A Constraint-Based Region Inference Algorithm

Lars Birkedal [1]

School of Computer Science, Carnegie Mellon University

Mads Tofte [2]

Dept. of Computer Science, University of Copenhagen

Abstract

Region inference is a type-based technique for determining runtime memory management at compile time. It is targeted at a runtime model in which the store consists of a stack of regions and memory management predominantly consists of pushing and popping regions. Region inference has previously been specified by a set of inference rules, which formalize when regions may be allocated and deallocated. We present a constraint-based algorithm which implements the specification. It consists of a constraint-based formulation of region inference and a rewriting algorithm for solving a set of constraints. We prove that the constraint-based formulation of region inference is sound with respect to the region inference rules. Moreover we prove "restricted completeness" of the constraint-based formulation, that is, we define an "implementation-oriented" formulation of region inference and show that every derivation in the implementation-oriented formulation can be obtained by a solution to a corresponding constraint problem. We show that rewriting of a constraint problem always terminates with a solution to the original system.

Keywords: Dynamic Storage Management, Regions, Standard ML.

Contents

[1] Supported in part by U.S. National Science Foundation Grant CCR-9409997. Address: Lars Birkedal, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA. E-mail: `birkedal@cs.cmu.edu`

[2] Address: Mads Tofte, DIKU, Universitetsparken 1, DK-2100 Copenhagen, Denmark. E-mail:`tofte@diku.dk`

1 Introduction

We first give a brief introduction to the concept of Region Inference, for readers who are not familiar with previous work on the topic. We then describe the contribution of this paper.

1.1 Background

Region Inference is a type-based technique for determining runtime memory management at compile time.

Memory management is an essential aspect of computer programming. Simply put, the problem is that computers have finite memory and that some recycling of memory is necessary. A program that consistently allocates more memory than it releases will eventually use up all the memory of the computer. A program that de-allocates memory too early (i.e., when the memory contains values that are actually required by the remainder of the computation) may crash or give wrong results.

Most programming languages provide the programmer with a memory management discipline, which helps the user manage memory, in return for imposing restrictions on how memory may be used.

Indeed, memory management considerations have often played a central role in the design of programming languages. One famous example is the programming language Algol 60 [10], which introduced what we will call the *pure* stack discipline. In the pure stack discipline the runtime stack can hold all values produced by the computation, including temporary variables, non-local variables, return addresses, and even certain variable-sized arrays. (The term *pure* refers to the idea that *all* values can be stored on the stack.) The beauty of the pure stack discipline is that every point of allocation is paired with a point of de-allocation and these points are easily identified from the program text. This gives predictability in programming and, when properly used, very space-economical memory use.

The pure stack discipline has severe limitations, however. One limitation concerns physical sizes of values. The physical size of a value must be known at the latest when memory allocation for the value takes place at runtime.

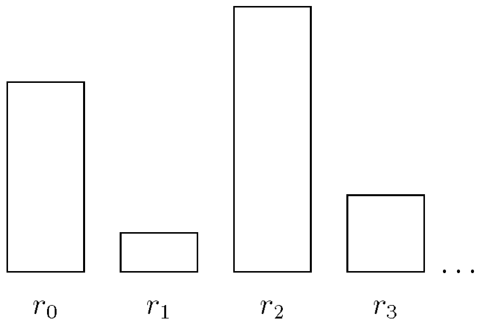

Fig. 1. The store is a stack of regions; a region is a box in the picture.

This rules out lists and other recursive datatypes, which are normally constructed incrementally. Also, the use of the stack for holding the environment rules out useful programming techniques, especially returning functions from function calls (under call-by-value). In particular, if we regard an object in object-oriented programming as containing functions (methods), then methods cannot in general return objects as results under the pure stack discipline.

Many programming languages get around these limitations by using a less pure stack discipline, where the values that cannot be put on the stack are stored in a *heap*. Recycling of memory is then done either explicitly by the programmer or automatically, by the *garbage collector*, which is a separate routine in the runtime system. Explicit management of memory is notoriously difficult and time consuming for the programmer. Garbage collection has been developed over a period of some fourty years (see Wilson [18] for an excellent overview). Thanks to this work, garbage collection has matured to be a practical form of memory management. With the success of Java, garbage collection is even achieving wide-spread use and acceptance.

However, garbage collection does not offer the predictability of the pure stack discipline. The separation of allocation and de-allocation makes it difficult to know how much memory a program uses. Typically, programmers resort to simply running the program on different input values to see what happens. It is difficult for a programmer who has no way of asserting or checking lifetimes of values to avoid that the garbage collector hangs on to values which are actually not needed.

Region-based memory management [16,2,17] offers an alternative to the above memory management disciplines. At runtime, all values are stored in a so-called *stack of regions*, see Figure 1. All values, including function closures and values of recursive types (such as lists and trees) are stored in regions. Every region can grow dynamically and there is no fixed bound on the number of regions that can exist at runtime.

The size of a region can sometimes, but not always, be inferred at compile time. Thus a region stack as shown in Figure 1 cannot directly be represented in one-dimensional memory. However, static analyzes and implementation techniques for overcoming this difficulty have been developed [2,15]. As a result, all the central region operations (allocation of region, de-allocation of region, allocation of value in region) are constant-time operations. Thus the overhead of memory management is spread over the execution, without interruptions of unbounded duration. This property is attractive for real-time programming.

The allocation and de-allocation of regions is determined at compile time, by a type-based analysis called *region inference*.

Region inference annotates every value-creating sub-expression of the source program with an annotation of the form

$$\texttt{at}\ \rho$$

where ρ is a *region variable*. Note the distinction between *region* and *region variable*. A region is an area of memory at runtime. A region variable is a syntactic object in the program, so a program contains only finitely many region variables. At runtime, an environment maps region variables to regions.

Region inference also introduces bindings of region variables. One form of binding is the expression

$$\texttt{letregion}\ \rho\ \texttt{in}\ e$$

which binds ρ in the expression e. At runtime, first a region is allocated at the top of the region stack, then e is evaluated (presumably storing and fetching values from the region) and finally, when e terminates, the region is popped off the stack. The expression e may itself contain further `letregion` expressions, or call functions that contain `letregion` expressions. However, the `letregion` construct is the *only* primitive for allocating and de-allocating regions, from which it follows that regions obey a stack discipline.

The second form of binding of region variables takes the form

$$\texttt{letrec}\ f\texttt{[}\rho_1\texttt{,}\ldots\texttt{,}\rho_k\texttt{](}x\texttt{)}\ \texttt{at}\ \rho\ \texttt{=}\ e_1\ \texttt{in}\ e_2 \qquad (1)$$

Here f is a (possibly recursive) function with formal parameter x, body e_1 and scope e_1 and e_2. The region variables $\rho_1, \ldots, \rho_k$ are called the *formal region parameters* of f. Intuitively, they indicate where f should place the values it produces — so e_1 may contain annotations of the form $\texttt{at}\ \rho_i$, $i \in \{1, \ldots, k\}$, among others. The scope of the binding $\rho_1, \ldots, \rho_k$ is e_1. The region variable ρ indicates where the closure for f is to be placed.

A function declared with the above `letrec` construct is said to be *region-*

polymorphic. Every non-binding occurrence of f takes the form

$$f\texttt{[}\rho'_1\texttt{,}\ldots\texttt{,}\rho'_k\texttt{]}\ \texttt{at}\ \rho'$$

where $\rho'_1, \ldots, \rho'_k$ are *actual region parameters*. The notation $\texttt{[}\rho'_1\texttt{,}\ldots\texttt{,}\rho'_k\texttt{]}\ \texttt{at}$ ρ' indicates that a record for holding the region parameters is created and stored in the region denoted by ρ'.

Different non-binding occurrences of f may use different actual region parameters. Indeed, region inference allows *polymorphic recursion* in regions, that is, referring to (1), inside e_1, f may be applied to actual region parameters which are different from the formal region parameters $\rho_1, \ldots, \rho_k$.

Region-based memory management has been implemented in *The ML Kit with Regions* [15], a compiler for Standard ML which uses regions as its sole form of memory management.

1.2 Region Inference Rules

Tofte and Talpin [16] introduced region inference not as an algorithm, but as a set of inference rules in the style of type inference rules.

Region inference is based on earlier work on effects and (a different notion of) regions [13,3,6]. In particular, we use *effects* for determining the lifetime of values. As a first approximation, let us take an *effect* of an expression to be a finite set of region variables. The set is an upper bound on the set of regions that the evaluation of the expressions uses for storing and retrieving values. We use φ to range over effects.

The judgements of the region inference rules take the form

$$TE \vdash e \Rightarrow e' : (\tau, \rho), \varphi$$

where TE is a *type environment*, e is a *source expression*, e' is a *target expression*, τ is a *(region-annotated) type*, ρ is a region variable, and φ is an effect. These semantic objects are defined in detail below; the present description merely gives an overview. The type environment maps every free variable x of e to pairs of the form (σ, ρ') where σ is a *region type scheme* and ρ' is a region variable which indicates where the value of x resides. The region-annotated type is similar to a type in the Hindley-Milner type system, except that every type constructor is annotated by a region variable. The region variable ρ indicates where the value of e' will be when e' has been evaluated. The effect φ is an upper bound on the set of regions used during the evaluation of e'.

The source expression e and the target expression e' are identical, except that e' contains region information as described in Section 1.1, whereas e contains no region information. The algorithmic problem of region inference is the following: given TE and e as input, find e', (τ, ρ) and φ such that $TE \vdash e \Rightarrow e' : (\tau, \rho), \varphi$ can be inferred from the region inference rules.

It turns out that if e is closed and well-typed according to Milner's type discipline [8], then there is a trivial solution to the algorithmic region inference problem: choose the same same region ρ at every value-producing expression, never use `letregion` and never use region polymorphism. Unfortunately, the simple solution is not interesting, for it gives the worst possible runtime behaviour, similar to a single allocation space which is never garbage collected.

To be useful, a region inference algorithm should keep regions as separate as possible (in order to increase the possibility of separating lifetimes), it should push `letregion` constructs as deeply into expressions as possible (to shorten lifetimes) and it should use region polymorphism as much as possible (to distinguish and shorten lifetimes). Ideally, there would be an "optimal" solution, just as there are principal type schemes in Milner's type discipline [4]. However, region inference is more complex than ML type inference, due to the presence of effects and of polymorphic recursion in regions and effects. It is not known whether there is some decidable notion of optimal solution for the region inference problem.

Tofte and Birkedal [14] present a region inference algorithm. Like Milner's algorithm W [8,4], the algorithm is *syntax directed*, that is, it proceeds by induction on the structure of programs. Polymorphic recursion in regions is handled by iteration over the declaration of the recursive function, in the style of Mycroft [9]. Full polymorphic recursion in types leads to an undecidable type system [7,5]. In terms of Mycroft's algorithm, there must exist ill-typed terms for which the iterative algorithm fails to terminate. Similar termination problems exist for inferring regions and effects: if one is not careful, effects (which are sets) can grow unboundedly during iteration. However, Tofte and Birkedal [14] prove that their algorithm always terminates, at the cost of limiting the derivations that the algorithm constructs.

1.3 Constraint-Based Region Inference

The basic contribution of this paper is to present a new region inference algorithm which uses constraints instead of syntax-directed iteration.

There are several reasons why using constraints for region inference, if possible, is desirable. First, one can hope to break the problem of region inference into two separate problems: *generation* of constraints and *solution* of constraints. Second, constraint systems have been studied and classified as mathematical systems in their own right, and it would be interesting to know where region inference fits in, if at all. Third, one can hope that generic, algorithmic techniques developed for constraints, including fast data structures and algorithms, can be used for region inference. Finally, one could hope that formulating the problem as a constraint problem might shed light on the question of the existence of principal types for region inference.

In this paper we will show that region inference can indeed be formulated as a constraint solution problem. We prove that the constraint-based formulation is sound, in the sense that every solution to the constraint-based formulation of the region inference problem represents a derivation using the original region inference rules. We prove "restricted completeness" of the constraint-based formulation, in the following sense. We define an "implementation-oriented" version of the Tofte-Talpin region inference rules and show that every derivation in the implementation-oriented region inference rules can be obtained as a solution to a corresponding constraint problem.

Furthermore, we show that the constraint problems can be solved using mostly conventional constraint solving techniques. We show that rewriting of constraint problems always terminates and terminates with a solution to the original constraint problem. Unfortunately, the rewriting system is not confluent, so the constraint formulation does not settle the issue of whether there exists a notion of principal solutions for region inference. However, the constraint formulation makes it possible to pinpoint where the normalization procedure looses completeness.

1.4 Reader's Guide

We first introduce useful notation and review the Tofte-Talpin region inference rules and the Tofte-Birkedal region inference algorithm (Section 2). Common to this work is that it operates with so-called *semantic objects* in the type system, for example finite sets and maps.

In more detail, Section 2, consists of four subsections

- Notation
- Source Expressions
- Target Expressions
- Implementation-Oriented Formulation of Region Inference

In the subsection on Source Expressions (Section 2.2) we recall the definition of the source language of region inference. It is defined by means of ML typing rules, so we recall the necessary definitions of ML types and ML type schemes.

In the subsection on Target Expressions (Section 2.3) we recall the definition of the target language of region inference. It is defined by means of the region inference rules of Tofte and Talpin [17]. We refer to those inference rules as the semantic region inference rules. Before presenting the semantic region inference rules, we recall the definition of the semantic objects used in the definition of the region inference rules.

In the subsection on an Implementation-Oriented Formulation of Region Inference (Section 2.4) we impose some restrictions on the semantic region inference rules. The restrictions are expressed by an inference system that can be seen as an implementation-oriented formulation of region inference. The inference system is based on one that was developed as a foundation for the syntax-directed algorithm [14], but the present system is simpler and permits more derivations. The restrictions are formulated using the concept of consistency of semantic objects. Imposing consistency basically allows one to implement certain semantic objects more efficiently. Also, we define notions of normal form and well-formedness for semantic type schemes. We present the definitions of consistency, normal form, and well-formedness before presenting the actual rules for the implementation-oriented formulation of region inference.

The presentation of the constraint-based formulation starts in Section 3. The constraint system is purely syntactic in nature, so we refer to the objects in the constraint formulation as the *syntactic* objects. In more detail, Section 3 consists of three subsections

- Representation of Semantic Objects by Syntactic Objects
- Constraint Systems
- Constraint-based Formulation of Region Inference

In the subsection on Representation of Semantic Objects by Syntactic Objects (Section 3.1) we introduce the syntactic objects our constraint-based algorithm uses. The idea is that the syntactic objects are used to represent the semantic objects of region inference. We explain how the representation works.

In the subsection on Constraint Systems (Section 3.2) we then define what a constraint among syntactic objects is. We explain the intuition of the constraints by referring to the semantic objects which the syntactic objects represent. Since not all constraint sets correspond to real programs we define a notion of constraint system which essentially is a suitably well-formed collection of constraints. We further define what it means for a substitution to solve a constraint system; this definition of course makes use of the way in which syntactic objects represent semantic objects.

In the subsection on a Constraint-based Formulation of Region Inference (Section 3.3) we finally show how our constraints can be used to give a syntactic formulation of region inference. Using a substitution mapping syntactic objects to the semantic objects they represent, we show that the constraint-based formulation of region inference is sound with respect to the semantic region inference rules. We furthermore show that the constraint-based formulation of region inference is complete with respect to the implemention-oriented formulation of region inference.

In Section 4 we show how to find a solution to a constraint system by constraint normalization. There are of two subsections:

- Normal Form
- Constraint Normalization

In the Normal Form section (Section 4.1) we define what it means for a constraint system to be in normal form. The point of the normal form definition is that it is trivial to find a solution to a constraint system in normal form. In the section on Constraint Normalization (Section 4.2) we then define a set of rewriting rules with which a constraint system can be rewritten into a normal form constraint system. We show that the rewriting rules are sound and that rewriting always terminates.

Our constraint-based region inference algorithm works by first generating a set of constraints, as expressed by the constraint-based formulation of region inference, and then rewriting this set of constraints until a normal form is reached and a solution is found to the set of constraints. Using that solution, a target expression is found in the way expressed by the constraint-based formulation of region inference.

In Section 5 we give a few remarks on the implementation of the constraint-based region inference algorithm in the ML Kit and in Section 6 we finally conclude. At the end of the paper there is an index of notation and definitions.

We assume familiarity with the lambda calculus and ML-style polymorphic type inference.

2 Region Inference Using Semantic Objects

Section 2.1 presents notation that is used throughout this paper. Sections 2.2 and 2.3 describe the source and target languages of region inference, based on [16]. Section 2.4 describes an implementation-oriented version of the Tofte-Talpin region inference rules, based on [14].

2.1 Notation

A *finite map* is a map with finite domain. The domain and range of a finite map f are denoted $\mathrm{Dom}(f)$ and $\mathrm{Rng}(f)$, respectively. When f and g are finite maps, $f + g$ is the finite map whose domain is $\mathrm{Dom}(f) \cup \mathrm{Dom}(g)$ and whose value is $g(x)$, if $x \in \mathrm{Dom}(g)$, and $f(x)$ otherwise. For a finite map f and a set A, $f \downarrow A$ means the restriction of f to A, and $f \setminus\!\!\setminus A$ means f restricted to the complement of A. We write $\{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ for the finite map that maps x_i to y_i, for all $1 \leq i \leq n$.

For any set X, if x ranges over X, we let $\vec{x}$ range over the set of finite sequences over X. We are not concerned with infinite sequences so henceforth "sequence" always means "finite sequence." Formally, the empty sequence is denoted $\emptyset$, but when there is no risk of confusion, we simply omit writing it. The sequence consisting of the elements[3] $x_1, \ldots, x_n$, in that order, is written $x_1 \cdots x_n$, that is, by juxtaposing the elements $x_1, \ldots, x_n$. The concatenation of two sequences $\vec{x}$ and $\vec{x}'$ is written $\vec{x}\vec{x}'$. For a sequence $\vec{x} = x_1 \cdots x_n$, $|\vec{x}|$ denotes the set of elements $\{x_1, \ldots, x_n\}$. However, when there is no risk of confusion, we omit the vertical bars and simply write $\vec{x}$ for the set of elements in the sequence $\vec{x}$. For a sequence $\vec{x}$ and a set X, we write $\vec{x} \setminus X$ for the sequence $\vec{x}'$ satisfying that $|\vec{x}'| = |\vec{x}| \setminus X$ and for all $x_i, x_j \in |\vec{x}'|$, x_i comes before x_j in $\vec{x}'$ if and only if x_i comes before x_j in $\vec{x}$. For two sequences $\vec{x}$ and $\vec{x}'$, we write $\vec{x} \mathbin{\overline{\cup}} \vec{x}'$ for the sequence $\vec{x}\vec{x}''$ where $\vec{x}'' = \vec{x}' \setminus |\vec{x}|$. Note that, if all the elements in the sequence $\vec{x}$ are distinct and all the elements in the sequence $\vec{x}'$ are distinct, then all the elements in the sequence $\vec{x} \mathbin{\overline{\cup}} \vec{x}'$ are distinct. For two sequences $\vec{x} = x_1 \cdots x_n$ and $\vec{x}' = x'_1 \cdots x'_n$ of the same length $n \geq 0$, $\mathrm{pairs}(\vec{x}, \vec{x}')$ is the sequence $(x_1, x'_1) \cdots (x_n, x'_n)$. The length of a sequence $\vec{x}$ is denoted $\mathrm{length}(\vec{x})$. For a finite set X, we write $\mathrm{seq}(X)$ for the sequence consisting of the elements in the set X. (Strictly speaking, this definition only makes sense if one imposes some ordering on the elements of X. But we only use this operation when the actual ordering chosen is not important and when an ordering indeed can be defined on the elements of X.)

2.2 Source Expressions

The source language of region inference is the call-by-value lambda calculus with `letrec`. Expressions have to be well-typed according to Milner's type discipline [8,4]. More precisely, an expression can be annotated according to the region inference rules described below if and only if it is well-typed according to Milner's type discipline [17].

[3] The interval between two indices in $x_1, \ldots, x_n$ is always assumed to be 1 unless otherwise explicitly stated; thus $x_1, \ldots, x_n$ is short for $x_1, x_2, \ldots, x_n$.

Let ProgVar and TyVar be two denumerably infinite sets of *program variables*, and *type variables*, pairwise disjoint. We let x and f range over program variables and α range over type variables.

$$
\begin{aligned}
x, f &\in \mathrm{ProgVar} \\
\alpha &\in \mathrm{TyVar}
\end{aligned}
$$

ML types, $\overline{\tau}$, and *ML type schemes*, $\overline{\sigma}$, are defined as follows:

$$
\begin{aligned}
\overline{\tau} &::= \alpha \ \mid \ \overline{\tau} \to \overline{\tau} \\
\overline{\sigma} &::= \forall \vec{\alpha}.\overline{\tau} \qquad \text{where } \vec{\alpha} \subseteq \mathrm{ftv}(\overline{\tau}),
\end{aligned}
$$

where $\mathrm{ftv}(\overline{\tau})$ is the set of type variables in $\overline{\tau}$. The set of ML types is denoted MLType and the set of of ML type schemes is denoted MLTypeScheme.

Raw source expressions are defined by the following grammar:

$$
\begin{aligned}
e ::= {} & x : \overline{\tau} \\
\mid \ & (\lambda x : \overline{\tau_1}.e) : \overline{\tau} \\
\mid \ & (e_1\ e_2) : \overline{\tau} \\
\mid \ & (\texttt{letrec}\ f : \overline{\sigma}\ (x : \overline{\tau_1})\ \texttt{=}\ e_1\ \texttt{in}\ e_2) : \overline{\tau}
\end{aligned}
$$

They may be thought of as the result of ML-style type inference. We use e, e_1, e_2, ... to range over raw source expressions. An *ML type environment*, $\overline{TE}$, is a finite map from program variables to ML type schemes. The type system which allows inference of sentences of the form $\overline{TE} \vdash e : \overline{\tau}$ is that of Damas and Milner[4], which we do not reproduce here. The set of *source expressions* is denoted SExp and is defined to be the set of raw source expressions e for each of which there exists some ML type environment $\overline{TE}$ and ML type $\overline{\tau}$ such that $\overline{TE} \vdash e : \overline{\tau}$. Henceforth, we are concerned with source expressions (as opposed to raw source expressions) and we use e to range over source expressions.

Below we define the concept of free program variables of a source expression e. For reasons that will become clear later on, the order of the free variables of some object will sometimes be important to us. Therefore, in the formal development we will let all our free variable operations return finite sequences of variables, as defined in Section 2.1.

For a source expression e, $\mathrm{fv}(e)$ denotes the sequence of free program variables in e. Formally, we define:

$$\begin{aligned}
\mathrm{fv}(x : \overline{\tau}) &= x \\
\mathrm{fv}(\lambda x : \overline{\tau_1}.e : \overline{\tau}) &= \mathrm{fv}(e) \setminus \{x\} \\
\mathrm{fv}(e_1\ e_2 : \overline{\tau}) &= \mathrm{fv}(e_1) \,\overline{\cup}\, \mathrm{fv}(e_2) \\
\mathrm{fv}(\texttt{letrec } f : \overline{\sigma}\ (x : \overline{\tau_1}) \texttt{ = } e_1 \texttt{ in } e_2 : \overline{\tau}) &= ((\mathrm{fv}(e_1) \setminus \{x\}) \,\overline{\cup}\, \mathrm{fv}(e_2)) \setminus \{f\}
\end{aligned}$$

For a source expression e, MLTypeOf(e), denotes the ML type of e:

$$\begin{aligned}
\mathrm{MLTypeOf}(x : \overline{\tau}) &= \overline{\tau} \\
\mathrm{MLTypeOf}(\lambda x : \overline{\tau_1}.e : \overline{\tau}) &= \overline{\tau} \\
\mathrm{MLTypeOf}(e_1\ e_2 : \overline{\tau}) &= \overline{\tau} \\
\mathrm{MLTypeOf}(\texttt{letrec } f : \overline{\sigma}\ (x : \overline{\tau_1}) \texttt{ = } e_1 \texttt{ in } e_2 : \overline{\tau}) &= \overline{\tau}
\end{aligned}$$

2.3 Target Expressions

In this section we present the target language of region inference. We first define raw target expressions and then present the semantic region inference rules of Tofte and Talpin [16,17]; the semantic region inference rules define the target expressions.

Let RegVar and EffVar be two denumerably infinite sets of *region variables* and *effect variables*, pairwise disjoint. Assume further that RegVar and EffVar are also disjoint from ProgVar and TyVar. We let Var denote the union ProgVar $\cup$ RegVar $\cup$ TyVar $\cup$ EffVar. We let ρ range over region variables; ϵ range over effect variables; and ϑ range over the union of region, type, and effect variables. In summary:

$$\begin{aligned}
x, f &\in \mathrm{ProgVar} \\
\rho &\in \mathrm{RegVar} \\
\alpha &\in \mathrm{TyVar} \\
\epsilon &\in \mathrm{EffVar} \\
\vartheta &\in \mathrm{RegVar} \cup \mathrm{TyVar} \cup \mathrm{EffVar}
\end{aligned}$$

2.3.1 Raw Target Expressions

Raw target expressions are defined by the following grammar:

$$\begin{aligned} e' ::= {} & x \mid f\texttt{[}\vec{\rho}\texttt{]}\ \texttt{at}\ \rho \mid \lambda x.e'\ \texttt{at}\ \rho \mid e'_1 e'_2 \\ \mid {} & \texttt{letrec}\ f\texttt{[}\vec{\rho}\texttt{](}x\texttt{)}\ \texttt{at}\ \rho\ \texttt{=}\ e'_1\ \texttt{in}\ e'_2 \\ \mid {} & \texttt{letregion}\ \vec{\rho}\ \texttt{in}\ e' \end{aligned}$$

The set of raw target expressions is denoted TExp'. We use e', e'_1, e'_2, ... to range over TExp'.

2.3.2 Semantic Region Inference Rules

Definition 1 Atomic effects, *η, are defined by the following grammar:*

$$\eta ::= \rho \mid \epsilon$$

Equality of atomic effects is defined to be term equality.

Definition 2 *A* semantic effect *$\underline{\varphi}$ is a finite set of atomic effects, that is, a finite set of region and effect variables. The set of all semantic effects is denoted $\underline{\mathit{Effect}}$.*

Definition 3 *An* arrow effect, *$\epsilon.\underline{\varphi}$, is a pair of an effect variable and a semantic effect.*

(The dot in $\epsilon.\underline{\varphi}$ does not signify any kind of binding but is only used as a mnemonic that ϵ labels the semantic effect $\underline{\varphi}$; this will be made precise in Section 2.4.1).

Definition 4 Semantic types, *$\underline{\tau}$, are defined by the following grammar:*

$$\underline{\tau} ::= \alpha \mid (\underline{\tau}, \rho) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}', \rho')$$

The effect $\underline{\varphi}$ on the function arrow is called the *latent effect*; it describes which regions a function of that type will possibly use when called. The set of all semantic types is denoted $\underline{\mathrm{Type}}$.

Equality of semantic effects is defined to be set equality (using the above defined notion of equality for the elements). Two arrow effects $\epsilon_1.\underline{\varphi_1}$ and $\epsilon_2.\underline{\varphi_2}$ are defined to be equal if and only if $\epsilon_1 = \epsilon_2$ and $\underline{\varphi_1} = \underline{\varphi_2}$. Equality of semantic types is defined to be term equality, up to set equality of semantic effects on arrow effects.

Sometimes, we shall also use $\underline{\rho}$ to range over region variables, and $\underline{\epsilon}$ to range over effect variables.

Definition 5 *The* arrow effects *of* $\underline{\tau}$*, written* $arreff(\underline{\tau})$*, is the set of arrow effects defined by*

$$arreff(\alpha) = \emptyset$$

$$arreff((\underline{\tau}, \rho) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}', \rho')) = \{\epsilon.\underline{\varphi}\} \cup arreff(\underline{\tau}) \cup arreff(\underline{\tau}')$$

Temporarily, let o range over semantic effects, semantic arrow effects, and semantic types. Then $\mathrm{ftv}(o)$ denotes the set of free type variables in o; $\mathrm{frv}(o)$ denotes the set of free region variables in o; $\mathrm{fev}(o)$ denotes the set of free effect variables in o; and $\mathrm{fv}(o)$ denotes the set of free variables of either kind in o.

