposition 6.9 *Suppose* $TE \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$, *then* $S_r(TE) \rhd S_r(e_1) \lesssim^\circ S_r(e_2) : S_r(\pi), S_r(\varphi)$.

Proof. By definition of open extension, we have to prove that for all S'_r and ξ such that $S'_r(S_r(TE)) \succ \xi$ where $Clean\,(S'_r(S_r(\varphi)))$ it holds that $\{\,\} \rhd \xi((S'_r \circ S_r)(e_1)) \lesssim \xi((S'_r \circ S_r)(e_2)) : (S'_r \circ S_r)(\pi), (S'_r \circ S_r)(\varphi)$. The property now holds immediately since we have by assumption that $TE \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$ and can therefore apply the definition of open extension with region substitution $S'_r \circ S_r$. □

Proposition 6.10 *The open extension* $\approx^\circ$ *of bisimilarity is a well-formed equivalence relation:*

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

Proof.

1. Follows immediately by definition of bisimilarity and open extension.
2. To show reflexivity, it is easy to see by definition of simulation that the relation

$$\{(\{\,\}, e, e, \pi, \varphi) \mid \{\,\} \vdash_f e : \pi, \varphi' \textit{ with } \varphi' \subseteq \varphi \textit{ and } Clean\,(\varphi)\}$$

is well-formed and $\lceil . \rceil$-dense, using Lemma 6.3 and Lemma 2.1. Hence, by co-induction, $\lesssim$ is reflexive. The property holds now trivially for $\lesssim^\circ$ as well.

For transitivity, assume

$$\mathcal{R} = \{(\{\,\}, e_1, e_3, \pi, \varphi) \mid \begin{array}{l} \textit{there exists } e_2 \textit{ such that } \{\,\} \rhd e_1 \lesssim e_2 : \pi, \varphi \textit{ and} \\ \{\,\} \rhd e_2 \lesssim e_3 : \pi, \varphi\} \end{array}$$

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 the transition $(S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e'_1 : \pi', \varphi')$ for $Clean\,(S_r(\varphi))$ implies the transition $(S_r(e_2) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e'_2 : \pi', \varphi')$ such that $\{\,\} \rhd e'_1 \lesssim e'_2 : \pi', \varphi' \cup \varphi''$ with $Clean\,(\varphi'')$. Therefore, the transition $(S_r(e_3) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e'_3 : \pi', \varphi')$ holds 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 $\lesssim^\circ$.
3. Reflexivity and transitivity follow by Item 2. Symmetry is obtained by a similar co-induction argument using the definition of bisimilarity.

□

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

Proposition 6.11 *$TE \rhd e_1 \approx^\circ e_2 : \pi, \varphi$ 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 6.10.3, also $\approx \subseteq \lesssim^{-1}$, implying $\approx \subseteq \lesssim \cap \lesssim^{-1}$.

For the other direction, take the well-formed typed relation

$$\mathcal{R} = \lesssim \cap \lesssim^{-1} = \left\{(\{\ \}, e_1, e_2, \pi, \varphi) \mid \{\ \} \rhd e_1 \lesssim e_2 : \pi, \varphi \text{ and } \{\ \} \rhd e_2 \lesssim e_1 : \pi, \varphi\right\}$$

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 the transition $(S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$ where *Clean* $(S_r(\varphi))$. By definition of $\mathcal{R}$, there exists an e_2' such that $(S_r(e_2) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$ and $\{\ \} \rhd e_1' \lesssim e_2' : \pi', \varphi' \cup \varphi''$ with *Clean* (φ''). Also, we have that $\{\ \} \rhd e_2 \lesssim e_1 : \pi, \varphi$, so there exists an e_1'' such that $(S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_1'' : \pi', \varphi')$ and $\{\ \} \rhd e_2' \lesssim e_1'' : \pi', \varphi' \cup \varphi''$. By Lemma 6.4, 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:

$$\begin{gathered}\{\ \} \rhd e_1 \approx e_2 : \pi, \varphi \\ \textit{iff} \\ \textit{for all } S_r \textit{ such that Clean } (S_r(\varphi)) \textit{ it holds that} \\ (S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_1' : \pi', \varphi') \Leftrightarrow (S_r(e_2) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_2' : \pi', \varphi') \\ \textit{where } \{\ \} \rhd e_1' \approx e_2' : \pi', \varphi' \cup \varphi'' \textit{ with Clean } (\varphi'')\end{gathered}$$

Henceforth, we will assume that application of Proposition 6.11 is implicit.

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

Lemma 6.12 *Suppose $TE \rhd e \lesssim^\circ e' : \pi, \varphi$. If $e \downarrow$ then $e' \downarrow$.*

Proof. By open extension, we take ξ and S_r such that $S_r(TE) \succ \xi$ with *Clean* $(S_r(\varphi))$ and have $\{\ \} \rhd \xi(S_r(e)) \lesssim \xi(S_r(e')) : S_r(\pi), S_r(\varphi)$. By Lemma 2.8 and Lemma 2.9 we now have that $\xi(S_r(e)) \downarrow$, so using Lemma 3.1 and Lemma 6.1, we have a transition from $\xi(S_r(e))$. By the definition of similarity therefore also from $\xi(S_r(e'))$. Lemma 3.1 and Lemma 6.2 prove the claim. □

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

Lemma 6.13 *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 6.2. □

The left and right-hand side of a λ_f^{region}-reduction are bisimilar:

Lemma 6.14 *Suppose* $TE \vdash_f e : \pi, \varphi'$ *and* $e \rightharpoonup e'$ *where* $\varphi' \subseteq \varphi$ *and* $Clean(\varphi)$. *Then* $TE \rhd e \approx^\circ e' : \pi, \varphi$.

Proof. We prove the lemma for $\approx$ since it trivially holds for $\approx^\circ$ by Lemma 2.9. So define

$$\mathcal{R} = \big\{(\{\,\}, e, e', \pi, \varphi) \mid \{\,\} \vdash_f e : \pi, \varphi' \textit{ and } e \rightharpoonup e' \textit{ where } \varphi' \subseteq \varphi \textit{ with } Clean(\varphi)\big\}$$

By Proposition 2.3, 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, assume that $(S_r(e) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_0 : \pi', \varphi')$ such that $Clean(S_r(\varphi))$. Since $S_r(e)$ cannot be a value, the transition must be due to rule *(Obs-Red)* and therefore by Lemma 2.7 and Lemma 2.9 we have $(S_r(e') : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_0 : \pi', \varphi')$. By Lemma 6.3, we have that $\{\,\} \vdash_f e_0 : \pi', \varphi'$ with $Clean(\varphi')$ and hence by reflexivity of $\approx$, also that $\{\,\} \rhd e_0 \approx e_0 : \pi', \varphi' \cup \varphi''$ with $Clean(\varphi'')$. Hence, $\mathcal{R} \subseteq \lceil\approx\rceil$.

The other way around, assume transition $(S_r(e') : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_0' : \pi', \varphi')$ where $Clean(S_r(\varphi))$. Then, immediately by rule *(Obs-Red)* and Lemma 2.9, we have the transition $(S_r(e) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_0' : \pi', \varphi')$. Because of Lemma 6.3, we also have $\{\,\} \vdash_f e_0' : \pi', \varphi'$ with $Clean(\varphi')$ and therefore by reflexivity that $\{\,\} \rhd e_0' \approx e_0' : \pi', \varphi' \cup \varphi''$ with $Clean(\varphi'')$, implying $\mathcal{R} \subseteq \lceil\approx^{-1}\rceil^{-1}$. □

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 [15–17]. 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 $\lesssim^*$ of $\lesssim$ is inductively defined as the least well-formed typed relation closed under the following rule:

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

We have the following auxiliary result:

Lemma 7.1 *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 rule *(Def-Comp-Ext)* there exists an e_2' such that $TE \rhd e_1 \widehat{\lesssim^*} e_2' : \pi, \varphi$ 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 $\lesssim^*$ has the following basic properties:

Proposition 7.2 *1.* $\lesssim^*$ *is compatible*

2. $\lesssim^*$ *is reflexive*

3. $\lesssim^\circ \subseteq \lesssim^*$

Proof.