These free variable functions are defined in more or less the obvious way, except that, formally, they yield sequences of variables (see Appendix A for details).

Definition 6 *A* semantic region variable substitution*,* $\underline{S_r}$*, is a a map from region variables to region variables. A* semantic type variable substitution*,* $\underline{S_t}$*, is a map from type variables to semantic types. A* semantic effect variable substitution*,* $\underline{S_e}$*, is a map from effect variables to arrow effects. A* semantic substitution*,* $\underline{S}$*, is a triple* $(\underline{S_r}, \underline{S_t}, \underline{S_e})$ *consisting of a semantic region variable substitution, a semantic type variable substitution, and a semantic effect variable substitution.*

Application of a semantic substitution $\underline{S} = (\underline{S_r}, \underline{S_t}, \underline{S_e})$ to a semantic object is defined to be the simultaneous application of $\underline{S_r}$, $\underline{S_t}$, and $\underline{S_e}$:

Effects

$$\begin{aligned}\underline{S}(\underline{\varphi}) &= \{\underline{S_r}(\rho) \mid \rho \in \underline{\varphi}\} \\ &\cup \bigcup\{\{\epsilon'\} \cup \underline{\varphi}' \mid \epsilon'.\underline{\varphi}' = \underline{S_e}(\epsilon) \wedge \epsilon \in \underline{\varphi}\}\end{aligned}$$

Arrow Effects

$$\underline{S}(\epsilon.\underline{\varphi}) = \epsilon'.\left(\underline{\varphi}' \cup (\underline{S}(\underline{\varphi}))\right) \qquad \text{where } \underline{S_e}(\epsilon) = \epsilon'.\underline{\varphi}'$$

Semantic Types

$$\begin{aligned}\underline{S}(\alpha) &= \underline{S_t}(\alpha) \\ \underline{S}((\underline{\tau}, \rho) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}', \rho')) &= (\underline{S}(\underline{\tau}), \underline{S_r}(\rho)) \xrightarrow{\underline{S}(\epsilon.\underline{\varphi})} (\underline{S}(\underline{\tau}'), \underline{S_r}(\rho'))\end{aligned}$$

Semantic substitutions compose and $\underline{\mathrm{Id}} = (\underline{\mathrm{Id}_r}, \underline{\mathrm{Id}_t}, \underline{\mathrm{Id}_e})$ is the identity semantic substitution, defined by $\underline{\mathrm{Id}_r}(\rho) = \rho$, for all $\rho \in \mathrm{RegVar}$; $\underline{\mathrm{Id}_t}(\alpha) = \alpha$, for all $\alpha \in \mathrm{TyVar}$; and $\underline{\mathrm{Id}_e}(\epsilon) = \epsilon.\emptyset$, for all $\epsilon \in \mathrm{EffVar}$.

Definition 7 *The* support *of a semantic region variable substitution, written* $\mathrm{Supp}(\underline{S_r})$, *is the set* $\{\rho \in \mathit{RegVar} \mid \underline{S_r}(\rho) \neq \rho\}$. *The* support *of a semantic type variable substitution, written* $\mathrm{Supp}(\underline{S_t})$, *is the set* $\{\alpha \in \mathit{TyVar} \mid \underline{S_t}(\alpha) \neq \alpha\}$. *The* support *of a semantic effect variable substitution, written* $\mathrm{Supp}(\underline{S_e})$, *is the set* $\{\epsilon \in \mathit{EffVar} \mid \underline{S_e}(\epsilon) \neq \epsilon.\emptyset\}$. *The* support *of a semantic substitution* $\underline{S}$, *written* $\mathrm{Supp}(\underline{S})$, *is the union of the support of its three components.*

Whenever $\underline{S_r}$, $\underline{S_t}$, and $\underline{S_e}$ are finite maps of the appropriate types, we take the liberty to consider the triple $(\underline{S_r}, \underline{S_t}, \underline{S_e})$ a semantic substitution, without explicitly extending the finite maps to total maps.

Further, for a finite semantic effect variable substitution, $\underline{S_e}$, we will take the liberty to consider it as the semantic substitution $(\underline{\mathrm{Id}_r}, \underline{\mathrm{Id}_t}, \underline{S_e})$. (This way, application of a semantic effect variable substitution is defined by the definition of application for a semantic substitution.)

Definition 8 Semantic type schemes, $\underline{\sigma}$, *resemble the type schemes of Damas and Milner [4] but with additional quantification over region variables and effect variables, and are defined by the following grammar:*

$$\underline{\sigma} ::= \forall \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon}.\underline{\tau}$$

subject to the constraint that the variables in $\vec{\alpha}$, $\vec{\rho}$, *and* $\vec{\epsilon}$ *must all be distinct.*

The bound variables of a semantic type scheme $\underline{\sigma} = \forall \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon}.\underline{\tau}$, denoted $\mathrm{bv}(\underline{\sigma})$, is the sequence $\vec{\alpha} \cup \vec{\rho} \cup \vec{\epsilon}$. The free variables of a semantic type scheme $\underline{\sigma} = \forall \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon}.\underline{\tau}$, denoted $\mathrm{fv}(\underline{\sigma})$, is the sequence $\mathrm{fv}(\underline{\tau}) \setminus \mathrm{bv}(\underline{\sigma})$. Semantic type schemes that arises from each other by renaming of bound variables are considered equal. The set of all semantic type schemes is denoted $\underline{\mathrm{TypeScheme}}$.

Definition 9 *A semantic type* $\underline{\tau'}$ *is an* instance *of a semantic type scheme* $\underline{\sigma} = \forall \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon}.\underline{\tau}$, *written* $\underline{\sigma} \geq \underline{\tau'}$, *if there exists a semantic substitution* $\underline{S}$ *such that* $\mathrm{Supp}(\underline{S}) \subseteq \mathrm{bv}(\underline{\sigma})$ *and* $\underline{S}(\underline{\tau}) = \underline{\tau'}$. *When we want to make* $\underline{S}$ *explicit, we say that* $\underline{\tau'}$ *is an instance of* $\underline{\sigma}$ via $\underline{S}$, *written* $\underline{\sigma} \geq \underline{\tau'}$ via $\underline{S}$.

Definition 10 *A* semantic type environment *is a finite map from program variables to pairs of the form* $(\underline{\sigma}, \rho)$ *or* $(\underline{\tau}, \rho)$. *We use TE to range over semantic type environments and the set of all semantic type environments is denoted TypeEnv.*

More formally, a semantic type environment is a finite map from program variables to the disjoint union of $\underline{\mathrm{TypeScheme}} \times \mathrm{RegVar}$ and $\underline{\mathrm{Type}} \times \mathrm{RegVar}$. However, to ease the notation we shall never write the injections into (and the projections out of) the disjoint union. Informally speaking, it will always be the case that program variables that are λ-bound will be mapped to a pair of the form $(\underline{\tau}, \rho)$ and program variables that are `letrec`-bound will be mapped to a pair of the form $(\underline{\sigma}, \rho)$. The free variables of a semantic type environment TE is defined as follows:

$$\mathrm{fv}(TE) = \mathrm{seq}\left(\bigcup\{\,|\mathrm{fv}(TE(x))| \mid x \in \mathrm{Dom}(TE)\}\right)$$

The free variables of a pair $(TE, \underline{\varphi})$ of a type environment TE and a semantic effect $\underline{\varphi}$ is defined to be $\mathrm{fv}(TE) \mathbin{\overline{\cup}} \mathrm{fv}(\underline{\varphi})$. Let $\vec{\alpha}$ be a sequence of type variables, $\vec{\rho}$ be a sequence of region variables, and $\vec{\epsilon}$ a sequence of effect variables. Then $\mathrm{fv}(\vec{\alpha}, \vec{\epsilon}) = \vec{\alpha}\,\vec{\epsilon}$ and $\mathrm{fv}(\vec{\alpha}, \vec{\rho}, \vec{\epsilon}) = \vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}$.

Definition 11 *Let $\underline{\varphi}$ be a semantic effect and let TE be a type environment. Further, let $\underline{\tau}$ be a semantic type and let ρ be a region variable. Then we define* the observable part of $\underline{\varphi}$ with respect to TE, $\underline{\tau}$, and ρ, *written* $\mathrm{Observe}(TE, (\underline{\tau}, \rho), \underline{\varphi})$ *, to be the following subset of $\underline{\varphi}$:*

$$\mathrm{Observe}(TE, (\underline{\tau}, \rho), \underline{\varphi}) = \underline{\varphi} \cap (\mathrm{fv}(TE) \cup \mathrm{fv}(\underline{\tau}) \cup \{\rho\})$$

Definition 12 *The* ML erasure *of a semantic type $\underline{\tau}$, denoted* $\mathrm{ML}(\underline{\tau})$, *is the underlying ML type of $\underline{\tau}$; it is defined as follows:*

$$\begin{aligned} \mathrm{ML}(\alpha) &= \alpha \\ \mathrm{ML}((\underline{\tau}, \rho) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}', \rho')) &= \mathrm{ML}(\underline{\tau}) \to \mathrm{ML}(\underline{\tau}') \end{aligned}$$

Further, the ML erasure of a semantic type scheme $\underline{\sigma}$, *denoted* $\mathrm{ML}(\underline{\sigma})$, *is the underlying ML type scheme of $\underline{\sigma}$; it is defined as follows:*

$$\mathrm{ML}(\forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau}) = \forall\vec{\alpha}\,.\,\mathrm{ML}(\underline{\tau})$$

The semantic region inference rules allow the inference of sentences of the form

$$TE \vdash e \Rightarrow e' : (\underline{\tau}, \rho), \underline{\varphi}$$

read: *in TE, e translates to e', which has semantic type $\underline{\tau}$ and place ρ and semantic effect $\underline{\varphi}$.* Formally, the relation $TE \vdash e \Rightarrow e' : (\underline{\tau}, \rho), \underline{\varphi}$ is defined by rule induction [1,11] on the set TypeEnv × SExp × TExp' × <u>Type</u> × RegVar × <u>Effect</u> by the rules in Figures 2 and 3. Recall that e as a source expression is explicitly ML-typed, see the definition of source expression in Section 2.2.

In rule (2), the resulting effect is empty, since, in a call-by-value language, simply referring to a λ-bound variable x accesses the environment, but not the store. For a `letrec`-bound variable f, however, we access the region ρ' in which the region-polymorphic function resides and then produce a closure in a perhaps different region ρ; see rule (3). In rule (4), the effect of e becomes the latent effect of the lambda abstraction. The rule allows for an increase of effects, just before they are put onto function arrows. In rule (6) notice that

$$\frac{TE(x) = (\underline{\tau}, \rho) \qquad \mathrm{ML}(\underline{\tau}) = \overline{\tau}}{TE \vdash x : \overline{\tau} \Rightarrow x : (\underline{\tau}, \rho), \emptyset} \tag{2}$$

$$\frac{\begin{array}{c} TE(f) = (\underline{\sigma}, \rho') \qquad \underline{\sigma} = \forall \vec{\alpha}\, \rho_1 \cdots \rho_k \vec{\epsilon}.\underline{\tau}' \\ \underline{\sigma} \geq \underline{\tau} \text{ via } \underline{S} \qquad \underline{\varphi} = \{\rho', \rho\} \qquad \mathrm{ML}(\underline{\tau}) = \overline{\tau} \end{array}}{TE \vdash f : \overline{\tau} \Rightarrow f\texttt{[}\underline{S}(\rho_1)\texttt{,}\ldots\texttt{,}\underline{S}(\rho_k)\texttt{]}\ \texttt{at}\ \rho : (\underline{\tau}, \rho), \underline{\varphi}} \tag{3}$$

$$\frac{\begin{array}{c} TE + \{x \mapsto (\underline{\tau_1}, \rho_1)\} \vdash e \Rightarrow e' : (\underline{\tau_2}, \rho_2), \underline{\varphi} \qquad \underline{\varphi} \subseteq \underline{\varphi}' \\ \mathrm{ML}(\underline{\tau_1}) = \overline{\tau_1} \qquad \mathrm{ML}((\underline{\tau_1}, \rho_1) \xrightarrow{\epsilon.\underline{\varphi}'} (\underline{\tau_2}, \rho_2)) = \overline{\tau} \end{array}}{TE \vdash \lambda x : \overline{\tau_1}.e : \overline{\tau} \Rightarrow \lambda x.e'\ \texttt{at}\ \rho : ((\underline{\tau_1}, \rho_1) \xrightarrow{\epsilon.\underline{\varphi}'} (\underline{\tau_2}, \rho_2), \rho), \{\rho\}} \tag{4}$$

$$\frac{\begin{array}{c} TE \vdash e_1 \Rightarrow e_1' : ((\underline{\tau_2}, \rho_2) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}, \rho), \rho_1), \underline{\varphi_1} \\ TE \vdash e_2 \Rightarrow e_2' : (\underline{\tau_2}, \rho_2), \underline{\varphi_2} \qquad \mathrm{ML}(\underline{\tau}) = \overline{\tau} \end{array}}{TE \vdash e_1\ e_2 : \overline{\tau} \Rightarrow e_1'\ e_2' : (\underline{\tau}, \rho), \underline{\varphi} \cup \underline{\varphi_1} \cup \underline{\varphi_2} \cup \{\epsilon, \rho_1\}} \tag{5}$$

Fig. 2. Semantic Region Inference Rules, Part 1

e_1 is analyzed in an environment where f may be polymorphic in regions and effects, but not in types. (Polymorphic recursion in types would lead to an undecidable type-inference problem [7].) For more explanation of the rules in Figures 2 and 3, see [17].

The set of *target expressions* is denoted TExp and is defined to be the set of raw target expressions e' for each of which there exists some type environment TE, some source expression e, some semantic type $\underline{\tau}$, some region variable ρ, and some semantic effect $\underline{\varphi}$, such that $TE \vdash e \Rightarrow e' : (\underline{\tau}, \rho), \underline{\varphi}$.

Remark 13 *Our formulation of the semantic region inference rules closely follows the newer formuluation of Tofte and Talpin [17] which is slightly different from the original formulation in [16]. (In particular, rule (6) does not force one to quantify all the variables that can be quantified.) However, we have included an additional rule (8), which allows one to increase the seman-*

$$\frac{\begin{array}{c} TE + \{f \mapsto (\forall \vec{\rho}\,\vec{\epsilon}.\underline{\tau_0}, \rho_0)\} \vdash \lambda\, x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow \\ \qquad\qquad \lambda\, x.e_1' \;\texttt{at}\; \rho_0 : (\underline{\tau_0}, \rho_0), \underline{\varphi_1} \\ \mathrm{fv}(\vec{\alpha}, \vec{\rho}, \vec{\epsilon}) \cap \mathrm{fv}(TE, \underline{\varphi_1}) = \emptyset \\ TE + \{f \mapsto (\forall \vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau_0}, \rho_0)\} \vdash e_2 \Rightarrow e_2' : (\underline{\tau}, \rho), \underline{\varphi_2} \\ \mathrm{ML}(\forall \vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau_0}) = \forall \vec{\alpha}.\overline{\tau_0} \qquad \mathrm{ML}(\underline{\tau}) = \overline{\tau} \end{array}}{\begin{array}{l} TE \vdash \texttt{letrec}\; f : \forall \vec{\alpha}.\overline{\tau_0}\; \texttt{(}x : \overline{\tau_1}\texttt{)} \;\texttt{=}\; e_1 \;\texttt{in}\; e_2 : \overline{\tau} \Rightarrow \\ \qquad \texttt{letrec}\; f\texttt{[}\vec{\rho}\texttt{]}\texttt{(}x\texttt{)} \;\texttt{at}\; \rho_0 \;\texttt{=}\; e_1' \;\texttt{in}\; e_2' : (\underline{\tau}, \rho), \underline{\varphi_1} \cup \underline{\varphi_2} \end{array}} \tag{6}$$

$$\frac{\begin{array}{c} TE \vdash e \Rightarrow e' : (\underline{\tau}, \rho), \underline{\varphi} \\ \underline{\varphi'} \supseteq \mathrm{Observe}(TE, (\underline{\tau}, \rho), \underline{\varphi}) \quad \vec{\rho} = \mathrm{frv}(\underline{\varphi} \setminus \underline{\varphi'}) \end{array}}{TE \vdash e \Rightarrow \texttt{letregion}\; \vec{\rho} \;\texttt{in}\; e' : (\underline{\tau}, \rho), \underline{\varphi'}} \tag{7}$$

$$\frac{TE \vdash e \Rightarrow e' : (\underline{\tau}, \rho), \underline{\varphi}}{TE \vdash e \Rightarrow e' : (\underline{\tau}, \rho), \underline{\varphi} \cup \underline{\varphi'}} \tag{8}$$

Fig. 3. Semantic Region Inference Rules, Part 2

tic effect[4] *and we also allow to observe away less than the non-observable part of the semantic effect. These changes do not invalidate the soundness of region inference proved in [17].*

2.4 Implementation-Oriented Formulation of Region Inference

In this section we will impose some restrictions on the semantic region inference rules. The restrictions are expressed by an inference system that can be seen as an implementation-oriented formulation of region inference. The inference system is based on one that was developed as a foundation for the syntax-directed algorithm [14], but the present system is simpler and permits more derivations.

Later in this paper we show that our constraint-based formulation of region

[4] Using the new rule (8), one may omit the condition $\underline{\varphi} \subseteq \underline{\varphi'}$ in rule (4).

inference is complete with respect to the implementation-oriented inference rules.

The main restriction we shall impose is the notion of consistency, which allows to implement arrow effects by graphs, as explained below. Besides consistency we shall require type schemes to be "well-formed" and in "normal-form;" this is done for technical reasons and does not limit the expressive power of the system in interesting ways. We do not essentially restrict the quantification in type schemes, as was done in [14] where only so-called primary variables were allowed to be quantified. As a consequence, the present algorithm can infer more general types that the one in [14]. All these issues are discussed in more detail in the following.

2.4.1 Consistency

Definition 14 *Let* Φ *be a set of arrow effects. We say that* Φ *is* effect consistent*, written* $\vdash \Phi$*, if*

(1) Φ *is* functional*:* $\forall(\epsilon_1.\underline{\varphi_1}) \in \Phi.\forall(\epsilon_2.\underline{\varphi_2}) \in \Phi.(\epsilon_1 = \epsilon_2 \implies \underline{\varphi_1} = \underline{\varphi_2})$*;*
(2) Φ *is* closed*:* $\forall(\epsilon.\underline{\varphi}) \in \Phi.\forall\epsilon' \in \underline{\varphi}.\exists\underline{\varphi'}.((\epsilon'.\underline{\varphi'}) \in \Phi)$*;*
(3) Φ *is* transitive*:* $\forall(\epsilon_1.\underline{\varphi_1}) \in \Phi.\forall(\epsilon_2.\underline{\varphi_2}) \in \Phi.(\epsilon_2 \in \underline{\varphi_1} \implies \underline{\varphi_2} \subseteq \underline{\varphi_1})$.

We define $\mathrm{frv}(\Phi) = \cup\{\mathrm{frv}(\epsilon.\underline{\varphi}) \mid \epsilon.\underline{\varphi} \in \Phi\}$ *and* $\mathrm{fev}(\Phi) = \cup\{\mathrm{fev}(\epsilon.\underline{\varphi}) \mid \epsilon.\underline{\varphi} \in \Phi\}$*. For any functional* Φ*, the* effect map of Φ *, written* $\widehat{\Phi}$*, is the map from effect variables to effects defined by* $\widehat{\Phi}(\epsilon) = \underline{\varphi}$*, if* $\epsilon.\underline{\varphi} \in \Phi$*.*

Definition 15 *A* basis *is a pair* $B = (Q, \Phi)$*, where* Q *is a finite set of region variables and* Φ *is a finite set of arrow effects. The set of bases is called* Basis*. We use* A*,* B*, and* D *to range over bases.*

We say that $B = (Q, \Phi)$ *is* consistent*, written* $\vdash B$*, if* Φ *is effect consistent and* $Q \supseteq \mathrm{frv}(\Phi)$*.*

We define $\mathrm{frv}(B) = Q \cup \mathrm{frv}(\Phi)$ *and* $\mathrm{fev}(B) = \mathrm{fev}(\Phi)$*. The empty basis* $(\emptyset, \emptyset)$ *is denoted* $B_\emptyset$*. When* B *is a basis we write* Q *of* B *and* Φ *of* B *for the first and second component of* B*, respectively. The* domain *of* $B = (Q, \Phi)$*, written* $\mathrm{Dom}(B)$*, is the set* $Q \cup \mathrm{Dom}(\widehat{\Phi})$*.*

The concept of basis formalises certain finite, directed graphs, called *effect graphs*. Every node is labelled by a region or an effect variable. Writing B in the form (Q, Φ), there is a node for each $\rho \in Q$; moreover, for each $\epsilon.\varphi = \epsilon.\{\eta_1, \ldots, \eta_n\}$ there is a node labelled ϵ and edges from this node to the nodes labelled by $\eta_1, \ldots, \eta_n$, respectively. Thus if we regard region variables as singleton sets, effect graphs essentially represent set constraints: there is an edge from node a to node b if there is a constraint saying that b is a subset

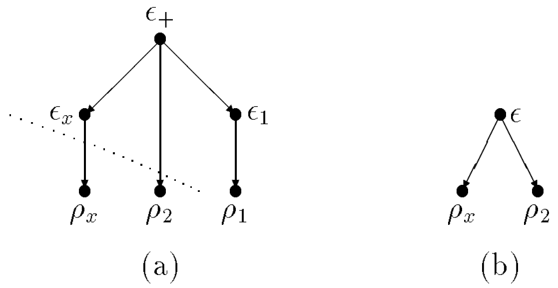

Fig. 4. An effect graph (a) before application of `letregion` and (b) after.

of a.

Effect graphs are used in the implementation of the Tofte-Birkedal algorithm [14] and also in the implementation of the constraint normalization algorithm of this paper. In the Tofte-Birkedal algorithm, there is one arrow effect for every subexpression of the program. For example, in the type environment $TE = \{\mathtt{x} \mapsto (\mathtt{int}, \rho_x)\}$, the expression `(x + (1 at` ρ_1`))at` ρ_2 gives rise to a graph shown in Figure 4(a). Applying rule (7) gives the expression `letregion` ρ_1 `in (x + (1 at` ρ_1`))at` ρ_2 , which has the effect graph shown in Figure 4(b). The dotted line in (a) separates the part of the graph that is kept from the part that is discharged. This partitioning of graphs is formalised by the notion of "disjoint union" of bases defined below.

Definition 16 *Let $B_1 = (Q_1, \Phi_1)$ and $B_2 = (Q_2, \Phi_2)$ be bases (they need not be consistent). We define the* union *of B_1 and B_2, written $B_1 \cup B_2$, to be $(Q_1 \cup Q_2, \Phi_1 \cup \Phi_2)$. We define the* disjoint union *of B_1 and B_2, written $B_1 \uplus B_2$, to be $B_1 \cup B_2$ provided $B_1 \cup B_2$ is consistent, $Q_1 \cap Q_2 = \emptyset$ and $\mathrm{Dom}(\widehat{\Phi_1}) \cap \mathrm{Dom}(\widehat{\Phi_2}) = \emptyset$, and undefined otherwise.*

When we form $B_1 \uplus B_2$, B_1 will be consistent and B_2 will be functional and transitive, but not closed. Then the requirements for the existence of $B_1 \uplus B_2$ express that $B_1 \uplus B_2$ is a consistent extension of B_1 with region and effect variables that are not in B_1.

Definition 17 *We now define the relation $B \vdash (\underline{\tau}, \rho)$. Let $B = (Q, \Phi)$ be a basis. We say that a semantic object o is* consistent *in B, if the sentence $B \vdash o$ can be inferred from the following rules:*

$$\frac{\vdash B \qquad \rho \in (Q \text{ of } B)}{B \vdash \rho} \qquad \frac{\vdash B \qquad \epsilon.\underline{\varphi} \in (\Phi \text{ of } B)}{B \vdash \epsilon.\underline{\varphi}} \qquad \frac{\vdash B \qquad \eta \in \mathrm{fv}(B)}{B \vdash \eta}$$

$$\frac{\vdash B}{B \vdash \alpha} \qquad \frac{B \vdash \tau \qquad B \vdash \rho}{B \vdash (\tau, \rho)} \qquad \frac{B \vdash (\underline{\tau_1}, \rho_1) \quad B \vdash (\underline{\tau_2}, \rho_2) \quad B \vdash \epsilon.\underline{\varphi}}{B \vdash (\underline{\tau_1}, \rho_1) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau_2}, \rho_2)}$$

Note that in the rule for function types, $\underline{\varphi}$ is uniquely determined by B and ϵ: $\underline{\varphi} = \widehat{\Phi}(\epsilon)$, where $\Phi = \Phi$ of B.

Definition 18 *For semantic effects, we define consistency as follows: a semantic effect $\underline{\varphi}$ is* consistent *in $B = (Q, \Phi)$, written $B \vdash \underline{\varphi}$, if B is consistent,* $\text{frv}(\underline{\varphi}) \subseteq Q$, $\text{fev}(\underline{\varphi}) \subseteq \text{Dom}(\widehat{\Phi})$ *and moreover, for all $\epsilon'.\underline{\varphi'} \in \Phi$, $\epsilon' \in \underline{\varphi}$ implies $\underline{\varphi'} \subseteq \underline{\varphi}$.*

As usual, we often write "$B \vdash o$" to mean "there exists a proof tree with conclusion $B \vdash o$." It is easy to see that $B \vdash o$ implies that B is consistent, that $\text{frv}(o) \subseteq (Q \text{ of } B)$ and that $\text{fev}(o) \subseteq \text{Dom}(\widehat{\Phi})$, where $\Phi = \Phi$ of B.

Inclusion $B \subseteq B'$ is defined to be componentwise inclusion and difference $B \setminus B'$ is componentwise set difference. For brevity, empty Q or Φ components are sometimes omitted; for example, $B \uplus \{\rho\}$ abbreviates $B \uplus (\{\rho\}, \emptyset)$.

2.4.2 Consistent Type Schemes

Definition 19 *We say that a semantic type scheme $\underline{\sigma} = \forall \vec{\alpha}\vec{\rho}\vec{\epsilon}.\underline{\tau}$ is* well-formed *if, for every arrow effect $\epsilon.\underline{\varphi}$ occurring on a function arrow in τ, if $\epsilon \notin \text{bv}(\sigma)$ then $\text{fv}(\underline{\varphi}) \cap \text{bv}(\sigma) = \emptyset$. Otherwise, $\underline{\sigma}$ is said to be* ill-formed. *We say that a semantic type environment TE is* well-formed *if, for all $x \in \text{Dom}(TE)$, if $TE(x) = (\underline{\sigma}, \rho)$ then $\underline{\sigma}$ is well-formed.*

The implementation-oriented inference system requires type schemes to be well-formed. This is a natural consequence of the idea of basing region inference on consistent effect graphs. If one allowed type schemes to be ill-formed, consistency of the basis could be compromised by instantiation of type schemes. For example, consider the type scheme $\sigma = \forall \rho.(\texttt{int}, \rho_0) \xrightarrow{\epsilon.\{\rho\}} (\texttt{int}, \rho_0)$ which is ill-formed. The problem with σ is that if it is instantiated twice, via different substitutions, say $S_1 = \{\rho \mapsto \rho_1\}$ and $S_2 = \{\rho \mapsto \rho_2\}$, then the two resulting types $(\texttt{int}, \rho_0) \xrightarrow{\epsilon.\{\rho_1\}} (\texttt{int}, \rho_0)$ and $(\texttt{int}, \rho_0) \xrightarrow{\epsilon.\{\rho_2\}} (\texttt{int}, \rho_0)$ are not consistent in any basis. This clearly does not agree well with basing region inference on consistency (or, for that matter, on set constraints). This example also illustrates that if one wants region polymorphism and consistency, then one must also allow effect polymorphism.

Conversely, by restricting attention to well-formed type schemes, we have the advantage that the same device which underlies binding at `letregion`, namely, disjoint union of bases (Section 2.4.1), can account for binding in type schemes as well. We can use disjoint union of bases to separate the 'bound' part of the arrow effects mentioned in the body of the type scheme from the free parts. This is exploited in the definition of consistent type scheme below.

The definition of well-formedness does not put any bound on the number of bound region and effect variables a type scheme may contain. If one is not careful, one may end up designing a region inference algorithm that generates an unbounded number of bound variables (see [14] for an example). Very briefly, the problem is with types in which there are region and effect variables that occur in effects on function arrows without also occurring paired with a type, such as ρ in the type scheme $\forall \rho\epsilon.(\texttt{int}, \rho_0) \xrightarrow{\epsilon.\{\rho\}} (\texttt{int}, \rho_0)$. However, as pointed out in [14], there is in a certain sense an upper bound on the number of bound region and effect variables that are needed in a semantic type scheme. To explain this, we need the notion of primary and secondary variables. Informally speaking, the primary region variables in a semantic type $\underline{\tau}$ are those region variables in $\underline{\tau}$ that occur paired with a type and the primary effect variables in $\underline{\tau}$ are those effect variables in $\underline{\tau}$ that occur paired with an effect as an arrow effect. See [14] for a thorough discussion of primary and secondary variables. For a semantic type, $\underline{\tau}$, $\mathrm{fprv}(\underline{\tau})$ denotes the set of free primary region variables; $\mathrm{fpev}(\underline{\tau})$ denotes the set of free primary effect variables; and $\mathrm{fprev}(\underline{\tau})$ denotes the set of free primary region or effect variables. Formally we define:

Definition 20 *For a semantic type, $\underline{\tau}$, $\mathrm{fprv}(\underline{\tau})$, $\mathrm{fpev}(\underline{\tau})$ and fprev are defined as follows:*

$$\mathrm{fprv}(\alpha) = \emptyset \qquad \mathrm{fprv}((\underline{\tau}, \rho) \xrightarrow{\epsilon.\varphi} (\underline{\tau}', \rho')) = \mathrm{fprv}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{fprv}(\underline{\tau}') \mathbin{\overline{\cup}} \rho \mathbin{\overline{\cup}} \rho'$$

$$\mathrm{fpev}(\alpha) = \emptyset \qquad \mathrm{fpev}((\underline{\tau}, \rho) \xrightarrow{\epsilon.\varphi} (\underline{\tau}', \rho')) = \mathrm{fpev}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{fpev}(\underline{\tau}') \mathbin{\overline{\cup}} \epsilon$$

$$\mathrm{fprev}(\underline{\tau}) = \mathrm{fprv}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{fpev}(\underline{\tau})$$

Given a semantic type $\underline{\tau}$, a region or effect variable is said to be primary *in $\underline{\tau}$ if it is a member of the set $\mathrm{fprev}(\underline{\tau})$; and it is said to be* secondary *in $\underline{\tau}$ if it is a member of the set $\mathrm{fv}(\underline{\tau}) \setminus \mathrm{fprev}(\underline{\tau})$.*

Given a semantic type $\underline{\tau}$ with n function arrows, one can use the structure of $\underline{\tau}$ to enumerate the subsets of the n arrows in some order such that each subset is given a number $j \in \{1, \ldots, 2^n\}$. We assume given such an enumeration function.

Definition 21 *We define the* index *of a region variable ρ in a semantic type $\underline{\tau}$ to be the number of the subset of arrows on which ρ occurs in the arrow effect on the arrow, if ρ is secondary in $\underline{\tau}$, and otherwise, if ρ is primary in $\underline{\tau}$, $2^n + j$ where j is the (unique) number satisfying $\mathrm{fprv}(\underline{\tau}) = \rho_1, \ldots, \rho_j, \ldots, \rho_k$ and $\rho = \rho_j$. We denote the index of ρ in $\underline{\tau}$ by $index(\rho, \underline{\tau})$. Likewise one can* *define the index of an effect variable ϵ in a semantic type $\underline{\tau}$; we denote it by $index(\epsilon, \underline{\tau})$.*

For example, consider the following semantic type, where we have numbered the function arrows:

$$\underline{\tau} = (\alpha, \rho') \xrightarrow{\epsilon_1.\{\rho\} \quad (1)} (\alpha, \rho') \xrightarrow{\epsilon_2.\{\} \quad (2)} (\alpha, \rho'').$$

The index of ρ in $\underline{\tau}$ is 2 if the enumeration function enumerates the subsets of arrows as {}, {1}, {2},{1,2}, and the index of ρ'' in $\underline{\tau}$ is $2^2 + 2 = 6$, since $\mathrm{fprv}(\underline{\tau}) = \rho', \rho''$.

Consider a well-formed semantic type scheme $\sigma = \forall\vec{\alpha}\vec{\rho}\vec{\epsilon}.\underline{\tau}$ whose body, $\underline{\tau}$, is consistent in some basis and assume that σ has two different, bound region variables with the same index. Note that these two region variables must be secondary. It turns out that σ has precisely the same type instances as the type scheme that arises by removing all occurrences of one of the two region variables from $\underline{\tau}$ [14]. Similarly for secondary, bound effect variables.

Therefore, in a certain sense, no well-formed semantic type scheme with a consistent body needs more than 2^n bound secondary region variables, where n is the number of arrows in the underlying ML type scheme (namely one secondary region variable for each subset of the n arrows). Likewise, one needs at most 2^n bound secondary effect variables.

Definition 22 *A semantic type scheme $\underline{\sigma} = \forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau}$ is in* normal form *if, writing $\vec{\rho}$ as $\rho_1 \cdots \rho_k$ and $\vec{\epsilon}$ as $\epsilon_1 \cdots \epsilon_l$ and letting n be the number of function arrows in $\underline{\tau}$, all of the following conditions hold:*

(1) $\vec{\alpha} \subseteq \mathrm{ftv}(\underline{\tau})$;
(2) $\mathrm{length}(\vec{\rho} \setminus \mathrm{fprv}(\underline{\tau})) = 2^n$ *and* $\mathrm{length}(\vec{\epsilon} \setminus \mathrm{fpev}(\underline{\tau})) = 2^n$;
(3) $\forall\rho \in (\mathrm{frv}(\underline{\tau}) \setminus \mathrm{fprv}(\underline{\tau})) \cap \vec{\rho}.\ index(\rho, \underline{\tau}) = j \implies \rho = \rho_j$;
(4) $\forall\epsilon \in (\mathrm{fev}(\underline{\tau}) \setminus \mathrm{fpev}(\underline{\tau})) \cap \vec{\epsilon}.\ index(\epsilon, \underline{\tau}) = j \implies \epsilon = \epsilon_j$.

(Recall that, by definition, for all semantic type schemes $\underline{\sigma} = \forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau}$, the bound variables are pairwise distinct.)

For example, with $\underline{\tau}$ as in the previous example,

$$\underline{\sigma} = \forall\rho_1, \rho, \rho_3, \rho_4, \rho'', \epsilon_{11}, \epsilon_{12}, \epsilon_{13}, \epsilon_{14}, \epsilon_1.\underline{\tau}$$

is a well-formed type scheme in normal form.

Note that item (2) in the definition of normal form can always be met by simply introducing quantified variables which do not occur in the body of the type scheme. Thus the essential requirement is that there is no pair of different bound secondary region variables with the same index (and likewise for effect variables).

In the implementation-oriented formulation of region inference, we shall require that type schemes are in normal form. The essential reason for this is that it gives us an upper bound on the number of bound secondary variables and thus an upper bound on the number of variables needed for instantiation of secondary variables. The idea is that it allows us to set up constraints which express which variables should be used for instances of secondary variables in a type scheme. In fact it allows us to prove that our constraint-based formulation of region inference is complete with respect to the implementation-oriented formulation of region inference.

We are now ready to extend the definition of consistency to cover semantic type schemes.

Definition 23 *A semantic type scheme* $\underline{\sigma} = \forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau}$ *is* consistent in basis B, *written* $B \vdash \underline{\sigma}$, *if* B *is consistent,* $\underline{\sigma}$ *is in normal form, and there exists a basis* B' *with* $\mathrm{Dom}(B') = \vec{\rho} \cup \vec{\epsilon}$ *and such that* $B \uplus B'$ *exists, and* $B \uplus B' \vdash \underline{\tau}$.

Lemma 24 *If* $B \vdash \underline{\sigma}$ *then* $\underline{\sigma}$ *is well-formed.*

PROOF. See [14]. □

The definition of consistency does not depend on any particular choice of bound variables in $\underline{\sigma}$, except that the bound variables must be chosen disjoint from B.

Definition 25 *We say that a semantic type environment* TE *is* consistent in basis B, *written* $B \vdash TE$, *if for all* $x \in \mathrm{Dom}(TE)$ *we have* $B \vdash TE(x)$, *where* $B \vdash (\underline{\tau}, \rho)$ *means* $B \vdash \underline{\tau} \wedge B \vdash \rho$ *and where* $B \vdash (\underline{\sigma}, \rho)$ *means* $B \vdash \underline{\sigma} \wedge B \vdash \rho$. *For arbitrary semantic objects we write* $B \vdash (o_1, \ldots, o_n)$ *to mean* $B \vdash o_1 \wedge \cdots \wedge B \vdash o_n$.

In order to state and prove completeness of the constraint-based formulation of region inference it is convenient that all the bound variables of type schemes in a semantic region inference derivation are distinct and that we have a precise account on exactly which set of variables are in play, for each subexpression. Therefore we shall make use of the notion of a "cone" [14].

Definition 26 *A* cone *is a pair* (A, B) *of bases such that* $\vdash B$, $\vdash A$ *and* $A \supseteq B$. *We let Cone denote the set of all cones. We usually write cones in the form* $\binom{A}{B}$.

The implementation oriented formulation of region inference allow the inference of sentences of the form $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$ and is defined by rule induction on the set Cone × TypeEnv × SExp × TExp′ × Type × RegVar × EffVar by the rules in Figures 5 and 6 (the rules make use of the definition of the sum of two cones which we give below).

In a sentence $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$, the B is a basis in which TE, $\underline{\tau}$, $\underline{\rho}$, and $\underline{\varphi}$ are consistent, while A is the basis obtained from B and e by forming the union of B, $\{\underline{\epsilon}.\underline{\varphi}\}$, and all the bases that are discharged locally within e by quantification or by means of Observe. The term "cone" is intended to suggest that the proof tree which has $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$ in its conclusion has the shape of a cone, with the upper part of $\binom{A}{B}$ being the basis which covers the proof tree that has $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$ as its conclusion.

To express that bound variables in different type schemes are all distinct we need to be sure that bases that are discharged by different subexpressions are kept distinct. For this we shall use the notion of "sum" of two cones.

Definition 27 *Let $\binom{B_1'}{B_1}$ and $\binom{B_2'}{B_2}$ be cones. The* sum *of $\binom{B_1'}{B_1}$ and $\binom{B_2'}{B_2}$, written $\binom{B_1'}{B_1} \uplus \binom{B_2'}{B_2}$, is defined by*

$$\binom{B_1'}{B_1} \uplus \binom{B_2'}{B_2} = \begin{cases} \binom{B_1' \cup B_2'}{B_1} & \textit{if } B_1 = B_2 \textit{ and } \mathrm{Dom}(B_1') \cap \mathrm{Dom}(B_2') \subseteq \mathrm{Dom}(B_1) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

In the inference rules, we write "$A = B \uplus C$ exists" as a shorthand for "$B \uplus C$ exists and equals A."

Lemma 28 *If $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$ then $B \vdash (TE, \underline{\tau}, \underline{\rho}, \underline{\varphi})$, $A \vdash \underline{\epsilon}.\underline{\varphi}$, and $\underline{\epsilon} \notin \mathrm{fv}(B)$.*

PROOF. By rule induction on $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$. □

The reason why we have chosen to ensure that in a derivation $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$, we have $A \vdash \underline{\epsilon}.\underline{\varphi}$ is that for simplicity the constraint-based formulation essentially only works with arrow effects, and $\underline{\epsilon}.\underline{\varphi}$ will then be the arrow effect used to represent the effect $\underline{\varphi}$. It is easy to see that A will always be of the form $A' \uplus \{\underline{\epsilon}.\underline{\varphi}\}$ which simply expresses that $\underline{\epsilon}$ is a fresh variable not used anywhere else in the derivation.

The basis A_1 is used in rule 10 to ensure that everything the instantiating

$$\frac{\begin{array}{c} TE(x) = (\underline{\tau}, \underline{\rho}) \qquad \mathrm{ML}(\underline{\tau}) = \overline{\tau} \\ B \vdash (TE, (\underline{\tau}, \underline{\rho})) \qquad A = B \uplus \{\underline{\epsilon}.\emptyset\} \quad \text{exists} \end{array}}{\binom{A}{B}, TE \vdash x : \overline{\tau} \Rightarrow x : (\underline{\tau}, \underline{\rho}), \emptyset, \underline{\epsilon}} \tag{9}$$

$$\frac{\begin{array}{c} TE(f) = (\underline{\sigma}, \underline{\rho'}) \qquad \underline{\sigma} = \forall \vec{\alpha}\, \underline{\rho_1} \cdots \underline{\rho_k} \vec{\epsilon}.\underline{\tau'} \\ \underline{\sigma} \geq \underline{\tau} \text{ via } \underline{S} \qquad \mathrm{ML}(\underline{\tau}) = \overline{\tau} \\ B \vdash (TE, (\underline{\tau}, \underline{\rho})) \qquad \underline{\varphi} = \{\underline{\rho'}, \underline{\rho}\} \\ A_1 = (\{\underline{S}(\rho) \mid \rho \in \underline{\rho_1} \cdots \underline{\rho_k} \setminus \mathrm{frv}(\underline{\tau'})\}, \{\underline{S}(\epsilon) \mid \epsilon \in \vec{\epsilon} \setminus \mathrm{fev}(\underline{\tau'})\}) \\ A_2 = B \uplus A_1 \quad \text{exists} \qquad A = A_2 \uplus \{\underline{\epsilon}.\underline{\varphi}\} \quad \text{exists} \end{array}}{\binom{A}{B}, TE \vdash f : \overline{\tau} \Rightarrow f\,\texttt{[}\underline{S}(\underline{\rho_1})\texttt{,}\ldots\texttt{,}\underline{S}(\underline{\rho_k})\texttt{]}\ \texttt{at}\ \underline{\rho} : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}} \tag{10}$$

$$\frac{\begin{array}{c} \binom{A_1}{B}, TE + \{x \mapsto (\underline{\tau_1}, \underline{\rho_1})\} \vdash e \Rightarrow e' : (\underline{\tau_2}, \underline{\rho_2}), \underline{\varphi_2}, \underline{\epsilon_2} \qquad \underline{\varphi_2} \subseteq \underline{\varphi_3} \\ \mathrm{ML}(\underline{\tau_1}) = \overline{\tau_1} \qquad \mathrm{ML}((\underline{\tau_1}, \underline{\rho_1}) \xrightarrow{\epsilon_3.\varphi_3} (\underline{\tau_2}, \underline{\rho_2})) = \overline{\tau} \\ B \vdash TE \qquad (\{\underline{\rho}\}, \{\underline{\epsilon_3}.\underline{\varphi_3}\}) \subseteq B \qquad A = A_1 \uplus \{\underline{\epsilon}.\{\underline{\rho}\}\} \quad \text{exists} \end{array}}{\binom{A}{B}, TE \vdash \lambda x : \overline{\tau_1}.e : \overline{\tau} \Rightarrow \lambda x.e'\ \texttt{at}\ \underline{\rho} : ((\underline{\tau_1}, \underline{\rho_1}) \xrightarrow{\epsilon_3.\varphi_3} (\underline{\tau_2}, \underline{\rho_2}), \underline{\rho}), \{\underline{\rho}\}, \underline{\epsilon}} \tag{11}$$

$$\frac{\begin{array}{c} \binom{A_1}{B}, TE \vdash e_1 \Rightarrow e_1' : ((\underline{\tau_2}, \underline{\rho_2}) \xrightarrow{\epsilon_3.\varphi_3} (\underline{\tau}, \underline{\rho}), \underline{\rho_1}), \underline{\varphi_1}, \underline{\epsilon_1} \\ \binom{A_2}{B}, TE \vdash e_2 \Rightarrow e_2' : (\underline{\tau_2}, \underline{\rho_2}), \underline{\varphi_2}, \underline{\epsilon_2} \qquad \mathrm{ML}(\underline{\tau}) = \overline{\tau} \\ \binom{A_3}{B} = \binom{A_1}{B} \uplus \binom{A_2}{B} \qquad \underline{\varphi_4} = \underline{\varphi_1} \cup \underline{\varphi_2} \cup \underline{\varphi_3} \cup \{\underline{\epsilon_3}, \underline{\rho_1}\} \\ A_4 = A_3 \uplus \{\underline{\epsilon_4}.\underline{\varphi_4}\} \quad \text{exists} \end{array}}{\binom{A_4}{B}, TE \vdash e_1\ e_2 : \overline{\tau} \Rightarrow e_1'\ e_2' : (\underline{\tau}, \underline{\rho}), \underline{\varphi_4}, \underline{\epsilon_4}} \tag{12}$$

Fig. 5. Implementation Oriented Formulation of Semantic Region Inference Rules, Part 1

$$
\frac{\begin{array}{c}
B \vdash (TE, \underline{\varphi_1}) \qquad \underline{\sigma} = \forall \vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau_0} \qquad \hat{\underline{\sigma}} = \forall \vec{\rho}\,\vec{\epsilon}.\underline{\tau_0} \\
\vec{\alpha} \cap \operatorname{ftv}(TE) = \emptyset \qquad \operatorname{Dom}(B_1) = \vec{\rho} \cup \vec{\epsilon} \qquad B \uplus B_1 \quad \text{exists} \\
\left(\begin{smallmatrix} A_1 \\ B \uplus B_1 \end{smallmatrix}\right), TE + \{f \mapsto (\hat{\underline{\sigma}}, \underline{\rho_0})\} \vdash \lambda\, x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow \\
\qquad\qquad \lambda\, x.e'_1 \;\texttt{at}\; \underline{\rho_0} : (\underline{\tau_0}, \underline{\rho_0}), \underline{\varphi_1}, \underline{\epsilon_1} \\
\left(\begin{smallmatrix} A_2 \\ B \end{smallmatrix}\right), TE + \{f \mapsto (\underline{\sigma}, \underline{\rho_0}))\} \vdash e_2 \Rightarrow e'_2 : (\underline{\tau}, \underline{\rho}), \underline{\varphi_2}, \underline{\epsilon_2} \\
\underline{\varphi_4} = \underline{\varphi_1} \cup \underline{\varphi_2} \\
\underline{\varphi_3} = (\operatorname{frv}(\underline{\tau_0}) \setminus \vec{\rho}\,) \cup \bigcup \{\{\epsilon\} \cup \underline{\varphi} \mid \epsilon \in \operatorname{frv}(\underline{\tau_0}) \setminus \vec{\epsilon} \wedge \epsilon.\underline{\varphi} = \widehat{\Phi \text{ of } B}(\epsilon)\} \\
\left(\begin{smallmatrix} A_3 \\ B \end{smallmatrix}\right) = \left(\begin{smallmatrix} A_1 \\ B \end{smallmatrix}\right) \uplus \left(\begin{smallmatrix} A_2 \\ B \end{smallmatrix}\right) \qquad A_4 = A_3 \uplus \{\underline{\epsilon_3}.\underline{\varphi_3}, \underline{\epsilon_4}.\underline{\varphi_4}\} \quad \text{exists} \\
\operatorname{ML}(\underline{\sigma}) = \forall \vec{\alpha}\,.\overline{\tau_0} \qquad \operatorname{ML}(\underline{\tau}) = \overline{\tau}
\end{array}}{\begin{array}{l}
\left(\begin{smallmatrix} A_4 \\ B \end{smallmatrix}\right), TE \vdash \texttt{letrec}\ f : \forall \vec{\alpha}.\overline{\tau_0}\ \ (x : \overline{\tau_1})\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\
\qquad\qquad \texttt{letrec}\ f\texttt{[}\vec{\rho}\,\texttt{](}x\texttt{)}\ \texttt{at}\ \underline{\rho_0}\ \texttt{=}\ e'_1\ \texttt{in}\ e'_2 : (\underline{\tau}, \underline{\rho}), \underline{\varphi_4}, \underline{\epsilon_4}
\end{array}} \tag{13}
$$

$$
\frac{\begin{array}{c}
B \uplus B_1 \quad \text{exists} \qquad B \vdash (TE, (\underline{\tau}, \underline{\rho})) \\
\left(\begin{smallmatrix} A_4 \\ B \uplus B_1 \end{smallmatrix}\right), TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi_4}, \underline{\epsilon_4} \\
\underline{\varphi} = \operatorname{Observe}(TE, (\underline{\tau}, \underline{\rho}), \underline{\varphi_4}) \quad \vec{\rho} = \operatorname{frv}(\underline{\varphi_4} \setminus \underline{\varphi}) \\
A = A_4 \uplus \{\underline{\epsilon}.\underline{\varphi}\} \quad \text{exists}
\end{array}}{\left(\begin{smallmatrix} A \\ B \end{smallmatrix}\right), TE \vdash e \Rightarrow \texttt{letregion}\ \vec{\rho}\ \ \texttt{in}\ e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}} \tag{14}
$$

$$
\frac{\begin{array}{c}
\left(\begin{smallmatrix} A_1 \\ B \end{smallmatrix}\right), TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi_1}, \underline{\epsilon_1} \\
\underline{\varphi_1} \subseteq \underline{\varphi} \qquad B \vdash \underline{\varphi} \qquad A = A_1 \uplus \{\underline{\epsilon}.\underline{\varphi}\} \quad \text{exists}
\end{array}}{\left(\begin{smallmatrix} A \\ B \end{smallmatrix}\right), TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}} \tag{15}
$$

Fig. 6. Implementation Oriented Formulation of Semantic Region Inference Rules, Part 2

substitution $\underline{S}$ maps bound variables to is collected in the basis A of the conclusion cone $\left(\begin{smallmatrix} A \\ B \end{smallmatrix}\right)$.

In rule 13, $\epsilon_3.\underline{\varphi_3}$ is used to collect the monomorphic part of $\underline{\tau_0}$ for proof-technical reasons; it is explained in the following section.

The implementation-oriented region inference rules are very similar to the rules used in [14]. The crucial difference lies in the definition of consistent type scheme. In [14] only primary variables are allowed to be quantified; we have no such restriction here. The following example illustrates that, as a consequence, the constaint-based algorithm can infer more polymorphic type schemes than the algorithm in [14].

Consider the expression

```
letrec f(x) =
  letrec g(y) = λz.if z>100 then y+z else g(y)(z+1)
  in g(x) end
in f(1)(2) + f(3)(4)
end
```

The Tofte-Talpin rules allow the following type scheme for `f`:

$$\forall\rho_1\rho_2\rho_3\rho_4\rho_5\epsilon_1\epsilon_2.(\mathtt{int},\rho_1)\xrightarrow{\epsilon_1.\{\rho_5,\rho_4\}}((\mathtt{int},\rho_2)\xrightarrow{\epsilon_2.\{\rho_1,\rho_2,\rho_3,\rho_4\}}(\mathtt{int},\rho_3),\rho_5)$$

where the secondary region variable ρ_4 is the region where the closure for `g` is stored. Consequently, the closure created by `f(1)` can be de-allocated after the evaluation of `f(1)(2)`. Similarly, the closure created by `f(3)` can be de-allocated after the evaluation of `f(3)(4)`. This is also what the constraint-based algorithm achieves. By contrast, the algorithm in [14] does not permit quantification of ρ_4. As a consequence, the two closures for `f(1)` and `f(3)` are put in a more global region which is only de-allocated after the entire outermost `letrec` expression has been fully evaluated.

3 Region Inference Using Constraints

3.1 Representation of Semantic Objects by Syntactic Objects

In this section we introduce the objects our constraint-based algorithm uses; we call these objects *syntactic* objects because they are simple terms not containing sets (as opposed to the semantic objects of region inference which contain effect sets). The idea is that the syntactic objects are used to represent the semantic objects of region inference. For example, effect variables represent semantic effects. *Constraints* are formulae which contain effect variables and express requirements of the corresponding semantic objects. Informally, a *solution substitution* is a map from syntactic objects to semantic objects that satisfy the requirements of the constraints.

Types and semantic effects are represented by syntatic objects in a relatively straightforward manner. A semantic type scheme $\underline{\sigma} = \forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau}$ is represented by a pair $(\tau, \vec{m})$, where τ is a syntatic type representing $\underline{\tau}$. Still informally speaking, the $\vec{m}$ component represents the type environment TE prevailing at the point where the type scheme $\underline{\sigma}$ has been formed. The idea is that in the semantic region inference rules, the type environment TE and the effect $\underline{\varphi}$ tells which variables of $\underline{\tau}$ are allowed to be quantified: only those variables, which do not occur in TE or $\underline{\varphi}$ (see rule (6)). Thus $\vec{m}$ expresses which set of variables of the semantic type corresponding to τ that is not allowed to be quantified, i.e., the set of variables, which should be monomorphic. For this reason $\vec{m}$ is called a syntactic mono object sequence. A mono object, m, is simply a type, region, or effect variable.

We first define syntactic objects (Section 3.1.1) and free variables of syntactic objects (Section 3.1.2). We then define the application of semantic substitutions to syntactic objects (Section 3.1.3) and solution substitutions (Section 3.1.4). Then we define the representation of semantic type schemes (Section 3.1.5) and of semantic type environments (Section 3.1.6).

3.1.1 Syntactic Objects

Definition 29 Syntactic effects, *$\overline{\varphi}$, syntactic types, τ, syntactic mono objects, m, and* syntactic type schemes, *σ, are defined as follows:*

$$
\begin{aligned}
\overline{\varphi} &::= \rho \mid \mathrm{Obs}(\Gamma, (\tau, \rho), \epsilon) \mid \epsilon \mid \overline{\epsilon} \\
\tau &::= \alpha \mid (\tau, \rho) \xrightarrow{\epsilon} (\tau', \rho') \\
m &::= \rho \mid \epsilon \mid \alpha \\
\sigma &::= (\tau, \vec{m})
\end{aligned}
$$

where $\vec{m}$ ranges over syntactic mono object sequences, following the convention for sequences established in the preceding, and Γ ranges over syntactic type environments: A syntactic type environment *Γ is a pair consisting of (1) a finite map from program variables to pairs of the form (σ, ρ) or (τ, ρ), and (2) a mono object sequence $\vec{m}$ satisfying the following conditions:*

(1) $\forall x \in \mathrm{Dom}(\Gamma).\Gamma(x) = (\tau, \rho) \implies \mathrm{fv}(\tau, \rho) \subseteq \vec{m}$*;*
(2) $\forall f \in \mathrm{Dom}(\Gamma).\Gamma(f) = (\sigma, \rho) \wedge \sigma = (\tau', \vec{m}') \implies \vec{m}' \subseteq \vec{m}$*;*
(3) $\forall \alpha \in \mathrm{ftv}(\vec{m}).\Big((\exists(\tau, \rho) \in \mathrm{Rng}(\Gamma).\alpha \in \mathrm{ftv}(\tau)) \vee$
$(\exists((\tau', \vec{m}'), \rho) \in \mathrm{Rng}(\Gamma).\alpha \in \mathrm{ftv}(\tau') \wedge \alpha \in \mathrm{ftv}(\vec{m}'))\Big)$.

The set of all syntactic types is denoted Type. We use Γ to range over syntactic type environments and the set of all syntactic type environments is denoted SynTypeEnv.

To simplify notation, we use the following conventions:

(1) We write $\Gamma(x)$ for the application of the finite map of Γ to x.
(2) We let mono(Γ) denote the second component of Γ.
(3) We write $\Gamma + \{x \mapsto (\tau, \rho)\}$ as shorthand for the syntactic type environment with first component the finite map of Γ extended with x bound to (τ, ρ) and with second component the second component of Γ extended with $\mathrm{fv}(\tau) \,\overline{\cup}\, \rho$.
(4) We write $\Gamma + \{f \mapsto ((\tau, \vec{m}), \rho)\}$ as a shorthand for the syntactic type environment with first component the finite map of Γ extended with x bound to $((\tau, \vec{m}), \rho)$ and with second component $\vec{m}$. (Note that this is always well-defined if $\vec{m}$ contains mono(Γ) and $\forall \alpha \in \mathrm{ftv}(\vec{m} \backslash \mathrm{mono}(\Gamma)).\alpha \in \mathrm{ftv}(\tau)$.)

Equality of syntactic effects, syntactic types, syntactic mono objects, and sequences of these syntactic objects is defined to be term equality.