1. Take $TE \rhd e_1 \widehat{\lesssim^*} e_2 : \pi, \varphi$. Since $\lesssim^*$ is well-formed by definition, we have by Lemma 4.1.2 that $\widehat{\lesssim^*}$ is well-formed and hence, $TE \vdash_f e_2 : \pi, \varphi'$ with $\varphi' \subseteq \varphi$ and $Clean(\varphi)$. 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 4.1.1 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 $Clean(\varphi)$. So by Item 2 this implies that $TE \rhd e_2 \lesssim^* e_2 : \pi, \varphi$. Using Lemma 7.1 this yields $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$, proving the claim.

□

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

Lemma 7.3 *If* $TE \rhd v_1 \lesssim^* v_2 : (\overline{\sigma}, \rho), \emptyset$ *and* $TE + \{y \mapsto (\forall \vec{\alpha}.\forall \vec{\epsilon}.\overline{\sigma}, \rho)\} \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ *with* $(\{\vec{\alpha}\} \cup \{\vec{\epsilon}\}) \cap fv(TE) = \emptyset$, *then* $TE \rhd e_1[y \mapsto v_1] \lesssim^* e_2[y \mapsto v_2] : \pi, \varphi$.

Proof. So we have $TE+\{y \mapsto (\forall \vec{\alpha}.\forall \vec{\epsilon}.\overline{\sigma}, \rho)\} \rhd e_1 \widehat{\lesssim^*} e_3 : \pi, \varphi$ and $TE+\{y \mapsto (\forall \vec{\alpha}.\forall \vec{\epsilon}.\overline{\sigma}, \rho)\} \rhd e_3 \lesssim^\circ e_2 : \pi, \varphi$. By Lemma 3.2 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 by applying Lemma 7.1. 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 7.2.1. □

Additionally, the relation $\lesssim^*$ obeys a region substitution lemma (similar to Proposition 6.9).

Lemma 7.4 *Suppose* $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$. *Then* $S_r(TE) \rhd S_r(e_1) \lesssim^* S_r(e_2) : S_r(\pi), S_r(\varphi)$.

Proof. So, we have by *(Def-Comp-Ext)* that

$$TE \rhd e_1 \widehat{\lesssim^*} e_3 : \pi, \varphi \tag{15}$$

and $TE \rhd e_3 \lesssim^\circ e_2 : \pi, \varphi$. Applying Proposition 6.9 on the latter yields $S_r(TE) \rhd S_r(e_3) \lesssim^\circ S_r(e_2) : S_r(\pi), S_r(\varphi)$. It remains to show that $S_r(TE) \rhd S_r(e_1) \lesssim^* S_r(e_3) : S_r(\pi), S_r(\varphi)$, since we can then use Lemma 7.1 to obtain the desired result. The claim is proven by a straightforward induction on the compatible refinement, using Proposition 7.2.1. □

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

Lemma 7.5 *Suppose* $TE \rhd e \lesssim^* e' : \pi, \varphi$. *If* $e \downarrow$ *then* $e' \downarrow$.

Proof. By *(Def-Comp-Ext)* we have $TE \rhd e \widehat{\lesssim^*} e'' : \pi, \varphi$ 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 6.12. □

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

Lemma 7.6 *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 $\lambda\, x.\, e$ $\texttt{at}\, \varrho \rightharpoonup \langle \lambda\, x.\, e\rangle_\varrho$**.** So we have that $\{\,\} \rhd \lambda\, x.\, e\ \texttt{at}\, \varrho \widehat{\lesssim^*} e_3 : \pi, \varphi$ and

$$\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi \tag{16}$$

By rule *(Comp-Lam)* we have that $\pi = \pi_1 \xrightarrow{\epsilon.\varphi'} \pi_2$ and $e_3 = \lambda\, x.\, e'\ \texttt{at}\, \varrho$. Additionally we have $\varrho \in \varphi$ and $\{x \mapsto \pi_1\} \rhd e \lesssim^* e' : \pi_2, \varphi'$. By Lemma 6.14 the similarity $\{\,\} \rhd \langle \lambda\, x.\, e'\rangle_\varrho \lesssim \lambda\, x.\, e'\ \texttt{at}\, \varrho : \pi, \varphi$ holds, and hence, by (16) and transitivity of $\lesssim$ also that

$$\{\,\} \rhd \langle \lambda\, x.\, e'\rangle_\varrho \lesssim e_2 : \pi, \varphi \tag{17}$$

Since the premises of *(Comp-Lam)* and *(Comp-Lamv)* are identical, we also have $\{\,\} \rhd \langle \lambda\, x.\, e\rangle_\varrho \widehat{\lesssim^*} e_3 : \pi, \varphi$, which combined with (17) and *(Def-Comp-Ext)* makes $\{\,\} \rhd \langle \lambda\, x.\, e\rangle_\varrho \lesssim^* e_2 : \pi, \varphi$.

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

Case $\texttt{new}\, \varrho.\, v \rightharpoonup v[\varrho \mapsto \bullet]$**.** Hence, $\{\,\} \rhd \texttt{new}\, \varrho.\, v \widehat{\lesssim^*} e_3 : \pi, \varphi$ and $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$. By rule *(Comp-Reglam)*, it must be that $e_3 = \texttt{new}\, \varrho.\, e_3''$ where $\{\,\} \rhd v \lesssim^* e_3'' : \pi, \varphi$ and $\varrho \notin \mathit{frv}\,(\pi)$. Using Lemma 7.5, we have that $e_3'' \overset{*}{\rightharpoonup} v_3$ and by reduction rule (3) also that $\texttt{new}\, \varrho.\, e_3'' \overset{*}{\rightharpoonup} v_3[\varrho \mapsto \bullet]$. Hence, by Lemma 6.14, 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 7.1 on the other make

$$\{\,\} \rhd v_3[\varrho \mapsto \bullet] \lesssim e_2 : \pi, \varphi \tag{18}$$

and $\{\,\} \rhd v \lesssim^* v_3 : \pi, \emptyset$. Applying Lemma 7.4 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 (18) using Lemma 7.1 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 $\{\,\} \rhd \langle \lambda\, x.\, e\rangle_\varrho @\, v \widehat{\lesssim^*} e_3 : \pi, \varphi$ 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

$$\{x \mapsto \pi'\} \rhd e \lesssim^* e'' : \pi, \varphi_2 \tag{19}$$

Using Lemma 7.5 it also holds that $e' \overset{*}{\rightharpoonup} v'$ and by Lemma 6.14 we have $\{\,\} \rhd e' \lesssim v' : \pi', \varphi_1$. From Lemma 7.1 this implies $\{\,\} \rhd v \lesssim^* v' : \pi', \varphi_1$. The latter, (19) and Lemma 7.3 imply

$$\{\,\} \rhd e[x \mapsto v] \lesssim^* e''[x \mapsto v'] : \pi, \varphi_2 \tag{20}$$

Since we have that $\langle \lambda\, x.\, e''\rangle_\varrho @\, e' \overset{*}{\rightharpoonup} e''[x \mapsto v']$, 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 (20) and Lemma 7.1 proves the case.

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

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

Now, we have that $\{\ \} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e\rangle_\varrho\ [\rho'_1,\ldots,\rho'_n]\ \mathtt{at}\ \varrho' \widehat{\lesssim}^* e_3 : \pi, \varphi$ 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 $\pi = (S_r(\tau), \varrho')$; the substitution $S_r = \{\varrho_1 \mapsto \rho'_1,\ldots,\varrho_n \mapsto \rho'_n\}$ and $\overline{\sigma} = \forall\, \varrho_1,\ldots,\varrho_n\, .\tau$. Additionally, we also have $\{\ \} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e\rangle_\varrho \lesssim^* e'_3 : (\overline{\sigma}, \varrho), \varphi'$ with $\varphi' \cup \left\{\varrho, \varrho'\right\} \subseteq \varphi$.

Using Lemma 7.5, we have that $e'_3 \xrightharpoonup{*} v_3$ and by the canonical forms Lemma 2.4 also that v_3 has the form $\langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e'\rangle_\varrho$. Using Lemma 6.14 and Lemma 7.1 we therefore have

$$\begin{array}{l}\{\ \} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e\rangle_\varrho \lesssim^* \\ \qquad \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e'\rangle_\varrho : (\overline{\sigma}, \varrho), \varphi'\end{array} \tag{21}$$

Since $e_3 \xrightharpoonup{*} \langle \lambda\, x.\ S_r(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e'\rangle_\varrho]\rangle_{\varrho'}$ too, we also obtain with Lemma 6.14 that

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

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

$$\{\ \} \rhd \langle \lambda\, x.\ S_r(e')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e'\rangle_\varrho]\rangle_{\varrho'} \lesssim e_2 : \pi, \varphi \tag{22}$$

By *(Def-Comp-Ext)*, we have from (21) that

$$\{\ \} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e\rangle_\varrho \widehat{\lesssim}^* e_4 : (\overline{\sigma}, \varrho), \varphi' \tag{23}$$

and

$$\{\ \} \rhd e_4 \lesssim \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e'\rangle_\varrho : (\overline{\sigma}, \varrho), \varphi' \tag{24}$$

From *(Comp-Fixv)*, the congruence (23) implies that e_4 must be a recursive function object of the form $\langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e''\rangle_\varrho$ with $\varrho \notin \left\{\varrho_1,\ldots,\varrho_n\right\}$ and also $\{f \mapsto (\forall\, \vec{\epsilon}.\overline{\sigma}, \varrho)\} \rhd \langle \lambda\, x.\, e\rangle_\varrho \lesssim^* \langle \lambda\, x.\, e''\rangle_\varrho : (\tau, \varrho), \emptyset$. The latter on its turn yields with *(Def-Comp-Ext)* and *(Comp-Lamv)* that $\{f \mapsto (\forall\, \vec{\epsilon}.\overline{\sigma}, \varrho)\} \rhd \langle \lambda\, x.\, e\rangle_\varrho \widehat{\lesssim}^* \langle \lambda\, x.\, e'''\rangle_\varrho : (\tau, \varrho), \emptyset$ and $\{f \mapsto (\forall\, \vec{\epsilon}.\overline{\sigma}, \varrho)\} \rhd \langle \lambda\, x.\, e'''\rangle_\varrho \lesssim^\circ \langle \lambda\, x.\, e''\rangle_\varrho : (\tau, \varrho), \emptyset$. The compatibility rule *(Comp-Lamv)*, the definition of open extension, the transition rule *(Obs-Lam)* and the type rule *(F-Lamv)* make that $\{f \mapsto (\forall\, \vec{\epsilon}.\overline{\sigma}, \varrho)\} \rhd \langle \lambda\, x.\, e\rangle_{\varrho'} \widehat{\lesssim}^* \langle \lambda\, x.\, e'''\rangle_{\varrho'} : (\tau, \varrho'), \emptyset$ and $\{f \mapsto (\forall\, \vec{\epsilon}.\overline{\sigma}, \varrho)\} \rhd \langle \lambda\, x.\, e'''\rangle_{\varrho'} \lesssim^\circ \langle \lambda\, x.\, e''\rangle_{\varrho'} : (\tau, \varrho'), \emptyset$ also holds. Recombining with *(Def-Comp-Ext)* yields $\{f \mapsto (\forall\, \vec{\epsilon}.\overline{\sigma}, \varrho)\} \rhd \langle \lambda\, x.\, e\rangle_{\varrho'} \lesssim^* \langle \lambda\, x.\, e''\rangle_{\varrho'} : (\tau, \varrho'), \emptyset$. Using Lemma 7.4 with S_r and Lemma 7.3 with (23) on this assertion, we obtain

$$\begin{array}{l}\{\ \} \rhd \langle \lambda\, x.\ S_r(e)[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e\rangle_\varrho]\rangle_{\varrho'} \lesssim^* \\ \qquad \langle \lambda\, x.\ S_r(e'')[f \mapsto \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e''\rangle_\varrho]\rangle_{\varrho'} : \pi, \emptyset\end{array} \tag{25}$$

But we also have by (24) and the definition of similarity (using *(Obs-Fix)*) that

$$\begin{array}{l}\{\ \} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e''\rangle_\varrho\, [\rho_1,\ldots,\rho_n]\ \mathtt{at}\ \varrho' \lesssim \\ \qquad \langle \mathtt{fix}\, f.\Lambda\, \varrho_1,\ldots,\varrho_n.\lambda\, x.\, e'\rangle_\varrho\, [\rho_1,\ldots,\rho_n]\ \mathtt{at}\ \varrho' : (\tau, \varrho'), \emptyset\end{array}$$

Using Lemma 6.14 and transitivity of $\lesssim$, we obtain

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

In combination with (25) via Lemma 7.1 this yields

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

By combining with (22) using Lemma 7.1, we prove the case.

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

$$\{\ \} \rhd e_3 \lesssim e_2 : \pi, \varphi \tag{26}$$

By *(Comp-App)*, e_3 must be an application of the form $e_0 \mathbin{@} e_0''$ 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 $\{\ \} \rhd e' \mathbin{@} e'' \widehat{\lesssim^*} e_0 \mathbin{@} e_0'' : \pi, \varphi$ and combining this with (26) and *(Def-Comp-Ext)* the case follows.

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

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

Lemma 7.7 *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 the relation

$$\mathcal{R} = \big\{(\{\ \}, e_1, e_2, \pi, \varphi) \mid \{\ \} \rhd e_1 \lesssim^* e_2 : \pi, \varphi\big\}$$

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 a transition $(S_r'(e_1) : S_r'(\pi), S_r'(\varphi)) \xrightarrow{\ \gamma\ } (e_1' : \pi', \varphi')$ such that *Clean* $(S_r'(\varphi))$.

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 7.6 to obtain $\{\ \} \rhd e_1'' \lesssim^* e_2 : \pi, \varphi$.

Hence, $\{\ \} \rhd e_1'' \,\mathcal{R}\, e_2 : \pi, \varphi$ and so, by induction and Lemma 2.9, there is a labeled transition $(S_r'(e_2) : S_r'(\pi), S_r'(\varphi)) \xrightarrow{\ \gamma\ } (e_2' : \pi', \varphi')$ such that $(\{\ \}, e_1', e_2', \pi', \varphi' \cup \varphi'') \in \mathcal{R} \cup \lesssim$ with *Clean* (φ''), proving the case.