3.1.2 Free Variables of Syntactic Objects

For a syntactic type τ, $\mathrm{ftv}(\tau)$ denotes the set of free type variables in τ; $\mathrm{frv}(\tau)$ denotes the set of free region variables in τ; $\mathrm{fev}(\tau)$ denotes the set of free effect variables in τ; and $\mathrm{fv}(\tau)$ denotes the set of free variables of either kind in τ. Formally, the free variable functions yield in fact sequences of variables (the reason for this will become clear later, c.f. Section 3.3), the free variable functions are defined in the obvious way. For the details, consult Appendix B.

3.1.3 Application of Semantic Substitutions to Syntactic Objects

Corresponding to syntactic mono objects, we define semantic mono objects as follows.

Definition 30 Semantic mono objects*, $\underline{m}$, are given by the following grammar*

$$\underline{m} ::= \rho \mid \epsilon.\underline{\varphi} \mid \underline{\tau}$$

The application of a semantic substitution to semantic mono object and semantic mono object sequences is defined in the obvious way; see Appendix C for details. Semantic substitutions can also be applied to syntactic objects; such an application results in a semantic object. The application of a semantic substitution $\underline{S} = (\underline{S_r}, \underline{S_t}, \underline{S_e})$ to a variable, to a syntactic type, to a syntactic mono object, and to a syntactic mono object sequence is defined to be the simultaneous application of $\underline{S_r}$, $\underline{S_t}$, and $\underline{S_e}$, see Appendix D for details.

3.1.4 Solution Substitutions

Definition 31 *A* solution substitution, *$\mathcal{S}$, is a finite semantic substitution for which there exists a consistent basis D and a consistent basis A such that* $\mathrm{Dom}(\mathcal{S}) \subseteq \mathrm{Dom}(D)$ *and* $\mathcal{S}(D) \subseteq A$.

Note that $\mathcal{S}(D)$ is not required to be consistent. Moreover, note that $\mathrm{Dom}(\mathcal{S}) \cap \mathrm{TyVar} = \emptyset$. We write SolutionSubst for the set of all solution substitutions.

Remark 32 *In the actual constraint-based algorithm, the basis D will be the simple basis with domain the free (region and effect) variables in the constraint system, and $\mathcal{S}$ will be a contraction ([14]) of D, so $\mathcal{S}(D)$ will in fact be consistent. The extra flexibility in the above definition is useful when proving completeness of the constraint-based formulation of region inference. In an arbitrary solution, a syntactic effect variable ϵ could be mapped by $\mathcal{S}$ to $\epsilon'.\underline{\varphi}$ with some $\epsilon'' \in \underline{\varphi}$ and $\epsilon'' \notin \mathrm{fv}(\mathcal{S}(D))$ but $\epsilon'' \in \mathrm{Dom}(A)$.*

Lemma 33 *Let D_1, D_2, and A be consistent bases and let $\mathcal{S}_1$ and $\mathcal{S}_2$ be solution substitutions such that* $\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_1)$ *and* $\mathrm{Dom}(\mathcal{S}_2) \subseteq \mathrm{Dom}(D_2)$, *$\mathcal{S}_1(D_1) \subseteq A$, and $\mathcal{S}_2(D_2) \subseteq A$. If* $\mathrm{Dom}(D_1) \cap \mathrm{Dom}(D_2) = \emptyset$, *then $\mathcal{S}_1 + \mathcal{S}_2$ is a solution substitution.*

PROOF. Let $D = D_1 \cup D_2$ and $A = A$ in the definition of solution substitution. $\square$

3.1.5 Representation of Semantic Type Schemes

Definition 34 *We say that* a syntactic type scheme $\sigma = (\tau, \vec{m})$ represents a semantic type scheme $\underline{\sigma} = \forall \vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau}$ with respect to solution substitution $\mathcal{S}$, *written $\sigma \simeq_{\mathcal{S}} \underline{\sigma}$, if the following conditions all hold*

(1) $\underline{\tau} = \mathcal{S}(\tau)$;
(2) $\vec{\alpha} = \mathrm{ftv}(\underline{\tau}) \setminus \mathrm{ftv}(\mathcal{S}(\vec{m}))$;
(3) $\vec{\rho}\,' = \mathrm{frv}(\underline{\tau}) \setminus \mathrm{frv}(\mathcal{S}(\vec{m}))$;
(4) $\vec{\epsilon}\,' = \mathrm{fev}(\underline{\tau}) \setminus \mathrm{fev}(\mathcal{S}(\vec{m}))$;
(5) *$\underline{\sigma}$ is in normal form;*
(6) *$\underline{\sigma}$ is well-formed;*
(7) $\vec{\rho}\,' \subseteq \vec{\rho}$;
(8) $\vec{\epsilon}\,' \subseteq \vec{\epsilon}$;
(9) $(\vec{\rho} \setminus \vec{\rho}\,') \cap \mathrm{frv}(\mathcal{S}(\vec{m})) = \emptyset$;
(10) $(\vec{\epsilon} \setminus \vec{\epsilon}\,') \cap \mathrm{fev}(\mathcal{S}(\vec{m})) = \emptyset$.

Here (3) and (4) may be regarded as defining equations; $\vec{\rho}\,'$ and $\vec{\epsilon}\,'$ are the bound variables that appear in the body of the type scheme (items (7) and (8));

any other bound variables must be regarded as non-generic in the syntactic representation (see items (9) and (10)). Since $\underline{\sigma}$ is in normal form it can (and normally will) contain bound variables which do occur in the body of the type scheme.

3.1.6 Representation of Type Environments

Definition 35 *We say that* Γ represents TE with respect to solution substitution $\mathcal{S}$*, written* $\Gamma \simeq_{\mathcal{S}} TE$*, if the following conditions all hold:*

(1) $\mathrm{Dom}(\Gamma) = \mathrm{Dom}(TE)$*;*
(2) $\forall x \in \mathrm{Dom}(\Gamma).\Gamma(x) = (\tau, \rho) \implies \Big(\exists \underline{\tau} \exists \underline{\rho}.\, TE(x) = (\underline{\tau}, \underline{\rho}) \wedge \mathcal{S}(\tau) = \underline{\tau} \wedge \mathcal{S}(\rho) = \underline{\rho}\Big)$*;*
(3) $\forall f \in \mathrm{Dom}(\Gamma).\Gamma(f) = (\sigma, \rho) \implies \Big(\exists \underline{\sigma} \exists \underline{\rho}.\, TE(x) = (\underline{\sigma}, \underline{\rho}) \wedge \mathcal{S}(\rho) = \underline{\rho} \wedge \sigma \simeq_{\mathcal{S}} \underline{\sigma} \wedge ((\tau, \vec{m}) = \sigma \implies \rho \in \mathrm{frv}(\vec{m}))\Big)$*.*

The following lemma is immediate from the definitions.

Lemma 36

(1) If $\sigma = (\tau, \vec{m}) \simeq_{\mathcal{S}} \underline{\sigma}$*, then* $\mathrm{fv}(\underline{\sigma}) \subseteq \mathrm{fv}(\mathcal{S}(\vec{m}))$*.*
(2) If $\Gamma \simeq_{\mathcal{S}} TE$ *and* $\vec{m} = \mathrm{mono}(\Gamma)$*, then* $\mathrm{fv}(TE) \subseteq \mathrm{fv}(\mathcal{S}(\vec{m}))$ *and* $\mathrm{ftv}(\vec{m}) \subseteq \mathrm{ftv}(TE)$*.*

Using the fact that two semantic type schemes are considered equal if they arise from each other by renaming of bound variables, we get the following uniqueness lemma.

Lemma 37 *If* $\sigma \simeq_{\mathcal{S}} \underline{\sigma}$ *and* $\sigma \simeq_{\mathcal{S}} \underline{\sigma'}$ *then* $\underline{\sigma} = \underline{\sigma'}$*.*

Definition 38 *We say that* a syntactic type scheme $\sigma = (\tau, \vec{m})$ is in normal form with respect to solution substitution $\mathcal{S}$ *if*

(1) $\forall \rho, \rho' \in \mathrm{frv}(\mathcal{S}(\tau)) \setminus \mathrm{frv}(\mathcal{S}(\vec{m})).\, \mathit{index}(\rho, \mathcal{S}(\tau)) = \mathit{index}(\rho', \mathcal{S}(\tau)) \implies \rho = \rho'$*;*
(2) $\forall \epsilon, \epsilon' \in \mathrm{frv}(\mathcal{S}(\tau)) \setminus \mathrm{frv}(\mathcal{S}(\vec{m})).\, \mathit{index}(\epsilon, \mathcal{S}(\tau)) = \mathit{index}(\epsilon', \mathcal{S}(\tau)) \implies \epsilon = \epsilon'$*.*

The condition of normal form is sufficient for existence:

Lemma 39 *Let* σ *be a syntactic type scheme and* $\mathcal{S}$ *a solution substituion such that* σ *is in normal form with respect to* $\mathcal{S}$*. Then there exists a semantic type scheme* $\underline{\sigma}$ *such that* $\sigma \simeq_{\mathcal{S}} \underline{\sigma}$*.*

PROOF. Write σ as $(\tau, \vec{m})$. Now let $(j \in 1, \dots, 2^n)$

- $\vec{\alpha} = \mathrm{ftv}(\mathcal{S}(\tau)) \setminus \mathrm{ftv}(\mathcal{S}(\vec{m}))$;
- $\vec{\rho}\,' = \mathrm{frv}(\mathcal{S}(\tau)) \setminus \mathrm{frv}(\mathcal{S}(\vec{m}))$;
- $\vec{\epsilon}\,' = \mathrm{fev}(\mathcal{S}(\tau)) \setminus \mathrm{fev}(\mathcal{S}(\vec{m}))$;
- $n =$ number of arrows in τ;
- $\rho_j =$ (if $\exists \rho' \in \vec{\rho}\,'$ with $index(\rho', \mathcal{S}(\tau)) = j$ then ρ' else a fresh region variable ρ' such that $\rho' \notin \mathrm{frv}(\mathcal{S}(\tau))$, $\rho' \notin \mathrm{frv}(\mathcal{S}(\vec{m}))$, and all ρ_j distinct);
- $\epsilon_j =$ (if $\exists \epsilon' \in \vec{\epsilon}\,'$ with $index(\epsilon', \mathcal{S}(\tau)) = j$ then ϵ' else a fresh effect variable ϵ' such that $\epsilon' \notin \mathrm{fev}(\mathcal{S}(\tau))$, $\epsilon' \notin \mathrm{fev}(\mathcal{S}(\vec{m}))$, and all ϵ_j distinct);
- $\vec{\rho} = \rho_1, \dots \rho_{2^n} \mathbin{\overline{\cup}} (\vec{\rho}\,' \setminus \rho_1, \dots \rho_{2^n})$;
- $\vec{\epsilon} = \epsilon_1, \dots \epsilon_{2^n} \mathbin{\overline{\cup}} (\vec{\epsilon}\,' \setminus \epsilon_1, \dots \epsilon_{2^n})$;
- $\underline{\sigma} = \forall \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon}. \mathcal{S}(\tau)$.

It remains to show that $\sigma \simeq_{\mathcal{S}} \underline{\sigma}$. Only condition (6) in the definition of representation of semantic type schemes is not obvious. To see that this condition holds, i.e., that $\underline{\sigma}$ is well-formed suppose $\epsilon.\underline{\varphi} \in \mathit{arreff}(\mathcal{S}(\tau))$ and $\epsilon \notin \vec{\epsilon}$. Then there exists $\epsilon' \in \mathrm{fv}(\tau)$ such that $\epsilon' \in \mathrm{Dom}(\mathcal{S})$ and $\mathcal{S}(\epsilon') = \epsilon.\underline{\varphi}$. By the assumption that $\epsilon \notin \vec{\epsilon}$, there are two cases to consider:

Case (i) There exists an $\epsilon'' \in \vec{m}$ such that $\mathcal{S}(\epsilon'') = \epsilon.\underline{\varphi'}$. Then by the fact that $\mathcal{S}$ is a solution substitution, there exists a consistent basis A such that, in particular, $\mathcal{S}(\epsilon') = \epsilon.\underline{\varphi} \in A$ and $\mathcal{S}(\epsilon'') = \epsilon.\underline{\varphi'} \in A$. By consistency of A, we have that $\underline{\varphi} = \underline{\varphi'}$. Hence $\mathrm{fv}(\underline{\varphi}) \subseteq \mathrm{fv}(\mathcal{S}(\vec{m}))$ so $\mathrm{fv}(\underline{\varphi}) \cap \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon} = \emptyset$, as required.

Case (ii) There exists an $\epsilon'' \in \vec{m}$ such that $\mathcal{S}(\epsilon'') = \epsilon'''.\underline{\varphi'}$ and $\epsilon \in \underline{\varphi'}$. Again using that $\mathcal{S}$ is a solution substitution, we get a consistent basis A such that $\epsilon'''.\underline{\varphi'}, \epsilon.\underline{\varphi} \in A$. Hence by consistency of A, in particular closure of A, we have that $\underline{\varphi} \subseteq \underline{\varphi'}$. Thus $\mathrm{fv}(\underline{\varphi}) \subseteq \mathrm{fv}(\underline{\varphi'}) \subseteq \mathrm{fv}(\mathcal{S}(\vec{m}))$, from which again the required follows. $\square$

Combining the two previous lemmas we get that, for a syntactic type scheme σ and a solution substitution $\mathcal{S}$, if σ is in normal form w.r.t. $\mathcal{S}$, then there exists a unique semantic type scheme $\underline{\sigma}$ such that $\sigma \simeq_{\mathcal{S}} \underline{\sigma}$. We shall refer to this semantic type scheme as $\mathrm{mk}\_\underline{\sigma}(\mathcal{S}, \sigma)$.

Definition 40 *We say that* a syntactic type environment Γ is in normal form with respect to solution substitution $\mathcal{S}$ *if, for all x in* $\mathrm{Dom}(\Gamma)$*, if* $\Gamma(x) = (\sigma, \rho)$ *and $\sigma = (\tau, \vec{m})$, then σ is in normal form with respect to $\mathcal{S}$ and, moreover, $\rho \in \mathrm{frv}(\vec{m})$.*

By extending the above observations to syntactic type environments we get that, if Γ is in normal form with respect to $\mathcal{S}$, then there exists a unique semantic type environment TE such that $\Gamma \simeq_{\mathcal{S}} TE$. We shall refer to this semantic type environment as mk_$TE(\mathcal{S},\Gamma)$.

3.1.7 Representation of Observe

Definition 41 *The semantics of the term* $\mathrm{Obs}(\Gamma,(\tau,\rho),\epsilon)$ *with respect to* $\mathcal{S}$*, denoted*

$$\mathcal{O}_{\mathcal{S}}(\Gamma,(\tau,\rho),\epsilon),$$

is $\mathrm{Observe}(\mathit{TE},(\underline{\tau},\underline{\rho}),\underline{\varphi})$ *where* $\mathit{TE} = \mathit{mk\_TE}(\mathcal{S},\Gamma)$*,* $\underline{\tau} = \mathcal{S}(\tau)$*,* $\underline{\rho} = \mathcal{S}(\rho)$*, and* $\epsilon'.\underline{\varphi} = \mathcal{S}(\epsilon)$*, if* Γ *is in normal form with respect to* $\mathcal{S}$*, and undefined otherwise.*

3.2 Constraint Systems

In this section we define the notion of constraint system.

3.2.1 Constraints

Definition 42 *A* constraint *is a term of one of the following forms:*

(1) $\epsilon \doteq \epsilon'$*,* $\rho \doteq \rho'$*, or, provided* $\mathrm{ML}(\tau) = \mathrm{ML}(\tau')$*,* $\tau \doteq \tau'$*;*
(2) $\epsilon \dot{\succeq} \epsilon'$*,* $\rho \dot{\succeq} \rho'$*,* $\alpha \dot{\succeq} \tau'$*, or, provided* $\exists S_t : \mathrm{ML}(S_t(\tau)) = \mathrm{ML}(\tau')$ *and* $\mathrm{length}(\vec{\rho}) = 2^n + \mathrm{length}(\mathrm{frv}(\tau'))$ *and* $\mathrm{length}(\vec{\epsilon}) = 2^n + \mathrm{length}(\mathrm{fev}(\tau'))$ *where* n *is the number of function arrows in* τ*,* $(\tau,\vec{m}) \dot{\succeq} (\vec{\rho},\vec{\epsilon},\tau')$*;*
(3) $\epsilon \dot{\supseteq} \vec{\varphi}$*;*
(4) $\mathrm{Norm}(\tau,\vec{m})$*.*

Here the dots in the terms emphasise that a constraint is a syntactic term; for example, $\epsilon \doteq \epsilon'$ is a term, not a fact.

A solution to a constraint set $\mathcal{C}$ is a semantic substitution which solves the constraints in $\mathcal{C}$, as defined in detail below. For example, $\mathcal{S}$ solves the constraint $\epsilon \doteq \epsilon'$ if $\mathcal{S}(\epsilon) = \mathcal{S}(\epsilon')$. Constraints of the form (1) are called *equality constraints*. Constraints of the form (2) are called *instantiation constraints*. Informally, $\mathcal{S}$ solves $\rho \dot{\succeq} \rho'$ if $\mathcal{S}(\rho)$ is a bound variable of a type scheme of some program variable f and $\mathcal{S}(\rho')$ is the generic instance of $\mathcal{S}(\rho)$ at some occurrence of f. (We shall soon tag instantiation constraints by so-called colours, one colour for each occurrence of f to represent the fact that — as is usual with ML-style polymorphism — different occurrences of the same program variable may be given different generic instances of the same type scheme.) The idea of the constraint $(\tau,\vec{m}) \dot{\succeq} (\vec{\rho},\vec{\epsilon},\tau')$ is as follows. The pair $(\tau,\vec{m})$ is to be thought of as the syntactic type scheme of a program variable f and τ'

is the instance of the the type scheme at some occurrence of f. The vector $\vec{\rho}$ takes the form $\vec{\rho}\,\vec{\rho}\,'$. Here $\vec{\rho}$ has the length 2^n, where n is the number of function arrows in τ; the idea is that element number j of $\vec{\rho}$ is the generic instance of the bound secondary region variable with index j in the type scheme represented by $(\tau, \vec{m})$. Likewise for the effect variables.

Next, a constraint of the form $\epsilon \mathrel{\dot{\supseteq}} \vec{\vec{\varphi}}$ is called an *inclusion constraint*; informally, $\mathcal{S}$ solves it if, letting $\epsilon'.\underline{\varphi'} = \mathcal{S}(\epsilon)$, one has $\underline{\varphi'} \supseteq \underline{\varphi''}$, where $\underline{\varphi''}$ is the result of evaluating the syntactic effect $\vec{\vec{\varphi}}$ to a semantic effect.

Finally, a constraint of the form $\mathrm{Norm}(\tau, \vec{m})$ is a *normal form constraint*. Note that $(\tau, \vec{m})$ is a syntactic type scheme; informally, $\mathcal{S}$ solves the constraint if $(\tau, \vec{m})$ is in normal form with respect to $\mathcal{S}$.

Equality of constraints is defined to be term equality. We use c to range over constraints. The free variables of a constraint c are denoted $\mathrm{fv}(c)$ and is defined in the obvious way. The free variables of a finite multi-set of constraints $\mathcal{C}$ are denoted $\mathrm{fv}(\mathcal{C})$. The application of a syntactic substitution to a constraint is defined by extension in the obvious way; likewise for the application of a syntactic substitution to a finite multi-set of constraints.

3.2.2 Instantiation Maps

Although region inference by constraint solution separates the generation of constraints from the solution of constraints, it is useful to keep in the notion of constraint system a connection between instantiation constraints and the original expression. First, we use integers (so-called "colours") to tag instantiation constraints. Intuitively, there will be one colour for each non-binding occurrence of every program variable in the source program. Colours are chosen consecutively, starting from 1, so a set of colours always takes the form $\{1, \ldots, k\}$, $(k \geq 0)$. Next, a so-called instantiation map maps each colour $i \in \{1, \ldots, k\}$ to a quintuple of the form $(\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau')$, where $(\tau, \vec{m})$ is the type scheme of the program variable and τ' is the instance of the type scheme at the program point coloured i.

Definition 43 *An* instantiation map *(of rank k) is a map from $\{1, \ldots, k\}$ to quintuples of the form $(\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau')$ satisfying*

(1) $\mathrm{length}(\vec{\rho}\,) = 2^n + \mathrm{length}(\mathrm{frv}(\tau'))$*;*
(2) $\mathrm{length}(\vec{\epsilon}\,) = 2^n + \mathrm{length}(\mathrm{fev}(\tau'))$*;*
(3) there exists S_t such that $\mathrm{ML}(S_t(\tau)) = \mathrm{ML}(\tau')$*;*

where n is the number of function arrows in τ.

Here a quintuple $(\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau')$ corresponds to a constraint $(\tau, \vec{m}) \mathrel{\dot{\succeq}} (\vec{\rho}, \vec{\epsilon}, \tau')$. There is deliberate redundancy here: the components of a constraint $(\tau, \vec{m}) \mathrel{\dot{\succeq}} (\vec{\rho}, \vec{\epsilon}, \tau')$ are kept both in the constraint itself and in an instantiation map. This redundancy simplifies the description of solution for a set of constraints and the presentation of the constraint-based region inference rules.

3.2.3 Well-tagged Sets of Constraints

Definition 44 *A* tagged set of constraints (of rank k) *is a triple* $(\mathcal{C}, I, k)$ *where* I *is an instantiation map of rank* k *and* $\mathcal{C}$ *is a finite multiset of constraints with the property that every instantiation constraint in* $\mathcal{C}$ *is decorated by a colour (taken from the set* $\{1, \ldots, k\}$*).*

Not all tagged constraint sets correspond to real programs. We now define a notion of *well-tagged* constraint set. Well-tagged constraint sets are guaranteed to have a solution. Moreover, as we shall prove later on, the problem of finding a region annotation of a program according to the implementation-oriented region inference rules of Section 2.4 can be reduced to a problem of finding a solution to a well-tagged constraint set.

Definition 45 *A tagged set of constraints* $(\mathcal{C}, I, k)$ *is* well-tagged *if*

(1) Whenever $\alpha \mathrel{\dot{\succeq}^i} \tau \in \mathcal{C}$ *and* $\alpha \mathrel{\dot{\succeq}^i} \tau' \in \mathcal{C}$ *then* $\mathrm{ML}(\tau) = \mathrm{ML}(\tau')$.

(2) (a) For all $i \in \{1, \ldots, k\}$*, if* $(\tau, \vec{m}, \_, \_, \_) = I(i)$ *then* $\mathrm{Norm}(\tau, \vec{m}) \in \mathcal{C}$*;*

(b) For all $\epsilon \mathrel{\dot{\supseteq}} \vec{\vec{\varphi}} \in \mathcal{C}$*, for all* $\mathrm{Obs}(\Gamma, (\tau, \rho), \epsilon) \in \vec{\vec{\varphi}}$*, for all* $((\tau', \vec{m}'), \rho') \in \mathrm{Rng}(\Gamma)$ *one has* $\mathrm{Norm}(\tau', \vec{m}') \in \mathcal{C}$.

(3) (a) For all $(\tau, \vec{m}) \mathrel{\dot{\succeq}^i} (\vec{\rho}, \vec{\epsilon}, \tau') \in \mathcal{C}$*,* $I(i) = (\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau')$.

(b) For all $i \in \{1, \ldots, k\}$*,* $\mathrm{frev}(I(i)) \in \mathrm{frev}(\mathcal{C})$.

Here item (1) comes from the fact that we are only considering programs which are already well-typed according to the ML type discipline. No type variable can be instantiated to two different types at the same program point.

Next, item (2) expresses the idea that all type schemes are supposed to be in normal form. Finally, item (3) expresses redundancy between the instantiation map and the constraint set $\mathcal{C}$.

3.2.4 Constraint Systems

Solving region inference constraints seems to require a mixture of evaluation of effect expressions and simplification of constraints. For example, consider a constraint of the form

$$\epsilon \mathrel{\dot{\supseteq}} \mathrm{Obs}(\Gamma, (\tau, \rho), \epsilon').$$

In the final solution, we want ϵ to denote a set of region and effect variables, not a (perhaps complex) expression involving terms like Obs. Therefore, constraint solution naturally involves evaluation of terms such as $\mathrm{Obs}(\Gamma, (\tau, \rho), \epsilon')$. However, it is not possible to evaluate such a term once and then remove it from the constraint set. The effect which ϵ' denotes may increase later on in the solution process, in which case $\mathrm{Obs}(\Gamma, (\tau, \rho), \epsilon')$ may have to be re-evaluated and the effect denoted by ϵ perhaps increased.

Fortunately, we already have a semantic notion which captures the idea that every effect variable "denotes" an effect, namely the notion of consistent basis (Definition 15 in Section 2.4.1). Below we define a constraint system to be a pair $((C, I, k), D)$ of a well-tagged constraint set and a consistent basis $D = (Q, \Phi)$. The idea is that $\epsilon.\underline{\varphi} \in \Phi$ means that ϵ currently denotes the set $\underline{\varphi}$ (which is simply a set of region and effect variables).

The rewrite rules defined later on allow one to make rewrite steps of the form

$$((C, I, k), D) \Rightarrow ((C', I', k'), D')$$

in which not only the constraint set but also the basis is transformed. The following example illustrates the idea.

Example 46 *The constraint system* $((C, I, k), D) = ((\{\epsilon_1 \doteq \epsilon_2\}, \{\}, 0), (\{\rho_1, \rho_2\}, \{\epsilon_1.\{\rho_1\}, \epsilon_2.\{\rho_2\}\})$ *can be rewritten to* $((C', I', k'), D') = ((\emptyset, \{\}, 0), (\{\rho_1, \rho_2\}, \{\epsilon_1.\{\rho_1, \rho_2\}\})$.

Definition 47 *Let* $D = (Q, \Phi)$ *be a consistent basis. We define the* finite semantic substitution induced by D*, written* $\mathcal{S}_D$*, to be the finite semantic substitution*

$$\mathcal{S}_D = (\{\}, \{\rho \mapsto \rho \mid \rho \in Q\}, \{\epsilon \mapsto \epsilon.\underline{\varphi} \mid \epsilon.\underline{\varphi} \in \Phi\})$$

Intuitively, applying $\mathcal{S}_D$ to a syntactic object, e.g., a syntactic type, yields the semantic object which the syntactic object denotes when interpreted relative to D. Note that (1) $\mathrm{Dom}(\mathcal{S}_D) = \mathrm{Dom}(D)$ and (2) $\mathcal{S}_D(D) = D$ (using that D is consistent, in particular transitive) and thus, by (1) and (2), that $\mathcal{S}_D$ is a solution substitution. Further note that for all $\epsilon \in \mathrm{Dom}(D)$, $\mathcal{S}_D(\mathcal{S}_D(\epsilon)) = \mathcal{S}_D(\epsilon)$.

Definition 48 (Constraint System) *A* constraint system *is a pair* $((\mathcal{C}, I, k), D)$ *of a well-tagged constraint set and a consistent basis* D *satisfying*

(1) $\mathrm{frev}(\mathcal{C}) \subseteq \mathrm{Dom}(D)$*; and*
(2) for all $i \in \{1, \ldots, k\}$*, for all* $\alpha \stackrel{.}{\succeq}^i \tau$*,* $\rho \stackrel{.}{\succeq}^i \rho'$*,* $\epsilon \stackrel{.}{\succeq}^i \epsilon' \in \mathcal{C}$ *it holds that* $\alpha, \rho, \epsilon \in \mathrm{fv}(\mathcal{S}_D(\tau_i))$*, where* $(\tau_i, \_, \_, \_, \_) = I(i)$.

Here item (1) expresses that every region and effect variable used in $\mathcal{C}$ must have an interpretation in D. Item (2) expresses that every variable occurring on the left-hand side of an instantiation constraint must occur in the body of the type scheme that is being instantiated. Such variables can be secondary in that type (note the application of $\mathcal{S}_D$).

3.2.5 Solution of Constraint

Let $\mathcal{S}$ be a solution substitution and let $\underline{S}_1, \ldots, \underline{S}_k$ be semantic substitutions. Below, we define what it means for $(S, (\underline{S}_1, \ldots, \underline{S}_k))$ to *solve* a single constraint. In Section 3.2.6 we extend the definition to constraint systems. The following definition makes use of a couple of operations, which are defined in the text following the definition itself.