Case *(Obs-Lam)***.** 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 *Clean* (φ'), the observation $\gamma = @\, v$, and $e_1' = S_r'(\langle \lambda\, x.\ e_1'' \rangle_{\varrho}) \mathbin{@} v$.

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 $\{\ \} \rhd \langle \lambda\, x.\ e_1'' \rangle_{\varrho} \widehat{\lesssim^*} \langle \lambda\, x.\ e_3'' \rangle_{\varrho} : \pi, \varphi$ and $\{\ \} \rhd \langle \lambda\, x.\ e_3'' \rangle_{\varrho} \lesssim e_2 : \pi, \varphi$. Hence, by Proposition 7.2.1, we also have $\{\ \} \rhd \langle \lambda\, x.\ e_1'' \rangle_{\varrho} \lesssim^* \langle \lambda\, x.\ e_3'' \rangle_{\varrho} : \pi, \varphi$ and therefore by Lemma 7.4 that $\{\ \} \rhd S_r'(\langle \lambda\, x.\ e_1'' \rangle_{\varrho}) \lesssim^* S_r'(\langle \lambda\, x.\ e_3'' \rangle_{\varrho}) : S_r'(\pi), S_r'(\varphi)$ too. Also, by Proposition 7.2.2, we have $\{\ \} \rhd v \lesssim^* v : S_r'(\pi_1), \emptyset$. Combining this with *(Comp-App)* we obtain that

$$\{\ \} \rhd S_r'(\langle \lambda\, x.\ e_1'' \rangle_{\varrho}) \mathbin{@} v \,\widehat{\lesssim^*}\, S_r'(\langle \lambda\, x.\ e_3'' \rangle_{\varrho}) \mathbin{@} v : S_r'(\pi_2'), S_r'(\varphi'') \tag{27}$$

where $S'_r(\varphi') \cup \{\epsilon, S'_r(\varrho)\} \subseteq S'_r(\varphi'')$ and $\mathit{Clean}\,(S'_r(\varphi''))$. From Lemma 6.5, if follows that transition $(S'_r(\langle\lambda\, x.\, e''_3\rangle_\varrho) : S'_r(\pi), S'_r(\varphi)) \xrightarrow{@\,v} (\langle\lambda\, x.\, e''_3\rangle_\varrho\,@\,v : \pi'_2, \varphi'')$ 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 $\{\,\} \rhd S'_r(\langle\lambda\, x.\, e''_3\rangle_\varrho)\,@\,v \lesssim S'_r(e_2)\,@\,v : S'_r(\pi'_2), S'_r(\varphi'')$. This, together with (27) 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 with place π is $(\forall\, \varrho_1, \ldots, \varrho_n\,.\tau, \varrho)$. So, the observation γ is $[\rho'_1, \ldots, \rho'_n]$ $\mathtt{at}\ \varrho'$ and the expression e'_1 is $S'_r(e_1)\ [\rho'_1, \ldots, \rho'_n]$ $\mathtt{at}\ \varrho'$. The type π' is $(S_r(S'_r(\tau)), \varrho')$ where the region substitution $S_r = \{\varrho_1 \mapsto \rho'_1, \ldots, \varrho_n \mapsto \rho'_n\}$.

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 $\{\,\} \rhd \langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e''_1\rangle_\varrho \widehat{\lesssim^*} e_3 : \pi, \varphi$ 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 7.2.1, Lemma 7.4 and *(Comp-Regapp)* we have that

$$\begin{aligned}\{\,\} \rhd\ & S'_r(\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e''_1\rangle_\varrho)\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho' \lesssim^* \\ & S'_r(\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e''_3\rangle_\varrho)\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho' : (\pi_0, \varrho'), \varphi''\end{aligned} \tag{28}$$

where $\pi_0 = S_r(S'_r(\tau))$, $\{S'_r(\varrho), \varrho'\} \subseteq \varphi''$ and $\mathit{Clean}\,(\varphi'')$. Since the transition

$$\begin{aligned}(S'_r(\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e''_3\rangle_\varrho) : S'_r(\forall\, \varrho_1, \ldots, \varrho_n\,.\tau, \varrho), S'_r(\varphi)) &\xrightarrow{\gamma} \\ (S'_r(\langle \mathtt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\, e''_3\rangle_\varrho)\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho' : (\pi_0, \varrho'), \varphi'')\end{aligned}$$

is possible by Lemma 6.5, and $\{\,\} \rhd e_3 \lesssim e_2 : \pi, \varphi$ holds too, we have by definition of $\lesssim$ that

$$(S'_r(e_2) : S'_r(\forall\, \varrho_1, \ldots, \varrho_n\,.\tau, \varrho), S'_r(\varphi)) \xrightarrow{\gamma} (e_2\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho' : (\pi_0, \varrho'), \varphi'')$$

and $\{\,\} \rhd S'_r(e_3)\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho' \lesssim e'_2 : (\pi_0, \varrho'), \varphi''$ where we take e'_2 to be the expression $S'_r(e_2)\ [\rho'_1, \ldots, \rho'_n]\ \mathtt{at}\ \varrho'$. Combining this with (28) and Lemma 7.1 proves that $\{\,\} \rhd e'_1 \lesssim^* e'_2 : \pi', \varphi''$ and hence the case.

Cases *(Obs-Dealloc)*, *(Obs-Dang)* and *(Obs-Fixdead)*. Because of Lemma 6.2 we have that $e_1 \downharpoonleft$ and hence by Lemma 7.5 also $e_2 \downharpoonleft$. The transition from e_1 goes to the expression $\Omega_{\varrho_\star}$ and by Lemma 6.5, we therefore have a transition from e_2 to $\Omega_{\varrho_\star}$ as well. Reflexivity of $\lesssim$ proves the case.

□

The actual theorem now follows easily:

Theorem 7.8 (Compatibility of Bisimilarity) $\approx^\circ$ *is compatible.*

Proof. We already have by Proposition 7.2.3 that $\lesssim^\circ \subseteq \lesssim^*$. For the other direction, assume $TE \rhd e_1 \lesssim^* e_2 : \pi, \varphi$ and take a ξ and S_r for which $S_r(TE) \succ \xi$ with $\mathit{Clean}\,(S_r(\varphi))$. Now, by Proposition 7.2.2 and Lemma 7.3, it is easy to prove that $\{\,\} \rhd \xi(S_r(e_1)) \lesssim^* \xi(S_r(e_2)) : S_r(\pi), S_r(\varphi)$. So, from Lemma 7.7 we obtain $\{\,\} \rhd \xi(S_r(e_1)) \lesssim \xi(S_r(e_2)) : S_r(\pi), S_r(\varphi)$, which by definition of completion implies $TE \rhd e_1 \lesssim^\circ e_2 : \pi, \varphi$.

Compatibility of $\approx^\circ$ follows immediately by Proposition 6.11. □

8 The Equational Theory is a Bisimulation

Bisimulation provides a strong proof principle, which we now use to prove that the equational theory of Section 5 is included in bisimilarity. We need two lemmas, which show that the region abstraction $\mathtt{new}\,\varrho.\,e$ is bisimilar to the expression e provided the well-typedness conditions hold. Here we use the refined notion of bisimulation *up to* $\approx$.

We first show the result for bisimilarity:

Lemma 8.1 *Suppose* $\{\,\} \vdash_f e : \pi, \varphi'$ *where* $\varrho \notin \mathit{frv}\,(\pi)$ *and* $\varphi' \subseteq \varphi$ *with* $\mathit{Clean}\,(\varphi)$*, then* $\{\,\} \rhd e \approx \mathtt{new}\,\varrho.\,e : \pi, \varphi$.