Definition 49 *Let $((\mathcal{C}, I, k), D)$ be a constraint system and let $c \in \mathcal{C}$ be a constraint in $\mathcal{C}$. Further, let $\mathcal{S}$ be a solution substitution and let $\underline{S}_1, \ldots, \underline{S}_k$ be semantic substitutions. We say that $(\mathcal{S}, (\underline{S}_1, \ldots, \underline{S}_k))$* solves *$c$ iff*

(1) $c \equiv \epsilon_1 \doteq \epsilon_2$ and $\mathcal{S}(\epsilon_1) = \mathcal{S}(\epsilon_2)$; or $c \equiv \rho_1 \doteq \rho_2$ and $\mathcal{S}(\rho_1) = \mathcal{S}(\rho_2)$; or $c \equiv \tau_1 \doteq \tau_2$ and $\mathcal{S}(\tau_1) = \mathcal{S}(\tau_2)$;
(2) $c \equiv \rho \mathrel{\dot{\succeq}^i} \rho'$ and $\underline{S}_i(\mathcal{S}(\rho)) = \mathcal{S}(\rho')$;
(3) $c \equiv \alpha \mathrel{\dot{\succeq}^i} \tau$ and $\underline{S}_i(\alpha) = \mathcal{S}(\tau)$ and $\underline{S}_i \downarrow \mathrm{fv}(\mathcal{S}(\tau)) = \underline{Id}$;
(4) $c \equiv \epsilon \mathrel{\dot{\succeq}^i} \epsilon$ and $\underline{S}_i \downarrow \mathrm{fv}(\mathcal{S}(\epsilon)) = \underline{Id}$;
(5) $c \equiv \epsilon \mathrel{\dot{\succeq}^i} \epsilon'$ $(\epsilon \neq \epsilon')$ and, letting $\epsilon.\underline{\varphi} = \mathcal{S}(\epsilon)$, $(\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau') = I(i)$, $\vec{\rho} = \rho_1 \cdots \rho_n$, and $\vec{\epsilon} = \epsilon_1 \cdots \epsilon_m$, one has
   (a) $\underline{S}_i(\mathcal{S}(\epsilon)) = \mathcal{S}(\epsilon')$; and
   (b) $\forall \rho'' \in \underline{\varphi} \setminus \mathrm{frv}(\mathcal{S}(\vec{m})).\ index(\rho'', \mathcal{S}(\tau)) = j \implies \underline{S}_i(\rho'') = \mathcal{S}(\rho_j)$;
   (c) $\forall \epsilon'' \in \underline{\varphi} \setminus \mathrm{fev}(\mathcal{S}(\vec{m})).\ index(\epsilon'', \mathcal{S}(\tau)) = j \implies \underline{S}_i(\epsilon'') = \mathcal{S}(\epsilon_j)$;
(6) $c \equiv \epsilon \mathrel{\dot{\supseteq}} \vec{\varphi}$ and, letting $\epsilon'.\underline{\varphi} = \mathcal{S}(\epsilon)$, one has $\underline{\varphi} \supseteq \mathcal{E}_{\mathcal{S}}(\vec{\varphi})$;
(7) $c \equiv \mathrm{Norm}(\tau, \vec{m})$ and
   (a) $\forall \rho, \rho' \in \mathrm{frv}(\mathcal{S}(\tau)) \setminus \mathrm{frv}(\mathcal{S}(\vec{m})).\ index(\rho, \mathcal{S}(\tau)) = index(\rho', \mathcal{S}(\tau)) \implies \rho = \rho'$;
   (b) $\forall \epsilon, \epsilon' \in \mathrm{fev}(\mathcal{S}(\tau)) \setminus \mathrm{fev}(\mathcal{S}(\vec{m})).\ index(\epsilon, \mathcal{S}(\tau)) = index(\epsilon', \mathcal{S}(\tau)) \implies \epsilon = \epsilon'$;
(8) $c \equiv (\tau, \vec{m}) \mathrel{\dot{\succeq}^i} (\vec{\rho}, \vec{\epsilon}, \tau')$ and $(\mathcal{S}, (S_1, \ldots, S_k))$ solves
   (a) $m \mathrel{\dot{\succeq}^i} m$, for all $m \in \vec{m}$;
   (b) $\alpha_0 \mathrel{\dot{\succeq}^i} \tau_0$, for all $(\alpha_0, \tau_0) \in$ tyvar-pairs(τ, τ');
   (c) $\rho \mathrel{\dot{\succeq}^i} \rho'$, for all $(\rho, \rho') \in$ regvar-pairs(τ, τ');
   (d) $\epsilon \mathrel{\dot{\succeq}^i} \epsilon'$, for all $(\epsilon, \epsilon') \in$ effectvar-pairs(τ, τ').

We shall now explain this definition. Since $\mathcal{S}$ is a solution substitution, it is the identity on type variables. The purpose of $\mathcal{S}$ is to map region variables and effect variables to semantic objects (namely region variables and arrow effects, respectively) that satisfy c.

Substitution $\underline{S_i}$, where i is a colour, arises from a situation where a program variable, say f, has both a binding occurrence (with type scheme $\underline{\sigma}$, say) and a non-binding occurrence with colour i. Then $\underline{S_i}$ is the substitution one applies to the bound variables of $\underline{\sigma}$ in order to get the type of f at the program point coloured i. Note that $\underline{S_i}$ need not be the identity on type variables.

An equality constraint is solved if $\mathcal{S}$ maps the left-hand side and the right-hand side to the same semantic object (see item (1)).

Item (2) expresses that $\underline{S_i}$ has to map $\mathcal{S}(\rho)$ (intuitively, a bound type variable of a type scheme) to $\mathcal{S}(\rho')$ (intuitively, the corresponding actual region variable at the program point coloured i).

In item (3), the condition $\underline{S_i} \downarrow \mathrm{fv}(\mathcal{S}(\tau)) = \underline{\mathrm{Id}}$ is needed for Lemma 50 below (see the explanation just before Lemma 50 for the intuition). Similarly for the condition $\underline{S_i} \downarrow \mathrm{fv}(\mathcal{S}(\epsilon)) = \underline{\mathrm{Id}}$ in item (4).

In item (5), (5a) expresses that ϵ is instantiated to ϵ', in analogy with (2). Next, (5b) expresses that every generic ρ'' in the effect denoted by ϵ must be instantiated to precisely that region variable at program point coloured i which has the index of ρ''. This is only well-defined if $\vec{\rho}$ has at least j elements (and which it indeed has by the definition of instantiation map). item (5c) is the corresponding requirement for effect variables.

Item (7) demands that, in a solution, there must not be two distinct generic region variables that have the same index. This is only interesting for region variables that only occur in secondary positions of $\mathcal{S}(\tau)$, since the condition is automatically satisfied for region variables that have a primary occurrence in $\mathcal{S}(\tau)$.

Item (8) makes use of operations tyvar-pairs, regvar-pairs, and effectvar-pairs, which are defined as follows. Let τ and τ' be syntactic types for which there exists a syntactic type variable substitution S_t such that $\mathrm{ML}(S_t(\tau)) = \mathrm{ML}(\tau')$. Then we define tyvar-pairs(τ, τ'), regvar-pairs(τ, τ'), and effectvar-pairs(τ, τ')

as follows.

$$\begin{aligned}
\text{tyvar-pairs}(\alpha, \tau') &= (\alpha, \tau') \\
\text{tyvar-pairs}((\tau_1, \rho_1) \xrightarrow{\epsilon} (\tau_2, \rho_2), & \\
(\tau_1', \rho_1') \xrightarrow{\epsilon'} (\tau_2', \rho_2')) &= \text{tyvar-pairs}(\tau_1, \tau_1') \\
&\ \text{tyvar-pairs}(\tau_2, \tau_2')
\end{aligned}$$

$$\begin{aligned}
\text{regvar-pairs}(\alpha, \tau') &= \emptyset \\
\text{regvar-pairs}((\tau_1, \rho_1) \xrightarrow{\epsilon} (\tau_2, \rho_2), & \\
(\tau_1', \rho_1') \xrightarrow{\epsilon'} (\tau_2', \rho_2')) &= (\rho_1, \rho_1')(\rho_2, \rho_2')\,\text{regvar-pairs}(\tau_1, \tau_1') \\
&\ \text{regvar-pairs}(\tau_2, \tau_2')
\end{aligned}$$

$$\begin{aligned}
\text{effectvar-pairs}(\alpha, \tau') &= \emptyset \\
\text{effectvar-pairs}((\tau_1, \rho_1) \xrightarrow{\epsilon} (\tau_2, \rho_2), & \\
(\tau_1', \rho_1') \xrightarrow{\epsilon'} (\tau_2', \rho_2')) &= (\epsilon, \epsilon')\,\text{effectvar-pairs}(\tau_1, \tau_1') \\
&\ \text{effectvar-pairs}(\tau_2, \tau_2')
\end{aligned}$$

(Note that tyvar-pairs(τ, τ'), regvar-pairs(τ, τ'), and effectvar-pairs(τ, τ') are all well-defined in Item (8) by the definition of constraint, which ensures that the required S_t exists.) Thus item (8) expresses that the semantic type scheme represented by $(\tau, \vec{m})$ via $\mathcal{S}$ must be instantiated to the type $\mathcal{S}(\tau')$. The requirement that $(\mathcal{S}, (\underline{S}_1, \dots, \underline{S}_k))$ solves all constraints $m \overset{\cdot}{\succeq}^i m$ (for all $m \in \vec{m}$) expresses that all the variables in $\mathcal{S}(\vec{m})$ must be non-generic, see the explanation and the lemma below.

Item (6) states that the effect ($\underline{\varphi}$) which ϵ denotes in the solution must be a superset of the result $\mathcal{E}_\mathcal{S}(\vec{\varphi})$ of evaluating the syntactic effect $\vec{\varphi}$ with respect to $\mathcal{S}$. We call the latter the semantics of $\vec{\varphi}$ with respect to $\mathcal{S}$; it is defined as follows. Let $\vec{\varphi}$ be a sequence of syntactic effects and let $\mathcal{S}$ be a solution substitution. If for all $\text{Obs}(\Gamma, (\tau, \rho), \epsilon') \in \vec{\varphi}$, Γ is in normal form with respect to $\mathcal{S}$, then the semantics of a $\vec{\varphi}$ with respect to $\mathcal{S}$ is $\mathcal{E}_\mathcal{S}(\vec{\varphi})$, where

$$\begin{aligned}
\mathcal{E}_\mathcal{S}(\vec{\varphi}) = &\{\mathcal{S}(\rho) \mid \rho \in \vec{\varphi}\} \\
&\cup \bigcup\{\mathcal{O}_\mathcal{S}(\Gamma, (\tau, \rho), \epsilon') \mid \text{Obs}(\Gamma, (\tau, \rho), \epsilon') \in \vec{\varphi}\} \\
&\cup \bigcup\{\underline{\varphi_1} \mid \epsilon_1.\underline{\varphi_1} = \mathcal{S}(\epsilon) \land \epsilon \in \vec{\varphi}\} \\
&\cup \bigcup\{\underline{\varphi_1} \cup \{\epsilon_1\} \mid \epsilon_1.\underline{\varphi_1} = \mathcal{S}(\epsilon) \land \overline{\epsilon} \in \vec{\varphi}\}
\end{aligned}$$

(Note that $\mathcal{E}_{\mathcal{S}}(\vec{\underline{\varphi}})$ is indeed well-defined in item (6) by the fact that $(\mathcal{C}, I, k)$ is a well-tagged set of constraints; see item (2b) in the definition of well-tagged set of constraints.)

The following lemma expresses the main purpose of having mono objects: an instantiation constraint of the form $m \dot{\succeq}^i m$ (i.e., with the same mono-object on the left and the right-hand side) forces all variables in $\mathcal{S}(m)$ to be non-generic at the program point coloured i:

Lemma 50 *$(\mathcal{S}, (\underline{S}_1, \ldots, \underline{S}_k))$ solves $m \dot{\succeq}^i m$ iff $\underline{S}_i \downarrow \mathrm{fv}(\mathcal{S}(m)) = \underline{Id}$.*

PROOF. Follows from items (2), (3), and (4) in Definition 49. □

3.2.6 Solution of Constraint Systems

We now define what it means for a semantic substitution to be a solution to a constraint system.

Definition 51 *A* solution *to a constraint system $((\mathcal{C}, I, k), D)$ is a semantic substitution $\mathcal{S}$ with* $\mathrm{Dom}(\mathcal{S}) \subseteq \mathrm{Dom}(D)$ *for which there exists a consistent basis A and k semantic substitutions $\underline{S}_1$, … $\underline{S}_k$ such that*

(1) $\mathcal{S}(D) \subseteq A$;
(2) $\forall \epsilon \in \mathrm{Dom}(D).\mathcal{S}(\mathcal{S}_D(\epsilon)) = \mathcal{S}(\epsilon)$;
(3) $\forall i \in \{1, \ldots, k\}.\, \mathrm{Dom}(\underline{S}_i) \subseteq \mathrm{fv}(\mathcal{S}(\tau_i)) \setminus \mathrm{fv}(\mathcal{S}(\vec{m}_i))$, *where* $(\tau_i, \vec{m}_i, \_, \_, \_) = I(i)$;
(4) $\forall c \in C.(\mathcal{S}, (\underline{S}_1, \ldots, \underline{S}_k))$ *solves c.*

Note that by item (1), a solution to a constraint system is *a forteriori* a solution substitution (and hence our choice of notation, $\mathcal{S}$, for the solution is consistent with the choice of using $\mathcal{S}$ to range over solution substitutions). See the remark following the definition of solution substitution (Definition 31 on Page 32) for further explanation.

To understand item (2), recall that $\mathcal{S}_D(\epsilon) = \epsilon.\underline{\varphi}$ is to be interpreted as saying that in the constraint system $((C, I, k), D)$, the interpretation of the effect variable ϵ prior to solving the constraints is the set $\underline{\varphi}$. Let $\epsilon'.\underline{\varphi'} = \mathcal{S}(\epsilon)$. We have $\mathcal{S}(\mathcal{S}_D(\epsilon)) = \mathcal{S}(\epsilon.\underline{\varphi}) = \epsilon'.(\underline{\varphi'} \cup \mathcal{S}(\underline{\varphi}))$. Thus item (2) amounts to demanding $\underline{\varphi'} \supseteq \mathcal{S}(\underline{\varphi})$, i.e., that the solution $\mathcal{S}$ should associate with ϵ an effect which is at least as big as the result of applying the solution substitution to $\underline{\varphi}$, the initial interpretation of ϵ.

Finally, item (3) expresses that the domain of $\underline{S}_i$ should be at most the generic variables of the type scheme that is being instantiated at the program point coloured i, while item (4) requires that $(\mathcal{S}, (\underline{S}_1, \ldots, \underline{S}_k))$ solve all the constraints in the constraint system.

3.2.7 Pre-well-tagged Sets of Constraints

The constraint-based region inference algorithm does not generate a constraint system but just a so-called pre-well-tagged set of constraints. From a pre-well-tagged set of constraints there is a mechanical way of obtaining a constraint system and a solution to a pre-well-tagged set of constraints is then defined to be a solution to that mechanically obtained constraint system. Formally, we define:

Definition 52 *A* pre-well-tagged set of constraints *is a finite multiset $\mathcal{C}$ of constraints satisfying that*

(1) all instantiation constraints in $\mathcal{C}$ are of the form $(\tau, \vec{m}) \dot{\succeq} (\vec{\rho}, \vec{\epsilon}, \tau')$;
(2) for all $(\tau, \vec{m}) \dot{\succeq} (\vec{\rho}, \vec{\epsilon}, \tau') \in \mathcal{C}$, one has $\mathrm{Norm}(\tau, \vec{m}) \in \mathcal{C}$; and
(3) for all $\epsilon \dot{\supseteq} \vec{\varphi} \in \mathcal{C}$, for all $\mathrm{Obs}(\Gamma, (\tau, \rho), \epsilon) \in \vec{\varphi}$, for all $((\tau', \vec{m}'), \rho') \in \mathrm{Rng}(\Gamma)$ one has $\mathrm{Norm}(\tau', \vec{m}') \in \mathcal{C}$.

The set of all pre-well-tagged sets of constraints is denoted Pre-Well-Tagged.

Definition 53 *A* solution *to a pre-well-tagged set of constraints $\mathcal{C}$ is defined as follows. Let k be the number of instantiation constraints in $\mathcal{C}$. Decorate all the instantiation constraints with distinct integers $i \in \{1, \ldots, k\}$ and let I be the finite map which for $i \in \{1, \ldots, k\}$ maps i to $(\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau')$ if and only if the i^{th} decorated instantiation constraint is $(\tau, \vec{m}) \dot{\succeq}^i (\vec{\rho}, \vec{\epsilon}, \tau')$. Call the resulting multiset of constraits $\mathcal{C}'$. Then clearly $(\mathcal{C}', I, k)$ is a well-tagged set of constraints. Let D be the simple basis with $\mathrm{Dom}(D) = \mathrm{frev}(\mathcal{C}')$. Then $((\mathcal{C}', I, k), D)$ is a constraint system. By a* solution *to $\mathcal{C}$ we understand a solution to $((\mathcal{C}', I, k), D)$.*

Strictly speaking, this definition only makes sense if one imposes some ordering on the instantiation constraints but the ordering is immaterial.

If $\mathcal{S}$ is a solution to a pre-well-tagged set of constraints $\mathcal{C}$ we also say that $\mathcal{S}$ *solves* $\mathcal{C}$. It is immediate to see that if $\mathcal{S}$ solves pre-well-tagged set of constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ then $\mathcal{C}_1 \cup \mathcal{C}_2$ is pre-well-tagged and $\mathcal{S}$ solves $\mathcal{C}_1 \cup \mathcal{C}_2$. Moreover, if $\mathcal{S}$ solves $\mathcal{C}$, then $\mathcal{S}$ solves $\mathcal{C}'$ for any pre-well-tagged set of constraints $\mathcal{C}' \subseteq \mathcal{C}$.

3.3 Constraint-based Formulation of Region Inference

In this section we present the constraint-based formulation of region inference. To this end we need a couple of definitions.

We assume two total functions

$$\text{freshRegVar} : \text{Basis} \to \text{RegVar} \times \text{Basis}$$

$$\text{freshEffVar} : \text{Basis} \to \text{EffVar} \times \text{Basis}$$

where $\text{freshRegVar}(D)$ returns a pair (ρ, D_1) satisfying $D_1 \setminus D = \{\rho\}$, and $\text{freshEffVar}(D)$ returns a pair (ϵ, D_1) satisfying $D_1 \setminus D = \{\epsilon\}$.

Using these two functions one can define a total function

$$\text{spread} : \text{MLType} \times \text{Basis} \to (\text{Type} \times \text{RegVar} \times \text{Basis})$$

satisfying that, when $\text{spread}(\overline{\tau}, D) = (\tau, \rho, D_1)$, then no region or effect variable occurs twice in (τ, ρ) and $D_1 = D \cup \text{frv}(\tau, \rho) \cup \text{fev}(\tau, \rho)$.

We write $(\rho_1 \cdots \rho_k, D_1) = \text{freshRegVar}(D)$ for the obvious k repeated applications of freshRegVar. Likewise for freshEffVar. Further, we write $(\tau, \rho) \doteq (\tau', \rho')$ as a shorthand for two constraints $\tau \doteq \tau'$ and $\rho \doteq \rho'$.

The constraint-based region inference rules allow the inference of sentences of the form

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} e \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$$

read: *in* $\mathcal{C}_1$, D_1 *and* Γ, e *translates to* e', *which has syntactic type* τ, *place* ρ *and effect* ϵ, *provided* $\mathcal{S}$ *is a solution to all the constraints in* $\mathcal{C}_2$ *and* $\text{frev}(\mathcal{C}_2) \subseteq \text{Dom}(D_2)$. Formally, the relation $\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} e \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$ is defined by rule induction on the set Pre-Well-Tagged × Basis × SynTypeEnv × SolutionSubst × SExp × TExp' × Type × RegVar × EffVar × Basis × Pre-Well-Tagged by the rules in Figures 7–9. (In rule (16) we make use of an operation *actuals* which we define in the explanation that follows the rules.) For future reference, we have numbered the constraints in the rules in Figures 7– 9 by $\mathbf{c_1}$, $\mathbf{c_2}$, etc.

We now explain the rules informally, hoping that this will convince the reader that the rules are not as complicated as they look! Thereafter, we present a proof of the soundness of the constraint-based region inference rules with respect to the semantic region inference rules.

All uses of spread and freshEffVar just express that the region and effect variables in the constraints are “fresh” and that all the syntactic types have as underlying ML types the types of the source expression.

$$\frac{\begin{array}{c}\Gamma(f) = ((\tau', \vec{m}), \rho') \qquad (\tau, \rho, D_2) = \mathrm{spread}(\overline{\tau}, D_1) \\ n = \text{number of arrows in } \overline{\tau}' \qquad \exists S_t.\,\mathrm{ML}(S_t(\tau')) = \overline{\tau} \\ (\vec{\rho}\,' = \rho'_1, \ldots, \rho'_{2^n}, D_3) = \mathrm{freshRegVar}(D_2) \\ (\vec{\epsilon}\,' = \epsilon'_1, \ldots, \epsilon'_{2^n}, D_4) = \mathrm{freshEffVar}(D_3) \\ (\epsilon, D_5) = \mathrm{freshEffVar}(D_4) \\ \begin{array}{lll} \mathcal{S} \text{ solves } \mathcal{C}_2 = \{ & (\tau', \vec{m}) \,\dot{\succeq}\, (\vec{\rho}\,' \,\overline{\cup}\, \mathrm{frv}(\tau), \vec{\epsilon}\,' \,\overline{\cup}\, \mathrm{fev}(\tau), \tau)\} & \mathbf{c_1} \\ & \epsilon \,\dot{\supseteq}\, \rho'\rho & \mathbf{c_2} \\ \} \cup \mathcal{C}_1 & & \end{array} \\ \rho''_1, \ldots, \rho''_k = \mathit{actuals}_{\mathcal{S}}(\tau', \tau, \vec{m}) \setminus (\mathcal{S}(\rho'_1), \ldots, \mathcal{S}(\rho'_{2^n}))\end{array}}{\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} f : \overline{\tau} \Rightarrow f\,[\mathcal{S}(\rho'_1), \ldots, \mathcal{S}(\rho'_{2^n}), \rho''_1, \ldots, \rho''_k] \;\texttt{at}\; \mathcal{S}(\rho) : (\tau, \rho), \epsilon, D_5, \mathcal{C}_2} \tag{16}$$

$$\frac{\Gamma(x) = (\tau, \rho) \qquad (\epsilon, D_2) = \mathrm{freshEffVar}(D_1) \qquad \mathrm{ML}(\tau) = \overline{\tau}}{\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} x : \overline{\tau} \Rightarrow x : (\tau, \rho), \epsilon, D_2, \mathcal{C}_1} \tag{17}$$

$$\frac{\begin{array}{c}(\tau_1, \rho_1, D_2) = \mathrm{spread}(\overline{\tau_1}, D_1) \\ \mathcal{C}_1, D_2, \Gamma + \{x \mapsto (\tau_1, \rho_1)\} \Vdash_{\mathcal{S}} e \Rightarrow e' : (\tau_2, \rho_2), \epsilon_2, D_3, \mathcal{C}_2 \\ (\rho, D_4) = \mathrm{freshRegVar}(D_3) \\ (\epsilon, \epsilon_3, D_5) = \mathrm{freshEffVar}(D_4) \\ \begin{array}{lll} \mathcal{S} \text{ solves } \mathcal{C}_3 = \{ & \epsilon_3 \,\dot{\supseteq}\, \epsilon_2 & \mathbf{c_1} \\ & \epsilon \,\dot{\supseteq}\, \rho & \mathbf{c_2} \\ & \} \cup \mathcal{C}_2 & \end{array}\end{array}}{\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} \lambda x : \overline{\tau_1}.e : \overline{\tau} \Rightarrow \lambda x.e' \;\texttt{at}\; \mathcal{S}(\rho) : ((\tau_1, \rho_1) \xrightarrow{\epsilon_3} (\tau_2, \rho_2), \rho), \epsilon, D_5, \mathcal{C}_3} \tag{18}$$

Fig. 7. Constraint-based Region Inference Rules, Part 1

The syntactic type environment Γ records for each bound variable in scope its syntactic type or type scheme at the binding occurrence, as explained in the definition of $\Gamma \simeq_{\mathcal{S}} TE$.

The pre-well-tagged sets of constraints $\mathcal{C}_i$ collect all the constraints generated for an expression. In fact the following two lemmas are easily shown by rule induction (the second uses the first).