Proof. As usual, define the relation

$$\mathcal{R} = \{(\{\,\}, e, \mathtt{new}\,\varrho.\,e, \pi, \varphi) \mid \begin{array}{l} \{\,\} \vdash_f e : \pi, \varphi' \textit{ such that } \varphi' \subseteq \varphi \textit{ with} \\ \mathit{Clean}\,(\varphi) \textit{ and } \varrho \notin \mathit{frv}\,(\pi)\} \end{array}$$

It is immediate that $\mathcal{R}$ is well-formed, hence, by co-induction we have to show that $\mathcal{R} \subseteq [(\approx \mathcal{R} \approx) \cup \approx]$ or $\mathcal{R} \subseteq \lceil(\approx \mathcal{R} \approx) \cup \approx\rceil \cap \lceil(\approx \mathcal{R}^{-1} \approx) \cup \approx\rceil^{-1}$. We show that $\mathcal{R} \subseteq \lceil(\approx \mathcal{R} \approx) \cup \approx\rceil$, the other case is by similar reasoning. Take $\{\,\} \rhd e\,\mathcal{R}\,\mathtt{new}\,\varrho.\,e : \pi, \varphi$ and a transition $(S_r(e) : S_r(\pi), S_r(\varphi)) \xrightarrow{\ \gamma\ } (e_0 : \pi_0, \varphi_0)$ such that $\mathit{Clean}\,(S_r(\varphi))$. First, we can assume without loss of generalization that, by α-renaming, $\varrho \notin \mathit{frv}\,(\mathit{Img}\,(S_r)) \cup \mathit{frv}\,(\gamma)$. From Lemma 6.2, we have that $e \downharpoonleft$ and hence by Lemma 2.9, also that $S_r(e) \downharpoonleft$. By reduction rule (5), this implies that $S_r(\mathtt{new}\,\varrho.\,e) \downharpoonleft$ as well. Hence, from Lemma 6.1 we have the transition $(S_r(\mathtt{new}\,\varrho.\,e) : S_r(\pi), S_r(\varphi)) \xrightarrow{\ \gamma\ } (e_0' : \pi_0, \varphi_0)$.

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 $S_r(e) \overset{*}{\rightharpoonup} v$ and $S_r(\mathtt{new}\,\varrho.\,e) \overset{*}{\rightharpoonup} v[\varrho \mapsto \bullet]$ where $e_0 = v\gamma$ and $e_0' = (v[\varrho \mapsto \bullet])\gamma$.

First, observe that by either rule *(Obs-Lam)* or *(Obs-Fix)* we have that $\varrho \notin \mathit{frv}\,(\pi_0) \cup \mathit{frv}\,(\varphi_0)$. Also, since $\varrho \notin \mathit{frv}\,(\gamma)$, we have that $e_0' = (v\gamma)[\varrho \mapsto \bullet]$. Since $\mathit{Clean}\,(\varphi_0)$ and $\{\,\} \vdash_f v\gamma : \pi_0, \varphi_0$, we have that $v\gamma \downharpoonleft$ or $v\gamma \upharpoonleft$. Moreover, if $v\gamma \upharpoonleft$ then also $(v\gamma)[\varrho \mapsto \bullet] \upharpoonleft$ and the case is proven by Lemma 6.13. Similarly, if we have $v\gamma \downharpoonleft$, then $(v\gamma)[\varrho \mapsto \bullet] \downharpoonleft$, so assume $v\gamma \overset{*}{\rightharpoonup} v'$ and so by Lemma 2.9 also $(v\gamma)[\varrho \mapsto \bullet] \overset{*}{\rightharpoonup} v'[\varrho \mapsto \bullet]$. Using reduction rule (5), transitivity of $\approx$ and Lemma 6.14, we therefore have $\{\,\} \rhd e_0 \approx v' : \pi_0, \varphi_0$ and $\{\,\} \rhd \mathtt{new}\,\varrho.\,v' \approx e_0' : \pi_0, \varphi_0$. Therefore we have that $\{\,\} \rhd e_0 \approx \mathcal{R} \approx e_0' : \pi_0, \varphi_0$, proving the lemma.

$\square$

Now, we extend the result to open extensions:

Lemma 8.2 *Suppose* $\mathit{TE} \vdash_f e : \pi, \varphi'$ *where* $\varrho \notin \mathit{frv}\,(\pi, \mathit{TE})$ *and* $\varphi' \subseteq \varphi$ *with* $\mathit{Clean}\,(\varphi)$*, then* $\mathit{TE} \rhd e \approx^\circ \mathtt{new}\,\varrho.\,e : \pi, \varphi$.

Proof. To prove the result for open extensions, we take a ξ and a S_r such that $S_r(\mathit{TE}) \succ \xi$ and $\mathit{Clean}\,(S_r(\varphi))$. By definition of open extension, it is sufficient to prove that $\{\,\} \rhd \xi(S_r(e)) \approx \xi(S_r(\mathtt{new}\,\varrho.\,e)) : S_r(\pi), S_r(\varphi)$. By α-renaming, we can assure that $\varrho \notin \mathit{frv}\,(\xi) \cup \mathit{frv}\,(\mathit{Img}\,(S_r))$, hence the assertion is equivalent to $\{\,\} \rhd \xi(S_r(e)) \approx \mathtt{new}\,\varrho.\,\xi(S_r(e)) : S_r(\pi), S_r(\varphi)$.

We have $\mathit{Clean}\,(S_r(\varphi))$ and because $\varrho \notin \mathit{frv}\,(\pi)$, we also have $\varrho \notin \mathit{frv}\,(S_r(\pi))$. Moreover, since $\mathit{TE} \vdash_f e : \pi, \varphi'$ and by Lemma 3.1, we can apply Lemma 8.1 to prove the result. $\square$

Now, we can prove the main theorem:

Theorem 8.3 $\triangleq \subseteq \approx^\circ$.

Proof. We have to show that $\approx^\circ$ satisfies the rules in Figure 6. Then, because $\triangleq$ is the least relation satisfying these rules, we automatically have $\triangleq \subseteq \approx^\circ$. Observe that by Lemma 5.1, we already have that $\triangleq$ is well-formed.

(Eq-Comp). Immediate by Theorem 7.8.

(Eq-Symm) and *(Eq-Trans)*. Immediate from Proposition 6.10.3.

(Eq-Beta-n) with $n \in \{1, 2, 5, 6\}$. Suppose $TE \vdash_f e : \pi, \varphi'$ where $\varphi' \subseteq \varphi$ with $Clean(\varphi)$ and $e \overset{n}{\dashrightarrow} e'$. For arbitrary ξ and S_r where $S_r(TE) \succ \xi$ with $Clean(S_r(\varphi))$, we have by Lemma 3.1 that $\{\ \} \vdash_f \xi(S_r(e)) : S_r(\pi), S_r(\varphi')$. Also, by Lemma 2.8 and Lemma 2.9, we have that $\xi(S_r(e)) \overset{n}{\dashrightarrow} \xi(S_r(e'))$. So, by Lemma 6.14 (and via reduction rule (7)), it holds that $\{\ \} \rhd \xi(S_r(e)) \approx \xi(S_r(e')) : S_r(\pi), S_r(\varphi)$ and this for arbitrary ξ and S_r with $S_r(TE) \succ \xi$. 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 $Clean(\varphi)$, 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 $x \notin Dom(TE)$. By definition of completion, this implies that x does not occur free in ξ either, where we take ξ and S_r such that $S_r(TE) \succ \xi$ with $Clean(S_r(\varphi))$. We have to show $\{\ \} \rhd \langle \lambda x.\, \xi(S_r(e_1)) \rangle_{S_r(\varrho)} \,@\, \xi(S_r(e_2)) \approx \mathtt{let}\, x = \xi(S_r(e_2)) \,\mathtt{in}\, \xi(S_r(e_1)) : S_r(\pi), S_r(\varphi)$.

First, we can easily see that if $\langle \lambda x.\, \xi(S_r(e_1)) \rangle_{S_r(\varrho)} \,@\, \xi(S_r(e_2)) \uparrow$ holds, then it must be that $(\mathtt{let}\, x = \xi(S_r(e_2)) \,\mathtt{in}\, \xi(S_r(e_1))) \uparrow$ and the other way around. So, by Lemma 6.13, we have that the two terms are bisimilar.

Suppose now $\langle \lambda x. \xi(S_r(e_1)) \rangle_{S_r(\varrho)} \,@\, \xi(S_r(e_2)) \overset{*}{\rightharpoonup} \langle \lambda x. \xi(S_r(e_1)) \rangle_{S_r(\varrho)} \,@\, v_2 \overset{*}{\rightharpoonup} \xi(S_r(e_1))[x \mapsto v_2]$ where we assume $\xi(S_r(e_2)) \overset{*}{\rightharpoonup} v_2$. Then, we obtain $\{\ \} \rhd \xi(S_r(e_1)) \,@\, \xi(S_r(e_2)) \approx \xi(S_r(e_1))[x \mapsto v_2] : S_r(\pi), S_r(\varphi)$ by Lemma 6.14. Now, we also have the reductions $\mathtt{let}\, x = \xi(S_r(e_2)) \,\mathtt{in}\, \xi(S_r(e_1)) \overset{*}{\rightharpoonup} \mathtt{let}\, x = v_2 \,\mathtt{in}\, \xi(S_r(e_1)) \overset{*}{\rightharpoonup} \xi(S_r(e_1))[x \mapsto v_2]$, so again by Lemma 6.14 we obtain $\{\ \} \rhd \mathtt{let}\, x = \xi(S_r(e_2)) \,\mathtt{in}\, \xi(S_r(e_1)) \approx \xi(S_r(e_1))[x \mapsto v_2] : S_r(\pi), S_r(\varphi)$. 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* [16, 41]: Given $\{\ \} \vdash_f e_1 : \pi, \varphi_1$ and $\{\ \} \vdash_f e_2 : \pi, \varphi_2$ with $Clean(\varphi_1 \cup \varphi_2)$, then e_1 is Kleene-equivalent to e_2 iff there exists a v such that $e_1 \overset{*}{\rightharpoonup} v$ and $e_2 \overset{*}{\rightharpoonup} v$. 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). Immediate by Lemma 8.2.

(Eq-Dealloc). Since we have $Clean(\varphi)$, by Type Soundness (Proposition 2.6) either $e \uparrow$ (and since $\varrho \notin \varphi$ also $e[\varrho \mapsto \bullet] \uparrow$) or $e \downarrow$ (and therefore $e[\varrho \mapsto \bullet] \downarrow$).

Assume $e \Uparrow$ first. Then bisimilarity follows immediately by Lemma 6.13. So, suppose we have $e \Downarrow$ and $e[\varrho \mapsto \bullet] \Downarrow$, so we can assume that $e \xrightarrow{*} v$ and $e[\varrho \mapsto \bullet] \xrightarrow{*} v[\varrho \mapsto \bullet]$. Therefore, we obtain by Lemma 6.14 that $\{\ \} \rhd e \approx v : \pi, \varphi$ and $\{\ \} \rhd v[\varrho \mapsto \bullet] \approx e[\varrho \mapsto \bullet] : \pi, \varphi$.

Also, by reduction (3) and the same Lemma we have $\{\ \} \rhd \texttt{new}\, \varrho.\, v \approx v[\varrho \mapsto \bullet] : \pi, \varphi$ too. By transitivity of $\approx$ and Lemma 8.2, we prove the case.

□

9 Bisimilarity equals Contextual Equivalence

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

Proposition 9.1 $\approx^\circ$ *is well-formed, compatible and adequate.*

Proof. From Proposition 6.10.1 we have that $\approx^\circ$ is well-formed and by Theorem 7.8, we also know it is compatible. Adequacy follows from Lemma 6.12. □

We can now prove a Context Lemma, but instead of taking context rules [34], we only consider a sequence of safe (and well-typed) observations:

Lemma 9.2 $\{\ \} \rhd e_1 \lesssim e_2 : \pi, \varphi$ *iff* $\{\ \} \vdash_f e_1 : \pi, \varphi_1$ *and* $\{\ \} \vdash_f e_2 : \pi, \varphi_2$ *where* $\varphi_1 \cup \varphi_2 \subseteq \varphi$ *and for all* S_r *with* $Clean\,(S_r(\varphi))$ *and for all observations* $\gamma_1, \ldots, \gamma_n$ *such that for all* $i \in \{1, \ldots, n\}$ *where* $Safe\,(\gamma_i)$ *and* $\{\ \} \vdash_f S_r(e_1)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_1^i$ *and* $\{\ \} \vdash_f S_r(e_2)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_2^i$, *it holds that* $S_r(e_1)\gamma_1 \ldots \gamma_n \Downarrow \Rightarrow S_r(e_2)\gamma_1 \ldots \gamma_n \Downarrow$.

Proof.

Left-to-right Take a sequence of observations $\gamma_1, \ldots, \gamma_n$ such that for all $i \in \{1, \ldots, n\}$ we have that $Safe\,(\gamma_i)$ where $\{\ \} \vdash_f S_r(e_1)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_1^i$ and $\{\ \} \vdash_f S_r(e_2)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_2^i$. Since we have that $S_r(e_1)\gamma_1 \ldots \gamma_n \Downarrow$, by reduction rules (8) and (11) the following reductions hold: $S_r(e_1) \xrightarrow{*} v_1^1$; $v_1^1\gamma_1 \xrightarrow{*} v_1^2$; $v_1^2\gamma_2 \xrightarrow{*} v_1^3$; ... and $v_1^n\gamma_n \Downarrow$. By Lemma 6.1 and Proposition 2.3, this implies that $(S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma_1} (v_1^1\gamma_1 : \pi^1, \varphi^1)$ 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 $(S_r(e_2) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma_1} (v_2^1\gamma_1 : \pi^1, \varphi^1)$ and for $i \in \{1, \ldots, n-1\}$ the transitions $(v_2^i\gamma_i : \pi^i, \varphi^i) \xrightarrow{\gamma_{i+1}} (v_2^{i+1}\gamma_{i+1} : \pi^{i+1}, \varphi^{i+1})$ where $\{\ \} \rhd v_1^i\gamma_i \lesssim v_2^i\gamma_i : \pi^i, \varphi^i$ 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 6.1, that $v_2^n\gamma_n \Downarrow$ too. Applying Lemma 6.2 subsequently on the transitions from $v_2^i\gamma_i$, makes that $S_r(e_2) \xrightarrow{*} v_2^1$; $v_2^1\gamma_1 \xrightarrow{*} v_2^2$; $v_2^2\gamma_2 \xrightarrow{*} v_2^3$; ... too. Therefore we have $S_r(e_2)\gamma_1 \ldots \gamma_n \Downarrow$, which proves the case.