Lemma 54 *If* $\forall((\tau, \vec{m}), \rho) \in \mathrm{Rng}(\Gamma).\exists\, \mathrm{Norm}(\tau, \vec{m}) \in \mathcal{C}_1$, $\mathcal{C}_1$ *is pre-well-tagged,*

$$
\frac{\begin{array}{c}
(\tau_0, \rho_0, D_2) = \mathrm{spread}(\overline{\tau_0}, D_1) \\
\vec{m_3} = \mathrm{mono}(\Gamma) \\
\vec{\alpha_1} = \mathrm{ftv}(\tau_0) \\
(\epsilon_{11}, \epsilon_3, D_3) = \mathrm{freshEffVar}(D_2) \\
\vec{m_1} = \vec{\alpha_1}\, \rho_0\, \epsilon_{11}\epsilon_3 \\
\mathcal{C}_2 = \mathcal{C}_1 \cup \{\mathrm{Norm}(\tau_0, \vec{m_1}\, \vec{m_3}\,)\} \\
\mathcal{C}_2, D_3, \Gamma + \{f \mapsto ((\tau_0, \vec{m_1}\, \vec{m_3}\,), \rho_0)\} \Vdash_{\mathcal{S}} \lambda x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow \\
\qquad\qquad \lambda x.e_1' \;\texttt{at}\; \mathcal{S}(\rho_1) : (\tau_1, \rho_1), \epsilon_1, D_4, \mathcal{C}_3 \\
\vec{\alpha_2} = \mathrm{ftv}(\tau_0) \setminus \vec{\alpha} \\
\vec{m_2} = \vec{\alpha_2}\, \rho_0\, \epsilon_{11}\epsilon_3 \\
\mathcal{C}_4 = \mathcal{C}_3 \cup \{\mathrm{Norm}(\tau_0, \vec{m_2}\, \vec{m_3}\,)\} \\
\mathcal{C}_4, D_4, \Gamma + \{f \mapsto ((\tau_0, \vec{m_2}\, \vec{m_3}\,), \rho_0)\} \Vdash_{\mathcal{S}} e_2 \Rightarrow e_2' : (\tau_2, \rho_2), \epsilon_2, D_5, \mathcal{C}_5 \\
(\epsilon_4, \epsilon, D_6) = \mathrm{freshEffVar}(D_5) \\
\mathcal{S} \text{ solves } \mathcal{C}_6 = \{ \quad \begin{array}{rcll}
(\tau_0, \rho_0) & \doteq & (\tau_1, \rho_1) & \mathbf{c_1} \\
\epsilon_4 & \dot{\supseteq} & \epsilon_1\epsilon_2 & \mathbf{c_2} \\
\epsilon & \dot{\supseteq} & \mathrm{Obs}(\Gamma, (\tau_2, \rho_2), \epsilon_4) & \mathbf{c_3} \\
\epsilon_{11} & \doteq & \epsilon_1 & \mathbf{c_4}
\end{array} \\
\} \cup \mathcal{C}_5 \\
\mathcal{S}(\epsilon_4) = \underline{\epsilon_4}.\underline{\varphi_4} \qquad \mathcal{S}(\epsilon) = \underline{\epsilon}.\underline{\varphi} \\
\vec{\rho}\,' = \mathrm{frv}(\underline{\varphi_4} \setminus \underline{\varphi}) \\
\underline{\sigma} = \forall \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon}.\mathcal{S}(\tau_0) = \mathrm{mk}\_\underline{\sigma}(\mathcal{S}, (\tau_0, \vec{m_2}\, \vec{m_3}\,)) \\
\vec{\alpha} \cap \mathrm{ftv}(\vec{m_3}\,) = \emptyset
\end{array}}{\begin{array}{l}
\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} \texttt{letrec}\; f : \forall\vec{\alpha}.\overline{\tau_0}\;\texttt{(}x : \overline{\tau_1}\texttt{)}\;\texttt{=}\; e_1 \;\texttt{in}\; e_2 : \overline{\tau} \Rightarrow \\
\qquad\qquad \texttt{letregion}\; \vec{\rho}\,' \;\texttt{in} \\
\qquad\qquad\quad \texttt{letrec}\; f\texttt{[}\vec{\rho}\texttt{](}x\texttt{)}\;\texttt{at}\; \mathcal{S}(\rho_0) \;\texttt{=}\; e_1' \;\texttt{in}\; e_2' : (\tau_2, \rho_2), \epsilon, D_6, \mathcal{C}_6
\end{array}} \quad (19)
$$

Fig. 8. Constraint-based Region Inference Rules, Part 2

$\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_1)$, $\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_1)$, *$\mathcal{S}$ solves $\mathcal{C}_1$, D_1 is simple, and* $\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$*, then $\mathcal{C}_2 \supseteq \mathcal{C}_1$, $\mathcal{C}_2$ is pre-well-tagged, $D_2 \supseteq D_1$, D_2 is simple,* $\mathrm{frev}(\mathcal{C}_2) \subseteq \mathrm{Dom}(D_2)$, *and $\mathcal{S}$ solves $\mathcal{C}_2$.*

$$\frac{\begin{array}{c}((\tau_{11},\rho_{11}) \xrightarrow{\epsilon_3} (\tau_{12},\rho_{12}), \_, D_2) = \mathrm{spread}(\mathrm{MLTypeOf}(e_1), D_1) \\ \mathcal{C}_1, D_2, \Gamma \Vdash_{\mathcal{S}} e_1 \Rightarrow e_1' : (\tau_1,\rho_1), \epsilon_1, D_3, \mathcal{C}_2 \\ \mathcal{C}_2, D_3, \Gamma \Vdash_{\mathcal{S}} e_2 \Rightarrow e_2' : (\tau_2,\rho_2), \epsilon_2, D_4, \mathcal{C}_3 \\ (\epsilon_4, \epsilon, D_5) = \mathrm{freshEffVar}(D_4) \\ \mathcal{S} \text{ solves } \mathcal{C}_4 = \{ \begin{array}{rcll} \tau_1 & \doteq & (\tau_{11},\rho_{11}) \xrightarrow{\epsilon_3} (\tau_{12},\rho_{12}) & \mathbf{c_1} \\ (\tau_2,\rho_2) & \doteq & (\tau_{11},\rho_{11}) & \mathbf{c_2} \\ \epsilon_4 & \dot{\supseteq} & \epsilon_1\epsilon_2\overline{\epsilon_3}\rho_1 & \mathbf{c_3} \\ \epsilon & \dot{\supseteq} & \mathrm{Obs}(\Gamma, (\tau_{12},\rho_{12}), \epsilon_4) & \mathbf{c_4} \end{array} \\ \} \cup \mathcal{C}_3 \\ \mathcal{S}(\epsilon_4) = \underline{\epsilon_4}.\underline{\varphi_4} \qquad \mathcal{S}(\epsilon) = \underline{\epsilon}.\underline{\varphi} \\ \vec{\rho} = \mathrm{frv}(\underline{\varphi_4} \setminus \underline{\varphi}) \end{array}}{\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} e_1 e_2 : \overline{\tau} \Rightarrow \texttt{letregion } \vec{\rho} \texttt{ in } e_1' e_2' : (\tau_{12},\rho_{12}), \epsilon, D_5, \mathcal{C}_4} \quad (20)$$

Fig. 9. Constraint-based Region Inference Rules, Part 3

Lemma 55 *If* $\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_1} e \Rightarrow e' : (\tau,\rho), \epsilon, D_2, \mathcal{C}_2$ *and* $\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_1)$ *and* $\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_2)$ *and* $\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_1)$ *and* $\mathrm{Dom}(\mathcal{S}_2) \cap \mathit{TyVar} = \emptyset$ *and* $\mathrm{Dom}(\mathcal{S}_2) \cap \mathrm{Dom}(D_2) = \emptyset$*, then* $\mathcal{C}_1, D_1 \Vdash_{\mathcal{S}_1+\mathcal{S}_2} e \Rightarrow e' : (\tau,\rho), \epsilon, D_2, \mathcal{C}_2$.

Generally, the equality constraints express that certain types must be equal. In the semantic region inference rules this is expressed by repeated use of the same meta-variables. There are two kinds of set-constraints: those involving Obs(...) terms, and the rest. The rest simply express how effects are determined by effects of subphrases. In the semantic region inference rules this is expressed using union of effects. There is a set-constraint involving an Obs(...) term for every expression, where it may be possible to observe away part of the effect. (Note that in the semantic region inference rules, rule (7) can never observe away part of the effect given for variables, rules (2) and (3), or lambda abstractions, rule (4).) Finally, the instantiation constraint in the rule for `letrec`-bound variables, rule (16), expresses that the type for an occurrence of a variable should be an instance of the type at the binding occurrence of the variable, as explained in Section 3.1.

The use of the solution substitution $\mathcal{S}$ expresses how a target expression is obtained from a solution to the constraints. Thus the constraint-based region inference rules both formalize what constraints to extract from a given source expression and how to use a solution to the extracted constraints to obtain a target expression — it is necessary to formalize both simultaneously in order to be able to argue, as we will do in the following, that a solution to the extracted constraints indeed can be used to obtain a target expression as defined by the semantic region inference rules.

With regard to rule (16), let us first define the operation *actuals*.

Definition 56 *Let τ' and τ be two syntactic types satisfying $\exists S_t : \mathrm{ML}(S_t(\tau')) = \mathrm{ML}(\tau)$. Let $\mathcal{S}$ be a solution substitution and let $\vec{m}$ be a mono object sequence. Further, let $\vec{\rho} = \mathrm{fprv}(\mathcal{S}(\tau')) \setminus \mathrm{frv}_{\mathcal{S}}(\mathcal{S}(\vec{m}))$. Then $actuals_{\mathcal{S}}(\tau', \tau, \vec{m})$ is defined to be the sequence $a(\mathcal{S}(\tau'), \mathcal{S}(\tau))$ of region variables so defined:*

$$
\begin{aligned}
a(\alpha, \_) &= \emptyset \\
a((\underline{\tau_1'}, \rho_1') \xrightarrow{\epsilon'.\varphi'} (\underline{\tau_2'}, \rho_2'), & \\
(\underline{\tau_1}, \rho_1) \xrightarrow{\epsilon.\varphi} (\underline{\tau_2}, \rho_2)) &= a(\underline{\tau_1'}, \underline{\tau_1}) \\
&\quad \overline{\cup}\ a(\underline{\tau_2'}, \underline{\tau_2}) \\
&\quad \overline{\cup}\ \mathit{if}\ \rho_1' \in \vec{\rho}\ \mathit{then}\ \rho_1\ \mathit{else}\ \emptyset \\
&\quad \overline{\cup}\ \mathit{if}\ \rho_2' \in \vec{\rho}\ \mathit{then}\ \rho_2\ \mathit{else}\ \emptyset
\end{aligned}
$$

Recalling the explanation in the preceding section of how semantic type schemes and semantic types are represented, one can see that $\mathcal{S}(\rho_1'), \ldots, \mathcal{S}(\rho_{2^n}')$ correspond to the instantiations of the secondary bound variables and $\rho_1'', \ldots, \rho_k''$ correspond to the instantiations of the bound primary variables so that

$$\mathcal{S}(\rho_1'), \ldots, \mathcal{S}(\rho_{2^n}')\rho_1'', \ldots, \rho_k''$$

correspond to $\underline{S}(\rho_1) \cdots \underline{S}(\rho_k)$ in the semantic region inference rule (3). Note that because the a operation used in the definition of $actuals_{\mathcal{S}}(\tau', \tau, \vec{m})$ collects region variables *in the same order* as the fprv operation (see the definition in Appendix A) used in the definition of the index of a region variable (see definition on Page 23), the variables $\mathcal{S}(\rho_1'), \ldots, \mathcal{S}(\rho_{2^n}')\rho_1'', \ldots, \rho_k''$ occur in the same order as the variables $\underline{S}(\rho_1) \cdots \underline{S}(\rho_k)$ in rule (3). This is required for rule (16) to implement rule (3) correctly since semantic type schemes are not equal under reordering of bound variables. (This is the reason why our free variable operations return sequences instead of sets.) Finally note that $\mathbf{c_2}$ expresses the effect requirement in rule (3).

In rule (20), recalling the definition of $\mathcal{E}_{\mathcal{S}}(\vec{\overline{\varphi}})$ one can see that $\mathbf{c_3}$ expresses the requirement on the effect in rule (5), in particular that the effect variable on the arrow effect in the function type should be included in the effect (expressed in $\mathbf{c_3}$ by the overlining of ϵ_3). Further, $\mathbf{c_4}$ means that ϵ corresponds to the effect after an application of the semantic rule (7).

In rule (19), $\vec{m_1}$ includes all the type variables of τ_0 because there is no type quantification in rule (6) in the type scheme bound to f in the type environment used for e_1. The variable ρ_0 is included in $\vec{m}$ because the corresponding

ρ_0 in rule (6) is not quantified. The variable ϵ_{11} is included in $\vec{m}_1$ because of the requirement in rule (6) that the quantified variables are disjoint from $\text{fv}(\underline{\varphi_1})$ (using that the constraint $\mathbf{c_4}$ expresses that ϵ_{11} corresponds to the effect $\underline{\varphi_1}$). Finally, ϵ_3 is included for technical reasons: it is used as a "slack variable" in the completeness proof below; by letting $\mathcal{S}$ map ϵ_3 to a suitable arrow effect, namely $\epsilon_3.\underline{\varphi_3}$ in the implementation-oriented rule (13), the constraint-based rules can also express the situation where one has less quantification than possible in the corresponding semantic region inference rule (13). See the proof for the details. The sequence $\vec{m}_2$ is like $\vec{m}_1$ except that not all type variables are included; only the type variables that are not quantified in the corresponding ML type are included. This expresses that type variables can be quantified in the type scheme bound to f in the type environment used for e_2 in rule (6), and that variables that are not quantified in the corresponding ML type scheme should not be quantified.

This concludes our description of the constraint-based region inference rules. Formally, we have the following soundness result of the constraint-based region inference rules indeed with respect to the semantic region inference rules.

Theorem 57 (Soundness) *Let $\mathcal{S}$ be a solution substitution. If $\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} e \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$, and $\Gamma \simeq_{\mathcal{S}} TE$ then there exist $\underline{\epsilon}$ and $\underline{\varphi}$ such that TE is well-formed and $TE \vdash e \Rightarrow e' : (\mathcal{S}(\tau), \mathcal{S}(\rho)), \underline{\varphi}$ and $\mathcal{S}(\epsilon) = \underline{\epsilon}.\underline{\varphi}$.*

PROOF. See Appendix E. □

Next, we have a completeness theorem, expressing the completeness of the constraint-based region inference rules with respect to the implementation-oriented region inference rules. We discuss the formulation of the theorem following the statement of it.

Theorem 58 (Completeness) *If*

(1) $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$ *with rule* (14) *only applied immediately after rule* (13) *or* (12)*;*
(2) $\Gamma \simeq_{\mathcal{S}_1} TE$*;*
(3) $\text{frev}(\Gamma) \subseteq \text{Dom}(D_1)$*;*
(4) $\text{Dom}(\mathcal{S}_1) \subseteq \text{Dom}(D_1)$*;*
(5) D_1 *is a simple basis;*
(6) $\text{Dom}(D_1) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset$*;*
(7) $\binom{A'}{B} \uplus \binom{A}{B}$ *exists;*
(8) $\mathcal{S}_1(D_1) \subseteq A'$*;*
(9) $\mathcal{C}_1$ *is pre-well-tagged and* $\forall((\tau, \vec{m}), \rho) \in \text{Rng}(\Gamma). \exists\, \text{Norm}(\tau, \vec{m}) \in \mathcal{C}_1$*;*
(10) $\text{frev}(\mathcal{C}_1) \subseteq \text{Dom}(D_1)$*;*

(11) $\mathcal{S}_1$ solves $\mathcal{C}_1$

then there exist D_2, $\mathcal{S}_2$, $\mathcal{C}_2$ τ, ρ, *and* ϵ *such that*

(1) $\mathrm{Dom}(D_2) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset$*;*
(2) $\mathrm{Dom}(\mathcal{S}_2) \subseteq \mathrm{Dom}(D_2 \setminus D_1)$*;*
(3) $(\mathcal{S}_1 + \mathcal{S}_2)(D_2) \subseteq A' \cup A$*;*
(4) $\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_1+\mathcal{S}_2} e \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$*;*
(5) $(\mathcal{S}_1 + \mathcal{S}_2)(\tau) = \underline{\tau}$*;*
(6) $(\mathcal{S}_1 + \mathcal{S}_2)(\rho) = \underline{\rho}$*;*
(7) $(\mathcal{S}_1 + \mathcal{S}_2)(\epsilon) = \underline{\epsilon}.\underline{\varphi}$*.*

PROOF. See Appendix F. □

Let us first remark that the condition in assumption (1) that rule (14) is only applied immediately after rule (13) or (12) is not a real restriction: the observe rule (7) can only observe away a part of an effect if it is applied after one of the rules (13) or (12). Of course, the observe rule can also be used to observe away part of an effect just added using the rule (15), but first adding an effect and then discharging part of it can always be avoided. The constraint set $\mathcal{C}_1$ can be thought of as the set of constraints which $\mathcal{S}_1$ must solve in order for Γ to represent TE via $\mathcal{S}_1$. One should think of D_1 as consisting of all the syntactic variables possibly used by Γ and $\mathcal{C}_1$, hence items (2) to (5), and items (10) and (11). Item (9) just says that $\mathcal{C}_1$ is a sensible set of constraints for the syntactic type environment Γ (recall the discussion of pre-well-tagged sets of constraints in Section 3.2.7). The "parameter" cone $\binom{A'}{B}$ is used to express what the range of $\mathcal{S}_1$ is, c.f. (8). One would perhaps have expected $\mathcal{S}_1$ to map into A, but that does not work. To see why, consider the case where the derivation (1) has two subderivations; for concreteness, let us think of the case where e is an application expression $e_1\ e_2$. Then the basis A consists of two bases, A_1 and A_2, for the first and second subderivation, respectively. When we consider the first subderivation, we will, as expressed by the conclusion of the theorem, get a substition, call it $\mathcal{S}_{A_1}$, and a basis, call it D_{A_1}, satisfying certain conditions. The idea is that $D_{A_1} \setminus D_1$ consists of the syntactic variables needed to represent the first semantic subderivation, covered by A_1, and that $\mathcal{S}_{A_1}$ maps the syntactic variables in $D_{A_1} \setminus D_1$ into the semantic variables in A_1. Now when we consider the second subderivation, covered by the basis A_2, our current substitution is $\mathcal{S}_1 + \mathcal{S}_{A_1}$ — but the range of this substitution is not in A_2, but rather in $A \cup A_1$. This explains the use of A' in (7). (To see in more detail how this argument works, look at the proof for the case of application in Appendix F.) The "parameter" basis A'' intuitively simply consists of the variables which in the future will be used in a semantic derivation — the syntactic variables introduced to represent the derivation in (1) (i.e., $\mathrm{Dom}(D_2 \setminus D_1)$), should not overlap with A''. The remaining part of the conclusion of the theorem expresses that there exists a constraint-based region inference derivation corresponding to the semantic one given in (1).

4 Constraint Normalization

This section describes the process of normalizing a finite set of constraints. The purpose is to reduce the set of constraints to a form (called a normal form, defined in Subsection 4.1 below) from which it is trivial to extract a solution. For this to succeed, equality and instantiation constraints must be rewritten into a simplified form or eliminated by substitution. Furthermore, we need to associate semantic effects with effect variables to make sure that set constraints become satisfied. Hence we will in fact rewrite not simply a set of constraints but a constraint system.

Figure 10 outlines how constraint normalization works. The initial constraint system $((\mathcal{C}_0, I_0, k), D_0)$ (shown in the lower left corner in the figure) undergoes a finite number of rewriting steps, each spawning off a syntactic substitution (denoted S_i in the figure). The final constraint system $((\mathcal{C}_n, I_n, k), D_n)$ is in normal form and can trivially be solved; in fact, $\mathcal{S}_n = \mathcal{S}_{D_n}$ (the semantic substitution induced by D_n) is a solution to $((\mathcal{C}_n, I_n, k), D_n)$. Finally, the solution to $((\mathcal{C}_0, I_0, k), D_0)$ is obtained by composing with the substitutions from the rewriting steps. Rewriting is sound in the sense that this procedure does indeed give a solution to the initial constraint system, see Theorem 64 in Subsection 4.2. Rewriting is strongly normalizing, Theorem 66, and always results in a normal form, Theorem 65. Strong normalization means that, with regard to termination, rewriting steps can be applied in any order. However, the rewriting rules are mildly incomplete in the sense that not all solutions are preserved by the rewriting rules. This is discussed in Subsection 4.2.

We shall use the terms rewrite and normalize interchangeably.

4.1 Normal Form

Definition 59 *Let $((\mathcal{C}, I, k), D)$ be a constraint system. Then we say that $((\mathcal{C}, I, k), D)$ is in* normal form *if and only if the following conditions are all met:*

(1) There are no constraints of the form $\tau \doteq \tau'$ $(\tau \neq \tau')$, $\rho \doteq \rho'$ $(\rho \neq \rho')$, or $\epsilon \doteq \epsilon'$ $(\epsilon \neq \epsilon')$ in $\mathcal{C}$.
(2) There are no constraints of the form $(\tau, \vec{m}) \dot{\succeq}^i (\vec{\rho}, \vec{\epsilon}, \tau')$.

$$
\begin{tikzcd}
((\mathcal{C}_n, I_n, k), D_n) \arrow[rr, "\mathcal{S}_{D_n}"] & & \mathcal{S}_n = \mathcal{S}_{D_n} \arrow[dd] \\
{} \arrow[rr, "S_n"] & & {} \\
((\mathcal{C}_{n-1}, I_{n-1}, k), D_{n-1}) \arrow[uu, Rightarrow] & & \mathcal{S}_{n-1} = \mathcal{S}_n \circ S_n \arrow[dd] \\
{} \arrow[rr, "S_{n-1}"] & & {} \\
\vdots \arrow[uu, Rightarrow] & \vdots & \vdots \arrow[dd] \\
{} \arrow[rr, "S_2"] & & {} \\
((\mathcal{C}_1, I_1, k), D_1) \arrow[uu, Rightarrow] & & \mathcal{S}_1 = \mathcal{S}_2 \circ S_2 \arrow[dd] \\
{} \arrow[rr, "S_1"] & & {} \\
((\mathcal{C}_0, I_0, k), D_0) \arrow[uu, Rightarrow] & & \mathcal{S}_0 = \mathcal{S}_1 \circ S_1
\end{tikzcd}
$$

Fig. 10. Solving a Constraint System by Normalization

(3) There is no pair of constraints $\alpha \stackrel{\cdot}{\succeq}^i \tau_1$ *and* $\alpha \stackrel{\cdot}{\succeq}^i \tau_2$ *with* $\tau_1 \neq \tau_2$ *in* $\mathcal{C}$.
(4) There is no pair of constraints $\rho \stackrel{\cdot}{\succeq}^i \rho_1$ *and* $\rho \stackrel{\cdot}{\succeq}^i \rho_2$ *with* $\rho_1 \neq \rho_2$ *in* $\mathcal{C}$.
(5) There is no pair of constraints $\epsilon \stackrel{\cdot}{\succeq}^i \epsilon_1$ *and* $\epsilon \stackrel{\cdot}{\succeq}^i \epsilon_2$ *with* $\epsilon_1 \neq \epsilon_2$ *in* $\mathcal{C}$.
(6) For all $\epsilon \stackrel{\cdot}{\succeq}^i \epsilon \in \mathcal{C}$ *it holds that*
  (a) $\forall \rho' \in \mathrm{frv}(\mathcal{S}_D(\epsilon)) : \rho' \stackrel{\cdot}{\succeq}^i \rho' \in \mathcal{C}$
  (b) $\forall \epsilon' \in \mathrm{fev}(\mathcal{S}_D(\epsilon)) : \epsilon' \stackrel{\cdot}{\succeq}^i \epsilon' \in \mathcal{C}$
(7) For all constraints $\epsilon \mathrel{\dot{\supseteq}} \vec{\overline{\varphi}} \in \mathcal{C}$, *it holds that, if* $\underline{\varphi} = \widehat{\Phi \mathit{\ of} D}(\epsilon)$, *then* $\underline{\varphi} \supseteq \mathcal{E}_{\mathcal{S}_D}(\vec{\overline{\varphi}})$.
(8) For all $\epsilon \stackrel{\cdot}{\succeq}^i \epsilon'$ *in* $\mathcal{C}$ *with* $\epsilon \neq \epsilon'$ *it holds that, if* $\underline{\varphi} = \widehat{\Phi \mathit{\ of} D}(\epsilon)$ *and* $(\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau') = I(i)$, *then*
  (a) $\forall \rho'' \in \underline{\varphi} \setminus \mathrm{frv}(\mathcal{S}_D(\vec{m})).\ \mathit{index}(\rho'', \mathcal{S}_D(\tau)) = j \implies \rho'' \stackrel{\cdot}{\succeq}^i \rho_j \in \mathcal{C} \wedge \epsilon' \mathrel{\dot{\supseteq}} \rho_j \in \mathcal{C}$;
  (b) $\forall \rho'' \in \underline{\varphi} \cap \mathrm{frv}(\mathcal{S}_D(\vec{m})).\epsilon' \mathrel{\dot{\supseteq}} \rho'' \in \mathcal{C}$;
  (c) $\forall \epsilon'' \in \underline{\varphi} \setminus \mathrm{fev}(\mathcal{S}_D(\vec{m})).\ \mathit{index}(\epsilon'', \mathcal{S}_D(\tau)) = j \implies \epsilon'' \stackrel{\cdot}{\succeq}^i \epsilon_j \in \mathcal{C} \wedge \epsilon' \mathrel{\dot{\supseteq}} \overline{\epsilon_j} \in \mathcal{C}$;

(d) $\forall \epsilon'' \in \underline{\varphi} \cap \mathrm{fev}(\mathcal{S}_D(\vec{m})).\epsilon' \dot{\supseteq} \epsilon'' \in \mathcal{C}$;

(9) *For all* $\mathrm{Norm}(\tau\vec{m}) \in \mathcal{C}$ *it holds that*

(a) $\forall \rho, \rho' \in \mathrm{frv}(\mathcal{S}_D(\tau)) \setminus \mathrm{frv}(\mathcal{S}_D(\vec{m}))$.
$index(\rho, \mathcal{S}_D(\tau)) = index(\rho', \mathcal{S}_D(\tau)) \implies \rho \doteq \rho' \in \mathcal{C}$.

(b) $\forall \epsilon, \epsilon' \in \mathrm{frv}(\mathcal{S}_D(\tau)) \setminus \mathrm{frv}(\mathcal{S}_D(\vec{m}))$.
$index(\epsilon, \mathcal{S}_D(\tau)) = index(\epsilon', \mathcal{S}_D(\tau)) \implies \epsilon \doteq \epsilon' \in \mathcal{C}$.

(10) $\forall i \in \{1, \ldots, k\}.I(i) = (\_, \vec{m}, \_, \_, \_) \implies \forall m \in \vec{m}.m \dot{\succeq}^i m \in \mathcal{C}$.

Remark 60 *Note that in item* (9) *in the above definition, we have that, by item* (1), ρ *and* ρ' *will always be equal. Likewise for* ϵ *and* ϵ'.

Theorem 61 (Solution to Normal Form) *Let* $((\mathcal{C}, I, k), D)$ *be a constraint system. If* $((\mathcal{C}, I, k), D)$ *is in normal form, then* $\mathcal{S}_D$ *is a solution to* $\mathcal{C}$.

PROOF. First note that condition (2) in the definition of solution is clearly satisfied (by the remarks following the definition of $\mathcal{S}_D$, see Section 3.2.4).

Let A in the definition of solution be D; then condition (1) in the definition of solution is satisfied.

For all $i \in \{1, \ldots, k\}$ we define an $\underline{S}_i$ as follows. First define the following relations:

$$
\begin{aligned}
R_t^i &= \{(\alpha, \mathcal{S}_D(\tau)) \mid \alpha \dot{\succeq}^i \tau \in \mathcal{C} \wedge \alpha \neq \tau\} \\
R_r^i &= \{(\rho, \rho') \mid \rho \dot{\succeq}^i \rho' \in \mathcal{C} \wedge \rho \neq \rho'\} \\
R_e^i &= \{(\epsilon, \epsilon'.\underline{\varphi'}) \mid \epsilon \dot{\succeq}^i \epsilon' \in \mathcal{C} \wedge \epsilon \neq \epsilon' \wedge \mathcal{S}_D(\epsilon') = \epsilon'.\underline{\varphi'}\}
\end{aligned}
$$

Then note that these relations are all functional by items (3), (4) and (5) in the definition of normal form. Hence

$$\underline{S}_i = (R_t^i, R_r^i, R_e^i)$$

is indeed a well-defined semantic substitution.

Moreover, condition (3) in the definition of solution, i.e., the condition $\forall i \in \{1, \ldots, k\}. \mathrm{Dom}(\underline{S}_i) \subseteq \mathrm{fv}(\mathcal{S}(\tau_i)) \setminus \mathrm{fv}(\mathcal{S}(\vec{m}_i))$, where $(\tau_i, \vec{m}_i, \_, \_, \_) = I(i)$, is satisfied because

(1) by condition (2) of the definition of constraint system, we have that $\forall \vartheta \in \mathrm{Dom}(\underline{S}_i).\vartheta \in \mathrm{fv}(\mathcal{S}_D(\tau_i))$;
(2) by items (10) and (6) in the definition of normal form, we have that $\forall \vartheta \in \mathrm{Dom}(\underline{S}_i).\vartheta \notin \mathrm{fv}(\mathcal{S}_D(\vec{m}_i))$;
(3) by the preceding two items, $\mathrm{Dom}(\underline{S}_i) \subseteq \mathrm{fv}(\mathcal{S}_D(\tau_i)) \setminus \mathrm{fv}(\mathcal{S}_D(\vec{m}_i))$.

Hence it now only remains to verify condition (4) in the definition of solution, i.e., that $(\mathcal{S}, (\underline{S}_1, \ldots, \underline{S}_k))$ satisfies all the constraints in $\mathcal{C}$. See Appendix G for the details. $\Box$

4.2 Constraint Normalization

We define a set of rewriting rules with which a constraint system can be rewritten into another constraint system and a so-called resulting semantic substitution.

Convention 62 *When a semantic substitution $\underline{S}$ satisfies that $Dom(\underline{S}) \cap TyVar = \emptyset$ and $\forall \epsilon \in \mathrm{Dom}(\underline{S}).\underline{S}(\epsilon) = \epsilon.\emptyset$, then we also regard $\underline{S}$ as a syntactic substitution. Thus if $\underline{S}$ is such a semantic substitution, then $\underline{S}(\mathcal{C})$ is well-defined by the definition of application of a syntactic substitution to a multi-set of constraints. Moreover, for such a $\underline{S}$ and an instantiation map I, $\underline{S} \circ I$ is then a well-defined instantiation map.*

The rewriting rules appear in Figure 11 and are numbered $R1$ through $R12$ for future reference. We let Ri range over $\{R1, \ldots, R12\}$. We now explain how to read the rewriting rules in Figures 11 and 12. Each rewriting rule specifies how to rewrite a constraint system $((\mathcal{C}, I, k), D)$ into another constraint system $((\mathcal{C}', I', k), D')$, called the resulting constraint system, and a semantic substitution, called the resulting substitution (the column denoted $\underline{S}$ in Figures 11 and 12). Let $((\mathcal{C}, I, k), D)$ be a constraint system. Rule R1 says that if $c \equiv (\tau_1, \rho_1) \xrightarrow{\epsilon_1} (\tau_1', \rho_1') \doteq (\tau_2, \rho_2) \xrightarrow{\epsilon_2} (\tau_2', \rho_2') \in \mathcal{C}$, then the resulting constraint system is $((\mathcal{C} \setminus \{c\} \cup \{\tau_1 \doteq \tau_2, \rho_1 \doteq \rho_2, \epsilon_1 \doteq \epsilon_2, \tau_1' \doteq \tau_2', \rho_1' \doteq \rho_2'\}, I, k), D)$ and the resulting substitution is the identity substitution. Rules R2, R4, R6, R7, R10, and R11 are read analogously to R1.