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

$$\begin{aligned}\mathcal{R} = \; & \{(\{\,\}, e_1, e_2, \pi, \varphi) \mid \{\,\} \vdash_f e_1 : \pi, \varphi_1 \textit{ and } \{\,\} \vdash_f e_2 : \pi, \varphi_2 \textit{ where } \varphi_1 \cup \varphi_2 \subseteq \varphi \\ & \textit{and for all } S_r \textit{ with } \textit{Clean}\,(S_r(\varphi)) \textit{ and for all } \gamma_1, \ldots, \gamma_n \textit{ such that for all} \\ & i \in \{1, \ldots, n\} \textit{ we have } \textit{Safe}\,(\gamma_i) \textit{ and } \{\,\} \vdash_f S_r(e_1)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_1^i \textit{ and} \\ & \{\,\} \vdash_f S_r(e_2)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_2^i \textit{ it holds that} \\ & S_r(e_1)\gamma_1 \ldots \gamma_n \downharpoonleft \Rightarrow S_r(e_2)\gamma_1 \ldots \gamma_n \downharpoonleft\}\end{aligned}$$

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 transition $(S_r(e_1) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_1' : \pi', \varphi')$ such that $\textit{Clean}\,(S_r(\varphi))$. This implies that $S_r(e_1) \downharpoonleft$ and since $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$ we therefore have $S_r(e_2) \downharpoonleft$ (where we take the empty sequence of observations). By Lemma 6.5, $(S_r(e_2) : S_r(\pi), S_r(\varphi)) \xrightarrow{\gamma} (e_2' : \pi', \varphi')$ exists.

Whenever $\textit{Deadly}\,(S_r(\gamma))$, then the claim is proven by reflexivity of $\lesssim$. Therefore, assume $\textit{Safe}\,(S_r(\gamma))$. So, we also have that $S_r(e_1) \stackrel{*}{\rightharpoonup} v_1$ with $e_1' = v_1\gamma$ and $S_r(e_2) \stackrel{*}{\rightharpoonup} v_2$ with $e_2' = v_2\gamma$. By Lemma 6.3, we also have that $\{\,\} \vdash_f e_1' : \pi', \varphi'$ and $\{\,\} \vdash_f e_2' : \pi', \varphi'$.

Now, take an arbitrary sequence of observations $\gamma_1, \ldots, \gamma_n$ such that for all $i \in \{1, \ldots, n\}$ it holds that $\textit{Safe}\,(\gamma_i)$ and $\{\,\} \vdash_f S_r(e_1)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_1^i$ and $\{\,\} \vdash_f S_r(e_2)\gamma_1 \ldots \gamma_i : \pi^i, \varphi_2^i$. Now assume that $v_1 S_r(\gamma)\gamma_1 \ldots \gamma_n \downharpoonleft$. Since $S_r(e_1) \stackrel{*}{\rightharpoonup} v_1$, we have because of the reductions (8) and (11) that $S_r(e_1\gamma)\gamma_1 \ldots \gamma_n \downharpoonleft$. Because $\{\,\} \rhd e_1 \,\mathcal{R}\, e_2 : \pi, \varphi$, this implies $S_r(e_2\gamma)\gamma_1 \ldots \gamma_n \downharpoonleft$. But, we also have $S_r(e_2) \stackrel{*}{\rightharpoonup} v_2$ and again using the reductions (8) and (11) that $v_2 S_r(\gamma)\gamma_1 \ldots \gamma_n \downharpoonleft$. It now follows that $\{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi'$ (the typings are trivially checked). By definition of $\mathcal{R}$, we also have $\{\,\} \rhd e_1' \,\mathcal{R}\, e_2' : \pi', \varphi' \cup \varphi''$ with $\textit{Clean}\,(\varphi'')$, proving the claim.

□

Proposition 9.3 *If* $\{\,\} \rhd e_1 \simeq e_2 : \pi, \varphi$*, then* $\{\,\} \rhd e_1 \approx e_2 : \pi, \varphi$.

Proof. We have $\{\,\} \rhd e_1 \simeq e_2 : \pi, \varphi$ and hence, by definition 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 $\textit{Clean}\,(\varphi)$. Then, take S_r such that $\textit{Clean}\,(S_r(\varphi))$ and a sequence of observations $\gamma_1, \ldots, \gamma_n$ such that for all $i \in \{1, \ldots, n\}$ it holds that $\textit{Safe}\,(\gamma_i)$ and $\{\,\} \vdash_f S_r(e_1)\gamma_1 \ldots \gamma_i : \pi^i, \varphi^i$ and $\{\,\} \vdash_f S_r(e_2)\gamma_1 \ldots \gamma_i : \pi^i, \varphi^i$. If we can prove that $S_r(e_1)\gamma_1 \ldots \gamma_n \downharpoonleft \Leftrightarrow S_r(e_2)\gamma_1 \ldots \gamma_n \downharpoonleft$, the Proposition follows directly from Context Lemma 9.2.

To show the latter, let us assume that $\textit{Supp}\,(S_r) = \{\varrho_1, \ldots, \varrho_n\}$ and $\textit{Img}\,(S_r) = \{\rho_1', \ldots, \rho_n'\}$. By compatibility of $\simeq$ and the context rules *(Comp-Lamv)*, *(Comp-Fixv)*, and *(Comp-Regapp)* we have that $\{\,\} \rhd e_1' \simeq e_2' : S_r(\pi), \varphi \cup \{\varrho\}$, where $\varrho \notin \textit{Supp}\,(S_r)$, the expression e_1' is $(\langle \texttt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\ e_1\rangle_\varrho\ [\rho_1', \ldots, \rho_n']\ \texttt{at}\ \varrho) @ \langle \lambda\, x.\ x\rangle_\varrho$ and the expression e_2' similarly $(\langle \texttt{fix}\, f.\Lambda\, \varrho_1, \ldots, \varrho_n.\lambda\, x.\ e_2\rangle_\varrho\ [\rho_1', \ldots, \rho_n']\ \texttt{at}\ \varrho) @ \langle \lambda\, x.\ x\rangle_\varrho$, assuming that $\{x, f\} \cap (\textit{fv}\,(e_1) \cup \textit{fv}\,(e_2)) = \emptyset$.

Additionally, by context rules *(Comp-App)* and *(Comp-Regapp)*, we can derive that $\{\,\} \rhd e_1'\gamma_1 \ldots \gamma_n \ \{(\{\,\}, e_1', e_2', S_r(\pi), \varphi \cup \{\varrho\})\}^c \ e_2'\gamma_1 \ldots \gamma_n : \pi^n, \varphi^n$. Since contextual equivalence makes $\{(\{\,\}, e_1', e_2', S_r(\pi), \varphi \cup \{\varrho\})\}^c$ adequate, we also have $e_1'\gamma_1 \ldots \gamma_n \downharpoonleft \Leftrightarrow e_2'\gamma_1 \ldots \gamma_n \downharpoonleft$. From the definition of e_1' and e_2', it is immediate that $S_r(e_1)\gamma_1 \ldots \gamma_n \downharpoonleft \Leftrightarrow S_r(e_2)\gamma_1 \ldots \gamma_n \downharpoonleft$ also holds.
□

We conclude by formulating the main equivalence theorem: the notion of applicative bisimilarity for the $\lambda_f^{\textit{region}}$-calculus is an alternative characterization of contextual equivalence.

Theorem 9.4 (Bisimilarity equals Contextual Equivalence) $\approx^\circ = \simeq$

Proof. As an immediate consequence of Proposition 4.7 and Proposition 9.1, we have that $\approx^\circ \subseteq \simeq$. Remains to show that $\simeq \supseteq \approx^\circ$. Suppose $TE \rhd e_1 \simeq e_2 : \pi, \varphi$, an iterated substitution ξ and a region substitution S_r for which $TE \succ \xi$ such that $Clean\,(S_r(\varphi))$. To prove that $TE \rhd e_1 \approx^\circ e_2 : \pi, \varphi$, we have to show that $\{\ \} \rhd \xi(S_r(e_1)) \approx \xi(S_r(e_2)) : S_r(\pi), S_r(\varphi)$.

By Proposition 4.7 and Lemma 4.2, we have $\{\ \} \rhd \mathcal{T}(\xi, S_r(e_1)) \simeq \mathcal{T}(\xi, S_r(e_2)) : S_r(\pi), S_r(\varphi)$. Also, by Theorem 8.3 and Lemma 5.2 we have $\{\ \} \rhd \mathcal{T}(\xi, S_r(e_1)) \simeq \xi(S_r(e_2)) : S_r(\pi), S_r(\varphi)$ and $\{\ \} \rhd \xi(S_r(e_1)) \simeq \mathcal{T}(\xi, S_r(e_2)) : S_r(\pi), S_r(\varphi)$. So, by transitivity of $\simeq$, this implies that $\{\ \} \rhd \xi(S_r(e_1)) \simeq \xi(S_r(e_2)) : S_r(\pi), S_r(\varphi)$ and hence, by Proposition 9.3, it follows that $\{\ \} \rhd \xi(S_r(e_1)) \approx \xi(S_r(e_2)) : S_r(\pi), S_r(\varphi)$. $\Box$

10 Conclusions and Future Work

We have defined a typed equational theory for a region calculus in the style of Tofte and Talpin [53]. 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 [16, 17]. 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.

Although the equational theory provides a framework to reason about region-annotated programs, it is not immediately usable to show the translation correctness of region inference [53]. In their seminal paper on region inference, Tofte and Talpin combine translation soundness and type soundness, leading to a monolithic, complex and un-intuitive proof. However, we conjecture that is possible to use the bisimilarity notion of this paper to prove translation correctness. We already have by compatibility of bisimilarity and Lemma 8.2 that region allocation *does not matter*. If we are somehow able to formulate that region abstraction via `fix` does not matter either, we obtain a result which states that a region inferred program is equivalent to the same program without region allocations or abstractions and with one global region in the annotations. From this, it is trivial to show that such a program is equivalent to the program without region annotations.

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 [42, 47].

Acknowledgments

The author wishes to thank Peter Thiemann and Fritz Henglein for commenting an initial draft. Silvano Dal Zilio, Andy Gordon and Matthias Neubauer provided useful comments and suggestions.

References

[1] Samson Abramsky. The lazy lambda calculus. In David 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 *Proceedings of the 1995 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 Cooper, editor, *Proceedings of the 1998 Conference on Programming Language Design and Implementation*, pages 313–324, Montreal, Canada, June 1998. ACM. Volume 33(5) of SIGPLAN Notices.

[5] Anindya Banerjee, Nevin Heintze, and Jon Riecke. Region analysis and the polymorphic lambda calculus. In *Proceedings of the 1999 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 [46], 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 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 Alexander Aiken, editor, *Proceedings of the 1999 ACM SIGPLAN 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 [43], pages 59–69.

[13] David Gay and Alexander Aiken. Language support for regions. In PLDI2001 [43], pages 70–80.

[14] David Gifford and John Lucassen. Integrating functional and imperative programming. In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 28–38, 1986.

[15] Andrew Gordon. *Functional Programming and Input/Output.* Cambridge University Press, 1994.

[16] Andrew Gordon. Operational equivalences for untyped and polymorphic object calculi. In Gordon and Pitts [18], pages 9–54.

[17] Andrew Gordon. Bisimilarity as a theory of functional programming. *Theoretical Computer Science*, 228(1-2):5–47, October 1999.

[18] Andrew Gordon and Andrew Pitts, editors. *Higher Order Operational Techniques in Semantics (HOOTS)*, Publications of the Newton Institute. Cambridge University Press, 1998.

[19] Andrew Gordon and Gareth Rees. Bisimilarity for a first-order calculus of objects with subtyping. In POPL1996 [46], 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*, 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 (HOOTS)*, 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. (submitted for publication).

[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 Howe. Equality in lazy computation systems. In LICS1989 [29], pages 198–203.

[25] Douglas 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 Gifford. Algebraic reconstruction of types and effects. In *Proc. 1991 ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 303–310, Orlando, Florida, January 1991. ACM Press.

[28] Søren Lassen. Relational reasoning about contexts. In Gordon and Pitts [18], pages 91–136.

[29] *Proceedings of the 1989 IEEE Symposium on Logic in Computer Science*, Pacific Grove, CA, June 1989. IEEE Computer Society Press.

[30] Henning Makholm. Region-based memory management in Prolog. Master's thesis, Department of Computer Science, University of Copenhagen, Denmark, March 2000.

[31] Ian Mason and Carolyn Talcott. Programming, transforming, and proving with function abstractions and memories. In *Proceedings of the 1989 International EATCS Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 574–588. Springer-Verlag, 1989.

[32] Ian Mason and Carolyn Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):287–328, July 1991.

[33] Ian Mason and Carolyn Talcott. Inferring the equivalence of functional programs that mutate data. In *Proceedings of the 1992 IEEE Symposium on Logic in Computer Science*, pages 186–197. IEEE Computer Society Press, June 1992.

[34] Robin Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1978.

[35] Robin Milner. *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[36] Robin Milner, Mads Tofte, Robert Harper, and Dave MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[37] Eugenio Moggi. Computational lambda-calculus and monads. In LICS1989 [29], pages 14–23.

[38] James Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT Press, December 1968.

[39] Benjamin Pierce and Davide Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In Neil Jones, editor, *Proceedings of the 1997 ACM SIGPLAN Symposium on Principles of Programming Languages*, Paris, France, January 1997. ACM Press.

[40] Andrew Pitts. Reasoning about local variables with operationally-based logical relations. In *Proceedings of the 1996 IEEE Symposium on Logic in Computer Science*, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[41] 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.

[42] Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In Gordon and Pitts [18], pages 309–326.

[43] *Proceedings of the 2001 Conference on Programming Language Design and Implementation*, Snowbird, UT, June 2001. ACM Press.

[44] Gordon Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[45] Gordon Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Denmark, 1981.

[46] *Proceedings of the 1996 ACM SIGPLAN Symposium on Principles of Programming Languages*, St. Petersburg, Fla., January 1996. ACM Press.

[47] Caroline Talcott. Reasoning about functions with effects. In Gordon and Pitts [18], pages 346–390.

[48] Jean-Pierre Talpin and Pierre Jouvelot. Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3):245–272, July 1992.

[49] Mads Tofte and Lars Birkedal. A region inference algorithm. *ACM Transactions on Programming Languages and Systems*, 20(5):724–767, 1998.

[50] 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.

[51] 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`.

[52] Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value λ-calculus using a stack of regions. In *Proceedings of the 1994 ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 188–201, Portland, OR, January 1994. ACM Press.

[53] Mads Tofte and Jean-Pierre Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.

[54] 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.

[55] 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.

[56] David Walker and Kevin Watkins. On regions and linear types. In Xavier Leroy, editor, *Proceedings of the 2001 International Conference on Functional Programming*, pages 181–192, Florence, Italy, September 2001. ACM Press, New York.

[57] Paul Wilson. Uniprocessor garbage collection techniques. Technical report, University of Texas, 1999.

[58] Silvano Dal Zilio and Andrew Gordon. Region analysis and a pi-calculus with groups. Technical Report MSR-TR-2000-57, Microsoft Research, August 2000.

[59] Silvano Dal Zilio and Andrew Gordon. Region analysis and a pi-calculus with groups. *Journal of Functional Programming*, 12(3):229–292, May 2002.