Rule R3 says that if $\rho \doteq \rho' \in \mathcal{C}$ then the resulting constraint system is the one obtained by substituting ρ' for ρ in $((\mathcal{C} \setminus \{\rho \doteq \rho'\}, I, k), D)$ (note the application of and composition with $\underline{S} = \{\rho \mapsto \rho'\}$) and the resulting substitution is $\underline{S}$. Rule R5 is read analogously — note here that only $\underline{S}_3$ is applied to $\mathcal{C}$ and composed with I whereas $\underline{S}$ is applied to D and returned as the resulting substitution.

Rules R7, R8, R9, R11, and R12 are only applicable if the resulting constraint system will be different than the given constraint system, and the rules R3, R5, and R8 are only applicable if the stated side-condition is met (i.e., for R3, if $\rho \neq \rho'$, and for R5 and R8, if $\epsilon \neq \epsilon'$).

Rule R7 says that if $\epsilon \mathrel{\dot{\succeq}^i} \epsilon \in \mathcal{C}$, then the resulting constraint system is $((\mathcal{C} \cup \{\rho' \mathrel{\dot{\succeq}^i} \rho' \mid \rho' \in \mathrm{frv}(\mathcal{S}_D(\epsilon))\} \cup \{\epsilon' \mathrel{\dot{\succeq}^i} \epsilon' \mid \epsilon' \in \mathrm{fev}(\mathcal{S}_D(\epsilon))\}, I, k), D)$ and the resulting substitution is the identity substitution. (Note that the constraint

#	Constraint/ Condition	Action/ Replacement	$\underline{S}$
R1	$(\tau_1, \rho_1) \xrightarrow{\epsilon_1} (\tau_1', \rho_1') \doteq$ $(\tau_2, \rho_2) \xrightarrow{\epsilon_2} (\tau_2', \rho_2')$	$\tau_1 \doteq \tau_2,\ \rho_1 \doteq \rho_2,\ \epsilon_1 \doteq \epsilon_2$ $\tau_1' \doteq \tau_2',\ \rho_1' \doteq \rho_2',$	$\underline{\mathrm{Id}}$
R2	$\alpha \mathrel{\dot{\succeq}^i} \tau_1,\ \alpha \mathrel{\dot{\succeq}^i} \tau_2$	$\alpha \mathrel{\dot{\succeq}^i} \tau_1,\ \tau_1 \doteq \tau_2$	$\underline{\mathrm{Id}}$
R3	$\rho \doteq \rho',\ (\rho \neq \rho')$	Let $\underline{S} = \{\rho \mapsto \rho'\}$ in $((\underline{S}(\mathcal{C} \setminus \{\rho \doteq \rho'\}), \underline{S} \circ I, k), \underline{S}(D))$	$\underline{S}$
R4	$\rho \mathrel{\dot{\succeq}^i} \rho_1,\ \rho \mathrel{\dot{\succeq}^i} \rho_2$	$\rho \mathrel{\dot{\succeq}^i} \rho_1,\ \rho_1 \doteq \rho_2$	$\underline{\mathrm{Id}}$
R5	$\epsilon \doteq \epsilon',\ (\epsilon \neq \epsilon')$ $\mathcal{S}_D(\epsilon) = \epsilon.\underline{\varphi}$ $\mathcal{S}_D(\epsilon') = \epsilon'.\underline{\varphi'}$	Let $\underline{S}_1 = \{\epsilon \mapsto \epsilon.(\underline{\varphi} \cup \underline{\varphi'})\}$ $\underline{S}_2 = \{\epsilon' \mapsto \epsilon'.(\underline{\varphi} \cup \underline{\varphi'})\}$ $\underline{S}_3 = \{\epsilon \mapsto \epsilon'.\emptyset\}$ $\underline{S} = \underline{S}_3 \circ \underline{S}_2 \circ \underline{S}_1$ in $((\underline{S}_3(\mathcal{C} \setminus \{\epsilon \doteq \epsilon'\}), \underline{S}_3 \circ I, k), \underline{S}(D))$	$\underline{S}$
R6	$\epsilon \mathrel{\dot{\succeq}^i} \epsilon_1,\ \epsilon \mathrel{\dot{\succeq}^i} \epsilon_2$	$\epsilon \mathrel{\dot{\succeq}^i} \epsilon_1,\ \epsilon_1 \doteq \epsilon_2$	$\underline{\mathrm{Id}}$
R7	$\epsilon \mathrel{\dot{\succeq}^i} \epsilon,\ (*)$	$\forall \rho' \in \mathrm{frv}(\mathcal{S}_D(\epsilon)) : \rho' \mathrel{\dot{\succeq}^i} \rho'$ $\forall \epsilon' \in \mathrm{fev}(\mathcal{S}_D(\epsilon)) : \epsilon' \mathrel{\dot{\succeq}^i} \epsilon'$ $\epsilon \mathrel{\dot{\succeq}^i} \epsilon$	$\underline{\mathrm{Id}}$

$(*)$ means that the rule is only applicable if $((\mathcal{C}', I', k), D')$ then will differ from $((\mathcal{C}, I, k), D)$.

Fig. 11. Rewriting Rules, Part I

$\epsilon \mathrel{\dot{\succeq}^i} \epsilon$ is not removed from the resulting constraint system.) Rules R8 and R10 are read analogously.

Rule R9 says that if $\epsilon \mathrel{\dot{\supseteq}} \vec{\underline{\varphi}} \in \mathcal{C}$, then the resulting constraint system is $((\mathcal{C}, I, k), \underline{S}(D))$ with $\underline{S} = \{\epsilon \mapsto \epsilon.\underline{\varphi} \cup \underline{\varphi'}\}$ (Note that the constraint $\epsilon \mathrel{\dot{\supseteq}} \vec{\underline{\varphi}}$ is not removed from the resulting constraint system.)

Rule R12 is only applied if there is some $i \in \{1, \ldots, k\}$ with $I(i) = (\_, \vec{m}, \_, \_, \_)$ and there is some $m \in \vec{m}$ for which $m \mathrel{\dot{\succeq}^i} m$ is not in $\mathcal{C}$. For such i and m the constraint $m \mathrel{\dot{\succeq}^i} m$ is added to the constraint system. (In practice, rule R12 will never be applied because the required mono constraints $m \mathrel{\dot{\succeq}^i} m$ will be generated by R10.) Let $\underline{S}$ be a semantic substitution and let $((\mathcal{C}, I, k), D)$ and $((\mathcal{C}', I', k), D')$ be constraint systems. We then write $((\mathcal{C}, I, k), D) \Longrightarrow_{(Ri, \underline{S})} ((\mathcal{C}', I', k), D')$ if $((\mathcal{C}', I', k), D')$ is obtained from $((\mathcal{C}, I, k), D)$ by an application of rewriting rule Ri with $\underline{S}$ as resulting substitution. Morever, we write

#	Constraint/ Condition	Action/ Replacement	$\underline{S}$
R8	$\epsilon \dot{\succeq}^i \epsilon'$, $\epsilon \neq \epsilon'$, $\mathcal{S}_D(\epsilon) = \epsilon.\underline{\varphi}$, $I(i) = (\tau, \vec{m}, \vec{\rho}, \vec{\epsilon}, \tau')$, $(*)$	$\forall \rho'' \in \underline{\varphi} \setminus \mathrm{frv}(\mathcal{S}_D(\vec{m}))$. $\quad index(\rho'', \mathcal{S}_D(\tau)) = j \implies$ $\quad \rho'' \dot{\succeq}^i \rho_j, \epsilon' \dot{\supseteq} \rho_j$ $\forall \rho'' \in \underline{\varphi} \cap \mathrm{frv}(\mathcal{S}_D(\vec{m})).\epsilon' \dot{\supseteq} \rho''$ $\forall \epsilon'' \in \underline{\varphi} \setminus \mathrm{fev}(\mathcal{S}_D(\vec{m}))$. $\quad index(\epsilon'', \mathcal{S}_D(\tau)) = j \implies$ $\quad \epsilon'' \dot{\succeq}^i \epsilon_j, \epsilon' \dot{\supseteq} \overline{\epsilon_j}$ $\forall \epsilon'' \in \underline{\varphi} \cap \mathrm{fev}(\mathcal{S}_D(\vec{m})).\epsilon' \dot{\supseteq} \overline{\epsilon''}$ $\epsilon \dot{\succeq}^i \epsilon'$	$\underline{\mathrm{Id}}$
R9	$\epsilon \dot{\supseteq} \vec{\overline{\varphi}}$, $(*)$, $\mathcal{S}_D(\epsilon) = \epsilon.\underline{\varphi}$, $\mathcal{E}_{\mathcal{S}_D}(\vec{\overline{\varphi}}) = \underline{\varphi'}$	Let $\underline{S} = \{\epsilon \mapsto \epsilon.\underline{\varphi} \cup \underline{\varphi'}\}$ in $\quad((\mathcal{C}, I, k), \underline{S}(D))$	$\underline{S}$
R10	$(\tau, \vec{m}) \dot{\succeq}^i (\vec{\rho}, \vec{\epsilon}, \tau')$	$\forall m \in \vec{m}.m \dot{\succeq}^i m$ $\forall (\alpha_0, \tau_0) \in$ tyvar-pairs(τ, τ'). $\quad \alpha_0 \dot{\succeq}^i \tau_0$ $\forall (\rho, \rho') \in$ regvar-pairs(τ, τ'). $\quad \rho \dot{\succeq}^i \rho'$ $\forall (\epsilon, \epsilon') \in$ effectvar-pairs(τ, τ'). $\quad \epsilon \dot{\succeq}^i \epsilon'$	$\underline{\mathrm{Id}}$
R11	$\mathrm{Norm}(\tau, \vec{m})$ $(*)$	$\forall \rho, \rho' \in \mathrm{frv}(\mathcal{S}_D(\tau)) \setminus \mathrm{frv}(\mathcal{S}_D(\vec{m}))$. $\quad index(\rho, \mathcal{S}_D(\tau)) = index(\rho', \mathcal{S}_D(\tau)) \implies$ $\quad \rho \doteq \rho'$ $\forall \epsilon, \epsilon' \in \mathrm{fev}(\mathcal{S}_D(\tau)) \setminus \mathrm{fev}(\mathcal{S}_D(\vec{m}))$. $\quad index(\epsilon, \mathcal{S}_D(\tau)) = index(\epsilon', \mathcal{S}_D(\tau)) \implies$ $\quad \epsilon \doteq \epsilon'$ $\mathrm{Norm}(\tau, \vec{m})$	$\underline{\mathrm{Id}}$ $\underline{\mathrm{Id}}$
R12	$i \in \{1, \ldots, k\}$, $(*)$, $I(i) = (\_, \vec{m}, \_, \_, \_)$	$\forall m \in \vec{m}.m \dot{\succeq}^i m$	$\underline{\mathrm{Id}}$

$(*)$ means that the rule is only applicable if $((\mathcal{C}', I', k), D')$ then will differ from $((\mathcal{C}, I, k), D)$.

Fig. 12. Rewriting Rules, Part II

$((\mathcal{C}, I, k), D) \Longrightarrow_S ((\mathcal{C}', I', k), D')$ if there exists an Ri such that

$$((\mathcal{C}, I, k), D) \Longrightarrow_{(Ri,\underline{S})} ((\mathcal{C}', I', k), D'),$$

and we write $((\mathcal{C}, I, k), D) \not\Longrightarrow_{\underline{S}} ((\mathcal{C}', I', k), D')$ if there does not exists an Ri and $\underline{S}$ such that $((\mathcal{C}, I, k), D) \Longrightarrow_{(Ri,\underline{S})} ((\mathcal{C}', I', k), D')$.

Lemma 63 *Let $((\mathcal{C}, I, k), D)$ be a constraint system. If $((\mathcal{C}, I, k), D) \Longrightarrow_{(Ri,\underline{S})} ((\mathcal{C}', I', k), D')$ then $((\mathcal{C}', I', k), D')$ is a constraint system, $D' = \underline{S}(D)$, and $\underline{S}$ is a contraction of D.*

PROOF. Easy verification. □

Theorem 64 (Soundness of Normalization) *If*

$$((\mathcal{C}_1, I_1, k_1), D_1) \Longrightarrow_{\underline{S}} ((\mathcal{C}_2, I_2, k), D_2)$$

and $\mathcal{S}_2$ is a solution to $((\mathcal{C}_2, I_2, k), D_2)$, then $\mathcal{S}_1 = \mathcal{S}_2 \circ \underline{S}$ is a solution to $((\mathcal{C}_1, I_1, k_1), D_1)$.

PROOF. See Appendix H. □

Theorem 65 (Normal Form) *If $(\mathcal{C}_1, I_1, k), D_1) \not\Longrightarrow_{\underline{S}} (\mathcal{C}_2, I_2, k), D_2)$, then $(\mathcal{C}_1, I_1, k), D_1)$ is in normal form.*

PROOF. Suppose one of the conditions in the definition of normal form is not met; then it is straightforward to see that $(\mathcal{C}_1, I_1, k), D_1) \Longrightarrow_{(Ri,\underline{S})} (\mathcal{C}_2, I_2, k), D_2)$, for some Ri, some $\underline{S}$ and some $(\mathcal{C}_2, I_2, k), D_2)$, contradicting the assumption. □

Theorem 66 (Rewriting is Strongly Normalizing) *Let $((\mathcal{C}_1, I_1, k), D_1)$ be a constraint system. There is no infinite sequence of rewriting steps*

$$(\mathcal{C}_1, I_1, k), D_1) \Longrightarrow_{\underline{S}_1} (\mathcal{C}_2, I_2, k), D_2) \Longrightarrow_{\underline{S}_2} \cdots$$

PROOF. The idea of the proof is naturally to define a suitable size measure for constraint systems and then show that every rewriting rule decreases the size measure. The intuition for why the theorem holds is that the rewriting rules either simplify constraints, make more and more variables monomorphic, or increase the effect sets (which have a bounded size since no new variables are introduced during rewriting).

See Appendix I for the definition of a size measure and the proof of the theorem. □

The following theorem summarizes the constraint normalization.

Theorem 67 *Let $\mathcal{C}$ be a pre-well-tagged set of constraints. Then a solution to $\mathcal{C}$ can be obtained by application of the rewriting rules in Figure 11.*

PROOF. Let $((\mathcal{C}_0, I_0, k), D_0)$ be the constraint system obtained from $\mathcal{C}$ as in the definition of solution to a pre-well-tagged set of constraints. Now apply the rewriting rules R1–R12 exhaustively — in any order. This terminates by Theorem 66 after a finite number, say n, of steps. Write

$$((\mathcal{C}_0, I_0, k), D_0) \Longrightarrow_{\underline{S}_1} ((\mathcal{C}_1, I_1, k), D_1) \Longrightarrow_{\underline{S}_2} \cdots \Longrightarrow_{\underline{S}_n} ((\mathcal{C}_n, I_n, k), D_n)$$

for the sequence of rewriting steps. By Theorem 65, $((\mathcal{C}_n, I_n, k), D_n)$ is in normal form. Hence, by Theorem 61, $\mathcal{S}_{D_n}$ is a solution to $\mathcal{C}_n$. Then by n applications of Theorem 64, $\mathcal{S} = \mathcal{S}_{D_n} \circ \underline{S}_n \circ \underline{S}_{n-1} \circ \cdots \circ \underline{S}_2 \circ \underline{S}_1$ is a solution to $(\mathcal{C}_0, I_0, k), D_0)$, and thus, by definiton, to $\mathcal{C}$, as required. □

As mentioned earlier the rewriting rules are mildly incomplete; we shall discuss in what sense this is the case. By completeness of the rewriting rules we mean that all solutions are preserved during normalization. In other words, rewriting must not lead to the identification of variables which are not strictly required to be identified for a solution to exist and, moreover, the rewriting rules may not accidentally regard a variable to be an instance of another variable if this is not required to be the case for a solution to exist. Two of our rewriting rules are not complete in this sense: R8 and R11. (The remaining rules do preserve all solutions in a sense which can be made precise.)

For R11 the problem is that two variables, say ρ and ρ', may be identified (i.e., a constraint $\rho \doteq \rho'$ may be added to the set of constraints) because they have the same index. But this just means that they have the same index *at this stage of the rewriting process*, not necessarily that they will have the same index later on. For instance, ρ' may become unified with another variable ρ'' which occurs on a different subset of the arrows in $\mathcal{S}_D(\tau)$ such that, after this unification, ρ and ρ' do not have the same index.

For R8 the problem has also to do with the indices: a variable ρ with index 1, say, may be instantiated to a variable, ρ' say, also with index 1. That is, a constraint $\rho \dot{\succeq}^i \rho'$ is added to the set of constraints. But, again, later on the index of ρ may change, to 2 say, such that a constraint $\rho \dot{\succeq}^i \rho''$ is added to the set of constraints and thus ρ' and ρ'' will be identified (by rule R4). In other words, it is not a good idea to add the constraint $\rho \dot{\succeq}^i \rho'$ in rule R8 when the index of ρ can possibly change later on.

We see that, for both R8 and R11, the problem is that the index of a variable may change during rewriting. It is a natural question to ask whether one can choose (possibly using information about how the constraints were generated from the syntax in the first place) an ordering to apply the rewriting rules in which would ensure that the correct final indices of secondary variables could be computed *before* taking instances and normalizing type schemes. So far we have not been successful in answering this question and thus we do not have a complete rewriting strategy (we do not even know if a complete non-backtracking rewriting strategy exists). The reason we still maintain that the overall algorithm is only "mildly incomplete" is that the incompleteness has only to do with possible unwanted identifications of secondary variables and thus in many cases the algorithm can still infer more general type schemes than the algorithm in [14].

5 Implementation

We have implemented the constraint-based region inference algorithm for the Standard ML Core Language in the ML Kit. Given an untyped input expression e, the implementation first type checks e according to the Milner type discipline, producing an explicitly typed expression xe. The type inference is done by an efficient algorithm inspired by Remy [12]. Next, a constraint extraction module extract constraints from xe and inserts the generated variables into the syntax tree of xe. Then the constraints are normalized. Finally, a region annotated expression is obtained by traversing xe and using the solution to the constraints, as described by the constraint-based region inference rules. The implementation of the constraint extraction module and the constraint normalization module consists of approximately 5000 lines of Standard ML code.

The implementation uses a graph representation which is designed to make normalisation fast. Union-find data structures are used to implement substitutions. The nodes in the graph all contain mutable cells for efficient union operations. Equality constraints are always rewritten before any other constraints so that, for example, in $R11$ we can decide whether the rule is applicable just by testing whether the relevant variables are equal (rather than checking whether an equality constraint exists). Preliminary measurements on a range of benchmarks suggest that the constraint-based algorithm is about twice as fast as the syntax-directed algorithm. The constraint-based algorithm is also considerably more space consuming, however. It is difficult to make a precise comparison between the two implementations, since both are fairly highly engineered for speed, but in very different ways. Both algorithms have worst-case time complexity at most $O(n^4)$, where n is the size of the explicitly typed source program, but in practice they are much faster.

6 Conclusion

Returning to the potential advantages of basing region inference on constraint solving (Section 1.3), let us consider what has been achieved. From an engineering point of view, the separation of constraint generation and solution was certainly an advantage - it simplified the implementation. Also, implementation techniques for efficient representations of constraint sets were certainly useful.

The constraints for region inference we use are essentially standard set-based constraints, extended with two special constraints (the ones involving Norm and Obs). We are not aware of any general type of constraint system that has these ad-hoc constraints as a special case.

As for the question of principal types for region inference, we identified two rewrite rules that are "responsible" for the loss of completeness. Both rules point to secondary region and effect variables as the culprits. On the one hand, it seems necessary to unify secondary variables in order to ensure termination; on the other hand it is not clear (to the authors, at least) when such unification can be done without losing solutions.

A Free Variables of Semantic Objects

For a semantic effect, $\underline{\varphi}$, $\mathrm{ftv}(\underline{\varphi})$ denotes the set of free type variables in $\underline{\varphi}$; $\mathrm{frv}(\underline{\varphi})$ denotes the set of free region variables in $\underline{\varphi}$; $\mathrm{fev}(\underline{\varphi})$ denotes the set of free effect variables in $\underline{\varphi}$; and $\mathrm{fv}(\underline{\varphi})$ denotes the set of free variables of either kind in $\underline{\varphi}$. Formally, these free variable functions also yield sequences of variables and are defined thus:

$$
\begin{aligned}
\mathrm{ftv}(\underline{\varphi}) &= \emptyset \\
\mathrm{frv}(\underline{\varphi}) &= \mathrm{seq}(\{\rho \mid \rho \in \underline{\varphi} \in \underline{\varphi}\}) \\
\mathrm{fev}(\underline{\varphi}) &= \mathrm{seq}(\{\epsilon \mid \epsilon \in \underline{\varphi}\}) \\
\mathrm{fv}(\underline{\varphi}) &= \mathrm{frv}(\underline{\varphi}) \mathbin{\overline{\cup}} \mathrm{fev}(\underline{\varphi})
\end{aligned}
$$

For a semantic arrow effect, $\epsilon.\underline{\varphi}$, $\mathrm{ftv}(\epsilon.\underline{\varphi})$ denotes the set of free type variables in $\epsilon.\underline{\varphi}$; $\mathrm{frv}(\epsilon.\underline{\varphi})$ denotes the set of free region variables in $\epsilon.\underline{\varphi}$; $\mathrm{fev}(\epsilon.\underline{\varphi})$ denotes the set of free effect variables in $\epsilon.\underline{\varphi}$; and $\mathrm{fv}(\epsilon.\underline{\varphi})$ denotes the set of free variables of either kind in $\underline{\varphi}$. Formally, these free variable functions are defined thus:

$$
\begin{aligned}
\mathrm{ftv}(\epsilon.\underline{\varphi}) &= \emptyset \\
\mathrm{frv}(\epsilon.\underline{\varphi}) &= \mathrm{frv}(\underline{\varphi}) \\
\mathrm{fev}(\epsilon.\underline{\varphi}) &= \epsilon \mathbin{\overline{\cup}} \mathrm{fev}(\underline{\varphi}) \\
\mathrm{fv}(\epsilon.\underline{\varphi}) &= \mathrm{frv}(\epsilon.\underline{\varphi}) \mathbin{\overline{\cup}} \mathrm{fev}(\epsilon.\underline{\varphi})
\end{aligned}
$$

For a semantic type $\underline{\tau}$, $\mathrm{ftv}(\underline{\tau})$ denotes the set of free type variables in $\underline{\tau}$; $\mathrm{frv}(\underline{\tau})$ denotes the set of free region variables in $\underline{\tau}$; $\mathrm{fev}(\underline{\tau})$ denotes the set of free effect variables in $\underline{\tau}$; and $\mathrm{fv}(\underline{\tau})$ denotes the set of free variables of either kind in $\underline{\tau}$. Formally, these free variable functions also yield sequences of variables and are defined thus:

$$
\begin{aligned}
\mathrm{ftv}(\alpha) &= \alpha & \mathrm{ftv}((\underline{\tau},\rho) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}',\rho')) &= \mathrm{ftv}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{ftv}(\underline{\tau}') \\
\mathrm{frv}(\alpha) &= \emptyset & \mathrm{frv}((\underline{\tau},\rho) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}',\rho')) &= \mathrm{frv}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{frv}(\underline{\tau}') \\
& & &\mathbin{\overline{\cup}}\ \mathrm{frv}(\underline{\varphi}) \mathbin{\overline{\cup}} \rho \mathbin{\overline{\cup}} \rho' \\
\mathrm{fev}(\alpha) &= \emptyset & \mathrm{fev}((\underline{\tau},\rho) \xrightarrow{\epsilon.\underline{\varphi}} (\underline{\tau}',\rho')) &= \mathrm{fev}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{fev}(\underline{\tau}') \mathbin{\overline{\cup}} \mathrm{fev}(\epsilon.\underline{\varphi})
\end{aligned}
$$

$$
\mathrm{fv}(\underline{\tau}) = \mathrm{ftv}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{frv}(\underline{\tau}) \mathbin{\overline{\cup}} \mathrm{fev}(\underline{\tau})
$$

B Free Variables of Syntactic Objects

The free variables of syntactic objects are defined thus:

$$
\begin{aligned}
\mathrm{ftv}(\alpha) &= \alpha & \mathrm{ftv}((\tau,\rho) \xrightarrow{\epsilon} (\tau',\rho')) &= \mathrm{ftv}(\tau) \mathbin{\overline{\cup}} \mathrm{ftv}(\tau') \\
\mathrm{frv}(\alpha) &= \emptyset & \mathrm{frv}((\tau,\rho) \xrightarrow{\epsilon} (\tau',\rho')) &= \mathrm{frv}(\tau) \mathbin{\overline{\cup}} \mathrm{frv}(\tau') \mathbin{\overline{\cup}} \rho \mathbin{\overline{\cup}} \rho' \\
\mathrm{fev}(\alpha) &= \emptyset & \mathrm{fev}((\tau,\rho) \xrightarrow{\epsilon} (\tau',\rho')) &= \mathrm{fev}(\tau) \mathbin{\overline{\cup}} \mathrm{fev}(\tau') \mathbin{\overline{\cup}} \epsilon
\end{aligned}
$$

$$
\mathrm{fv}(\tau) = \mathrm{ftv}(\tau) \mathbin{\overline{\cup}} \mathrm{frv}(\tau) \mathbin{\overline{\cup}} \mathrm{fev}(\tau)
$$

For a pair of a syntactic type, τ, and a region variables, ρ, we define:

$$
\begin{aligned}
\mathrm{ftv}(\tau,\rho) &= \mathrm{ftv}(\tau) \\
\mathrm{frv}(\tau,\rho) &= \mathrm{frv}(\tau) \mathbin{\overline{\cup}} \rho \\
\mathrm{fev}(\tau,\rho) &= \mathrm{fev}(\tau) \\
\mathrm{fv}(\tau,\rho) &= \mathrm{fv}(\tau) \mathbin{\overline{\cup}} \rho
\end{aligned}
$$

For syntactic mono objects, m and for syntactic mono object sequences, $\vec{m}$, $\text{frv}(m)$ and $\text{frv}(\vec{m})$ denotes the set of free region variables in m and $\vec{m}$, respectively. Likewise for ftv, fev, and fv. Formally, we define:

$$\text{frv}(\rho) = \rho \;\; \text{ftv}(\rho) = \emptyset \;\; \text{fev}(\rho) = \emptyset$$
$$\text{frv}(\epsilon) = \emptyset \;\; \text{ftv}(\epsilon) = \emptyset \;\; \text{fev}(\epsilon) = \epsilon$$
$$\text{frv}(\alpha) = \emptyset \;\; \text{ftv}(\alpha) = \alpha \;\; \text{fev}(\alpha) = \emptyset$$

$$\text{fv}(m) = \text{frv}(m) \mathbin{\overline{\cup}} \text{ftv}(m) \mathbin{\overline{\cup}} \text{fev}(m)$$

$$\begin{aligned}
\text{frv}(m_1, \ldots, m_n) &= \text{frv}(m_1) \mathbin{\overline{\cup}} \cdots \mathbin{\overline{\cup}} \text{frv}(m_n)\\
\text{ftv}(m_1, \ldots, m_n) &= \text{ftv}(m_1) \mathbin{\overline{\cup}} \cdots \mathbin{\overline{\cup}} \text{ftv}(m_n)\\
\text{fev}(m_1, \ldots, m_n) &= \text{fev}(m_1) \mathbin{\overline{\cup}} \cdots \mathbin{\overline{\cup}} \text{fev}(m_n)\\
\text{fv}(\vec{m}) &= \text{frv}(\vec{m}) \mathbin{\overline{\cup}} \text{ftv}(\vec{m}) \mathbin{\overline{\cup}} \text{fev}(\vec{m})
\end{aligned}$$

Further, for a syntactic effect $\overline{\varphi}$, $\text{fv}(\overline{\varphi})$ denotes the set of free variables of $\overline{\varphi}$:

$$\begin{aligned}
\text{fv}(\rho) &= \rho\\
\text{fv}(\text{Obs}(\vec{m}, (\tau, \rho), \epsilon)) &= \text{fv}(\vec{m}) \mathbin{\overline{\cup}} \text{fv}(\tau) \mathbin{\overline{\cup}} \rho \mathbin{\overline{\cup}} \epsilon\\
\text{fv}(\epsilon) &= \epsilon\\
\text{fv}(\overline{\epsilon}) &= \epsilon
\end{aligned}$$

C Application of Semantic Substitution to Semantic Mono Objects

The application of a semantic substitution to semantic mono object and semantic mono object sequences is defined in the following way:

Semantic Mono Objects

$$\begin{aligned}
\underline{S}(\rho) &= \underline{S_r}(\rho)\\
\underline{S}(\epsilon.\underline{\varphi}) &= \underline{S}(\epsilon.\underline{\varphi})\\
\underline{S}(\underline{\tau}) &= \underline{S}(\underline{\tau})
\end{aligned} \tag{C.1}$$

where the definition of application of a semantic substitution to an arrow effect and a semantic type is used on the right hand sides of (C.1) (i.e., this is a good definition).

Semantic Mono Object Sequences

$$\underline{S}(\underline{m_1}\cdots\underline{m_n}) = (\underline{S}(\underline{m_1}))\cdots(\underline{S}(\underline{m_n}))$$

D Application of Semantic Substitutions to Syntactic Objects

Semantic substitutions can also be applied to syntactic objects; such an application results in a semantic object. Application of a semantic substitution $\underline{S} = (\underline{S_r}, \underline{S_t}, \underline{S_e})$ to variables, syntactic types, syntactic mono objects, and syntactic mono object sequences is defined to be the simultaneous application of $\underline{S_r}$, $\underline{S_t}$, and $\underline{S_e}$:

Variables

$$\begin{aligned} \underline{S}(\alpha) &= \underline{S_t}(\alpha) \\ \underline{S}(\rho) &= \underline{S_r}(\rho) \\ \underline{S}(\epsilon) &= \underline{S_e}(\epsilon) \end{aligned}$$

Syntactic Types

$$\begin{aligned} \underline{S}(\alpha) &= \underline{S_t}(\alpha) \\ \underline{S}((\tau,\rho) \xrightarrow{\epsilon} (\tau',\rho')) &= (\underline{S}(\tau), \underline{S_r}(\rho)) \xrightarrow{\underline{S_e}(\epsilon)} (\underline{S}(\tau'), \underline{S_r}(\rho')) \end{aligned}$$

Syntactic Mono Objects

$$\begin{aligned} \underline{S}(\alpha) &= \underline{S_t}(\alpha) \\ \underline{S}(\rho) &= \underline{S_r}(\rho) \\ \underline{S}(\epsilon) &= \underline{S_e}(\epsilon) \end{aligned}$$

Syntactic Mono Object Sequences

$$\underline{S}(m_1\cdots m_n) = (\underline{S}(m_1))\cdots(\underline{S}(m_n))$$

Note that application of a semantic substitution to a syntactic type, mono object, or mono object sequence, yields a *semantic* object of the corresponding kind.

E Proof of Soundness of Constraint-based Formulation of Region Inference

In this section we prove the soundness of the constraint-based formulation of region inference with respect to the semantic region inference rules. We first recall the Theorem 57 from Section 3.3:

Theorem 68 *Let $\mathcal{S}$ be a solution substitution. If $\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} e \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$, and $\Gamma \simeq_{\mathcal{S}} TE$ then there exist $\underline{\epsilon}$ and $\underline{\varphi}$ such that TE is well-formed and $TE \vdash e \Rightarrow e' : (\mathcal{S}(\tau), \mathcal{S}(\rho)), \underline{\varphi}$ and $\mathcal{S}(\epsilon) = \underline{\epsilon}.\underline{\varphi}$.*

In the proof of the theorem, we will make use of the following lemma, which is straightforward to prove.

Lemma 69 *If $TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}$, then* $\mathrm{ML}(\underline{\tau}) = \mathit{MLTypeOf}(e)$.

Proof of Theorem 68 By rule induction on

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} e \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$$

proceeding by cases on the last rule applied. In the proof we write $\Vdash_{\mathcal{S}} \mathbf{c_1}, \cdots, \mathbf{c_k}$ ($\mathbf{k} \geq 1$) to express that $\mathcal{S}$ solves the constraints numbered $\mathbf{c_1}, \ldots, \mathbf{c_k}$ in the rule under consideration.

Case (Rule (19)**)** Then

$$\begin{aligned}\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} &\ \texttt{letrec}\ f : \forall \vec{\alpha_0}.\overline{\tau_0}(x : \overline{\tau_1})\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\ &\ \texttt{letregion}\ \vec{\rho}'\ \texttt{in} \\ &\quad \texttt{letrec}\ f\texttt{[}\vec{\rho}\texttt{]}\texttt{(}x\texttt{)}\ \texttt{at}\ \mathcal{S}(\rho_0)\ \texttt{=}\ e_1'\ \texttt{in}\ e_2' : (\tau_2, \rho_2), \epsilon, D_6, \mathcal{C}_6\end{aligned} \tag{E.1}$$

and by inversion of rule (19) we get

$$(\tau_0, \rho_0, D_2) = \mathrm{spread}(\overline{\tau_0}, D_1) \tag{E.2}$$
$$\vec{m_3} = \mathrm{mono}(\Gamma) \tag{E.3}$$
$$\vec{\alpha_1} = \mathrm{ftv}(\tau_0) \tag{E.4}$$
$$(\epsilon_{11}, \epsilon_3, D_3) = \mathrm{freshEffVar}(D_2) \tag{E.5}$$
$$\vec{m_1} = \vec{\alpha_1}\,\rho_0\epsilon_{11}\epsilon_3 \tag{E.6}$$
$$\mathcal{C}_2 = \mathcal{C}_1 \cup \{\mathrm{Norm}(\tau_0, \vec{m_1}\,\vec{m_3}\,)\} \tag{E.7}$$
$$\begin{aligned}\mathcal{C}_2, D_3, \Gamma + \{f \mapsto ((\tau_0, \vec{m_1}\,\vec{m_3}\,), \rho_0)\} \Vdash_{\mathcal{S}}\ & \lambda\, x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow \\ & \lambda\, x.e_1'\ \texttt{at}\ \mathcal{S}(\rho_1) : (\tau_1, \rho_1), \epsilon_1, D_4, \mathcal{C}_3\end{aligned} \tag{E.8}$$
$$\vec{\alpha_2} = \mathrm{ftv}(\tau_0) \setminus \vec{\alpha_0} \tag{E.9}$$
$$\vec{m_2} = \vec{\alpha_2}\,\rho_0\epsilon_{11}\epsilon_3 \tag{E.10}$$
$$\mathcal{C}_4 = \mathcal{C}_3 \cup \{\mathrm{Norm}(\tau_0, \vec{m_2}\,\vec{m_3}\,)\} \tag{E.11}$$
$$\mathcal{C}_4, D_4, \Gamma + \{f \mapsto ((\tau_0, \vec{m_2}\,\vec{m_3}\,), \rho_0)\} \Vdash_{\mathcal{S}} e_2 \Rightarrow e_2' : (\tau_2, \rho_2), \epsilon_2, D_5, \mathcal{C}_5 \tag{E.12}$$
$$(\epsilon_4, \epsilon, D_6) = \mathrm{freshEffVar}(D_5) \tag{E.13}$$
$$\mathcal{S}\ \mathrm{solves}\ \mathcal{C}_6 = \{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}\} \cup \mathcal{C}_5 \tag{E.14}$$
$$\mathcal{S}(\epsilon_4) = \underline{\epsilon_4}.\underline{\varphi_4} \tag{E.15}$$
$$\mathcal{S}(\epsilon) = \underline{\epsilon}.\underline{\varphi} \tag{E.16}$$
$$\vec{\rho}\,' = \mathrm{frv}(\underline{\varphi_4} \setminus \underline{\varphi}) \tag{E.17}$$
$$\underline{\sigma} = \forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\mathcal{S}(\tau_0) = \mathrm{mk}\_\underline{\sigma}(\mathcal{S}, (\tau_0, \vec{m_2}\,\vec{m_3}\,) \tag{E.18}$$
$$\vec{\alpha_0} \cap \mathrm{ftv}(\vec{m_3}\,) = \emptyset. \tag{E.19}$$

with $\mathbf{c_1}, \ldots, \mathbf{c_4}$ as in rule (19). Further, by assumption we have

$$\Gamma \simeq_{\mathcal{S}} TE \tag{E.20}$$

and, since $\mathcal{S}$ is a solution substitution,

$$\mathrm{Dom}(\mathcal{S}) \cap \mathrm{TyVar} = \emptyset. \tag{E.21}$$

Let

$$\hat{\underline{\sigma}} = \forall\vec{\rho}\,\vec{\epsilon}.\mathcal{S}(\tau_0). \tag{E.22}$$

Then by (E.18), (E.22), (E.6), (E.3), and (E.20) we get that

$$\Gamma + \{f \mapsto ((\tau_0, \vec{m_1}\,\vec{m_3}\,), \rho_0)\} \simeq_{\mathcal{S}} TE + \{f \mapsto (\hat{\underline{\sigma}}, \mathcal{S}(\rho_0))\}. \tag{E.23}$$

By induction on (E.8) and (E.23) there exist $\underline{\epsilon_1}.\underline{\varphi_1}$ such that

$$TE + \{f \mapsto (\underline{\hat{\sigma}}, \mathcal{S}(\rho_0))\} \vdash \lambda\, x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow \tag{E.24}$$

$$\lambda\, x.e_1' \texttt{ at } \mathcal{S}(\rho_1) : (\mathcal{S}(\tau_1), \mathcal{S}(\rho_1)), \underline{\varphi_1}$$

$$\mathcal{S}(\epsilon_1) = \underline{\epsilon_1}.\underline{\varphi_1}. \tag{E.25}$$

By (E.20) and Lemma 36

$$\mathrm{fv}(TE) \subseteq \mathrm{fv}(\mathcal{S}(\vec{m_3}\,)) \tag{E.26}$$

so by (E.18) and (E.26) we have

$$\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon} \cap \mathrm{fv}(TE) = \emptyset. \tag{E.27}$$

Moreover, by (E.6), (E.10), (E.18), $\Vdash_{\mathcal{S}} \mathbf{c_4}$, and (E.25)

$$\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon} \cap \underline{\varphi_1} = \emptyset. \tag{E.28}$$

By (E.20) and (E.18),

$$\Gamma + \{f \mapsto ((\tau_0, \vec{m_2}\,\vec{m_3}\,), \rho_0)\} \simeq_{\mathcal{S}} TE + \{f \mapsto (\underline{\sigma}, \mathcal{S}(\rho_0))\}. \tag{E.29}$$

By induction on (E.12) and (E.29) there exist $\underline{\epsilon_2}$, $\underline{\varphi_2}$ such that

$$TE + \{f \mapsto (\underline{\sigma}, \mathcal{S}(\rho_0))\} \vdash e_2 \Rightarrow e_2' : (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \underline{\varphi_2} \tag{E.30}$$

$$\mathcal{S}(\epsilon_2) = \underline{\epsilon_2}.\underline{\varphi_2}. \tag{E.31}$$

By Lemma 69 on (E.30) we get that

$$\mathrm{ML}(\mathcal{S}(\tau_2)) = \mathrm{MLTypeOf}(e_2) \tag{E.32}$$

and by the ML well-typedness of $\texttt{letrec}\ f : \forall\vec{\alpha_0}\,.\overline{\tau_0}(x : \overline{\tau_1})\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau}$ and (E.32) we have that

$$\overline{\tau} = \mathrm{MLTypeOf}(e_2) \tag{E.33}$$

so by (E.32) and (E.33)

$$\mathrm{ML}(\mathcal{S}(\tau_2)) = \overline{\tau}. \tag{E.34}$$

Moreover,

$$\mathrm{ML}(\forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\mathcal{S}(\tau_0)) = \forall\vec{\alpha_0}\,.\overline{\tau_0} \tag{E.35}$$

because

(i) $\mathrm{ML}(\mathcal{S}(\tau_1)) = \overline{\tau_0}$ by Lemma 69 on (E.24) and since $\mathcal{S}(\tau_1) = \mathcal{S}(\tau_0)$ by $\Vdash_{\mathcal{S}} \mathbf{c_1}$; and

(ii) $\vec{\alpha} = \vec{\alpha_0}$ by (E.1) (see rule (19)).

Hence we can apply rule (6) on (E.24); (E.27), (E.28); (E.30); (E.34); (E.35); and using $\Vdash_{\mathcal{S}} \mathbf{c_1}$ to conclude

$$\begin{aligned} &TE \vdash \texttt{letrec}\ f : \forall\vec{\alpha_0}.\overline{\tau_0}(x : \overline{\tau_1})\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\ &\qquad \texttt{letrec}\ f[\vec{\rho}](x)\ \texttt{at}\ \mathcal{S}(\rho_0)\ \texttt{=}\ e_1'\ \texttt{in}\ e_2' : (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \underline{\varphi_1} \cup \underline{\varphi_2}. \end{aligned} \tag{E.36}$$

By (E.36) and $\Vdash_{\mathcal{S}} \mathbf{c_2}$ we can apply rule (8) to conclude

$$\begin{aligned} &TE \vdash \texttt{letrec}\ f : \forall\vec{\alpha_0}.\overline{\tau_0}(x : \overline{\tau_1})\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\ &\qquad \texttt{letrec}\ f[\vec{\rho}](x)\ \texttt{at}\ \mathcal{S}(\rho_0)\ \texttt{=}\ e_1'\ \texttt{in}\ e_2' : (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \underline{\varphi_4}. \end{aligned} \tag{E.37}$$

By $\Vdash_{\mathcal{S}} \mathbf{c_3}$, (E.20), (E.15), and (E.16) we get that

$$\underline{\varphi} \supseteq \text{Observe}(TE, (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \underline{\varphi_4}). \tag{E.38}$$

Finally, by (E.37), (E.38), and (E.17) we can apply rule (7) to get

$$\begin{aligned} &TE \vdash \texttt{letrec}\ f : \forall\vec{\alpha_0}.\overline{\tau_0}(x : \overline{\tau_1})\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\ &\qquad \texttt{letregion}\ \vec{\rho}'\ \texttt{in} \\ &\qquad\quad \texttt{letrec}\ f[\vec{\rho}](x)\ \texttt{at}\ \mathcal{S}(\rho_0)\ \texttt{=}\ e_1'\ \texttt{in}\ e_2' : (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \underline{\varphi} \end{aligned} \tag{E.39}$$

which together with (E.16) is the required, completing the proof of this case.

Case (Rule (20)**)** Then

$$\begin{aligned} &\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}}\ e_1\ e_2 : \overline{\tau} \Rightarrow \\ &\qquad\qquad \texttt{letregion}\ \vec{\rho}\ \texttt{in}\ e_1'\ e_2' : (\tau_{12}, \rho_{12}), \epsilon, D_5, \mathcal{C}_4 \end{aligned} \tag{E.40}$$

and by inversion of rule (20) we get

$$\Big((\tau_{11}, \rho_{11}) \xrightarrow{\epsilon_3} (\tau_{12}, \rho_{12}), \_, D_2\Big) = \text{spread}(\text{MLTypeOf}(e_1), D_1) \tag{E.41}$$

$$\mathcal{C}_1, D_2, \Gamma \Vdash_{\mathcal{S}} e_1 \Rightarrow e_1' : (\tau_1, \rho_1), \epsilon_1, D_3, \mathcal{C}_2 \tag{E.42}$$

$$\mathcal{C}_2, D_3, \Gamma \Vdash_{\mathcal{S}} e_2 \Rightarrow e_2' : (\tau_2, \rho_2), \epsilon_2, D_4, \mathcal{C}_3 \tag{E.43}$$

$$(\epsilon_4, \epsilon, D_5) = \text{freshEffVar}(D_4) \tag{E.44}$$

$$\mathcal{S}\ \text{solves}\ \mathcal{C}_4 = \{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}\} \cup \mathcal{C}_3 \tag{E.45}$$

$$\mathcal{S}(\epsilon_4) = \underline{\epsilon_4}.\underline{\varphi_4} \tag{E.46}$$

$$\mathcal{S}(\epsilon) = \underline{\epsilon}.\underline{\varphi} \tag{E.47}$$

$$\vec{\rho} = \text{frv}(\underline{\varphi_4} \setminus \underline{\varphi}) \tag{E.48}$$

with $\mathbf{c_1}, \ldots, \mathbf{c_4}$ as in rule (20). Further by assumption we have

$$\Gamma \simeq_{\mathcal{S}} TE. \tag{E.49}$$

By induction on (E.42) and (E.49) there exist $\underline{\epsilon_1}$, $\underline{\varphi_1}$ such that

$$TE \vdash e_1 \Rightarrow e_1' : (\mathcal{S}(\tau_1), \mathcal{S}(\rho_1)), \underline{\varphi_1} \tag{E.50}$$
$$\mathcal{S}(\epsilon_1) = \underline{\epsilon_1}.\underline{\varphi_1}. \tag{E.51}$$

By induction on (E.43) and (E.49) there exist $\underline{\epsilon_2}$, $\underline{\varphi_2}$ such that

$$TE \vdash e_2 \Rightarrow e_2' : (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \underline{\varphi_2} \tag{E.52}$$
$$\mathcal{S}(\epsilon_2) = \underline{\epsilon_2}.\underline{\varphi_2}. \tag{E.53}$$

By Lemma 69 on (E.50),

$$\mathrm{ML}(\mathcal{S}(\tau_1)) = \mathrm{ML}(\tau_1) = \mathrm{MLTypeOf}(e_1) \tag{E.54}$$

and thus by ML well-typedness of $e_1\ e_2 : \overline{\tau}$ and $\Vdash_{\mathcal{S}} \mathbf{c_1}$,

$$\mathrm{ML}(\mathcal{S}(\tau_{12})) = \overline{\tau}. \tag{E.55}$$

Then by (E.50), (E.52), (E.55), $\Vdash_{\mathcal{S}} \mathbf{c_1}$, and $\Vdash_{\mathcal{S}} \mathbf{c_2}$ we can apply rule (5) to get

$$TE \vdash e_1\ e_2 \Rightarrow e_1'\ e_2' : \Big(\mathcal{S}(\tau_{12}), \mathcal{S}(\rho_{12})\Big), \underline{\varphi_1} \cup \underline{\varphi_2} \cup \underline{\varphi_3} \cup \{\underline{\epsilon_3}, \mathcal{S}(\rho_1)\} \tag{E.56}$$

where we have let

$$\underline{\epsilon_3}.\underline{\varphi_3} = \mathcal{S}(\epsilon_3). \tag{E.57}$$

By $\Vdash_{\mathcal{S}} \mathbf{c_3}$, (E.56), and (E.46) we can apply rule (8) to get

$$TE \vdash e_1\ e_2 \Rightarrow e_1'\ e_2' : \Big(\mathcal{S}(\tau_{12}), \mathcal{S}(\rho_{12})\Big), \underline{\varphi_4} \tag{E.58}$$

and by rule 7 on (E.58), (E.47), $\Vdash_{\mathcal{S}} \mathbf{c_4}$, (E.46), and (E.48) we get

$$TE \vdash e_1\ e_2 \Rightarrow \texttt{letregion}\ \vec{\rho}\ \ \texttt{in}\ e_1'\ e_2' : \Big(\mathcal{S}(\tau_{12}), \mathcal{S}(\rho_{12})\Big), \underline{\varphi} \tag{E.59}$$

which together with (E.47) is the required, completing the proof of this case.

Case (Rule (17)**)** Easy by inversion, application of rule (2), and application of rule 8.

Case (Rule (18)**)** Then

$$\begin{aligned}\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} \lambda\, x : \overline{\tau_1}.e : \overline{\tau} \Rightarrow \\ \lambda\, x.e'\ \texttt{at}\ \mathcal{S}(\rho) : \Big((\tau_1, \rho_1) \xrightarrow{\epsilon_3} (\tau_2, \rho_2), \rho\Big), \epsilon, D_5, \mathcal{C}_3\end{aligned} \tag{E.60}$$

and by inversion of rule (18) we get

$$(\tau_1, \rho_1, D_2) = \text{spread}(\overline{\tau_1}, D_1) \tag{E.61}$$
$$\mathcal{C}_1, D_2, \Gamma + \{x \mapsto (\tau_1, \rho_1)\} \Vdash_{\mathcal{S}} e \Rightarrow e' : (\tau_2, \rho_2), \epsilon_2, D_3, \mathcal{C}_2 \tag{E.62}$$
$$(\rho, D_4) = \text{freshRegVar}(D_3) \tag{E.63}$$
$$(\epsilon, \epsilon_3, D_5) = \text{freshEffVar}(D_4) \tag{E.64}$$
$$\mathcal{S} \text{ solves } \mathcal{C}_3 = \{\mathbf{c_1}, \mathbf{c_2}\} \cup \mathcal{C}_2 \tag{E.65}$$

with $\mathbf{c_1}, \mathbf{c_2}$ as in rule(18). Further by assumption we have that

$$\Gamma \simeq_{\mathcal{S}} TE. \tag{E.66}$$

By (E.66) we get that

$$\Gamma + \{x \mapsto (\tau_1, \rho_1)\} \simeq_{\mathcal{S}} TE + \{x \mapsto (\mathcal{S}(\tau_1), \mathcal{S}(\rho_1))\} \tag{E.67}$$

so by induction on (E.62) and (E.67) there exist $\underline{\epsilon_2}$ and $\underline{\varphi_2}$ such that

$$TE + \{x \mapsto (\mathcal{S}(\tau_1), \mathcal{S}(\rho_1))\} \vdash e \Rightarrow e' : (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \underline{\varphi_2} \tag{E.68}$$
$$\mathcal{S}(\epsilon_2) = \underline{\epsilon_2}.\underline{\varphi_2}. \tag{E.69}$$

Clearly, by (E.61),

$$\text{ML}(\mathcal{S}(\tau_1)) = \text{ML}(\tau_1) = \overline{\tau_1} \tag{E.70}$$

and by Lemma 69 on (E.68)

$$\text{ML}(\mathcal{S}(\tau_2)) = \text{MLTypeOf}(e) \tag{E.71}$$

so by ML well-typedness of $\lambda\, x : \overline{\tau_1}.e : \overline{\tau}$ we get that

$$\text{ML}\left(\mathcal{S}((\tau_1, \rho_1) \xrightarrow{\epsilon_3} (\tau_2, \rho_2))\right) = \overline{\tau}. \tag{E.72}$$

Let

$$\underline{\epsilon_3}.\underline{\varphi_3} = \mathcal{S}(\epsilon_3). \tag{E.73}$$

By $\Vdash_{\mathcal{S}} \mathbf{c_1}$, (E.68), (E.70), and (E.72) we can apply rule (4) to conclude

$$\begin{aligned} &TE \vdash \lambda\, x : \overline{\tau_1}.e : \overline{\tau} \Rightarrow \\ &\quad \lambda\, x.e \texttt{ at } \mathcal{S}(\rho) : \left((\mathcal{S}(\tau_1), \mathcal{S}(\rho_1)) \xrightarrow{\underline{\epsilon_3}.\underline{\varphi_3}} (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \mathcal{S}(\rho)\right), \{\mathcal{S}(\rho)\}. \end{aligned} \tag{E.74}$$

Let

$$\underline{\epsilon}.\underline{\varphi} = \mathcal{S}(\epsilon). \tag{E.75}$$

Then by $\Vdash_{\mathcal{S}} \mathbf{c_2}$ and (E.74) we can apply rule (8) to conclude

$$\begin{aligned} & TE \vdash \lambda\, x : \overline{\tau_1}.e : \overline{\tau} \Rightarrow \\ & \qquad \lambda\, x.e \texttt{ at } \mathcal{S}(\rho) : \Big((\mathcal{S}(\tau_1), \mathcal{S}(\rho_1)) \xrightarrow{\epsilon_3.\varphi_3} (\mathcal{S}(\tau_2), \mathcal{S}(\rho_2)), \mathcal{S}(\rho)\Big), \underline{\varphi} \end{aligned} \tag{E.76}$$

which together with (E.75) is the desired, completing the proof of this case.

Case (Rule (16)**)** Then

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}} f : \overline{\tau} \Rightarrow f\,\texttt{[}\mathcal{S}(\rho'_1), \ldots, \mathcal{S}(\rho'_{2^n}), \rho''_1, \ldots, \rho''_k\ \texttt{]} \texttt{ at } \mathcal{S}(\rho) : (\tau, \rho), \epsilon, D_5, \mathcal{C}_2 \tag{E.77}$$

and by inversion

$$\mathrm{ML}(\tau') \geq^{\mathrm{ML}} \overline{\tau} \tag{E.78}$$
$$\Gamma(f) = ((\tau', \vec{m}), \rho') \tag{E.79}$$
$$(\tau, \rho, D_2) = \mathrm{spread}(\overline{\tau}, D_1) \tag{E.80}$$
$$n = \text{number of arrows in } \overline{\tau}' \tag{E.81}$$
$$(\vec{\rho}\,' = \rho'_1, \ldots, \rho'_{2^n}, D_3) = \mathrm{freshRegVar}(D_2) \tag{E.82}$$
$$(\vec{\epsilon}\,' = \epsilon'_1, \ldots, \epsilon'_{2^n}, D_4) = \mathrm{freshEffVar}(D_3) \tag{E.83}$$
$$(\epsilon, D_5) = \mathrm{freshEffVar}(D_4) \tag{E.84}$$
$$\mathcal{S} \text{ solves } \mathcal{C}_2 = \{\mathbf{c_1}, \mathbf{c_2}\} \cup \mathcal{C}_1 \tag{E.85}$$
$$\rho''_1, \ldots, \rho''_k = \mathit{actuals}_{\mathcal{S}}(\tau', \tau, \vec{m}) \setminus (\mathcal{S}(\rho'_1), \ldots, \mathcal{S}(\rho'_{2^n})) \tag{E.86}$$

with $\mathbf{c_1}, \mathbf{c_2}$ as in rule 16. Further by assumption

$$\Gamma \simeq_{\mathcal{S}} TE. \tag{E.87}$$

Let

$$(\underline{\sigma}, \underline{\rho'}) = TE(f). \tag{E.88}$$

Then by (E.87) and (E.79),

$$\underline{\rho'} = \mathcal{S}(\rho') \tag{E.89}$$

and

$$\underline{\sigma} \simeq_{\mathcal{S}} (\tau', \vec{m}). \tag{E.90}$$

Write $\underline{\sigma}$ as

$$\underline{\sigma} = \forall \vec{\alpha}\, \vec{\rho}\, \vec{\epsilon}.\mathcal{S}(\tau') \tag{E.91}$$

and let

$$\vec{\rho}'' = \mathrm{frv}(\mathcal{S}(\tau')) \setminus \mathrm{frv}(\mathcal{S}(\vec{m})) \tag{E.92}$$
$$\vec{\epsilon}'' = \mathrm{fev}(\mathcal{S}(\tau')) \setminus \mathrm{fev}(\mathcal{S}(\vec{m})). \tag{E.93}$$

By $\Vdash_{\mathcal{S}} \mathbf{c_1}$ and (E.90) there exists an $\underline{S}$ with

$$\mathrm{Dom}(\underline{S}) \subseteq \vec{\alpha}\,\vec{\rho}''\,\vec{\epsilon}'' \tag{E.94}$$

and

$$\vec{\rho}'' \subseteq \vec{\rho} \tag{E.95}$$
$$\vec{\epsilon}'' \subseteq \vec{\epsilon} \tag{E.96}$$

and

$$\underline{S}(\mathcal{S}(\tau')) = \mathcal{S}(\tau). \tag{E.97}$$

Let

$$\begin{aligned} \underline{S_1} = \{\rho \mapsto \mathcal{S}(\rho'_j) \mid \rho \in \vec{\rho} \setminus \vec{\rho}'' \wedge \mathit{index}(\rho, \mathcal{S}(\tau')) = j\} \\ \cup \{\epsilon \mapsto \mathcal{S}(\epsilon'_j) \mid \epsilon \in \vec{\epsilon} \setminus \vec{\epsilon}'' \wedge \mathit{index}(\epsilon, \mathcal{S}(\tau')) = j\} \end{aligned} \tag{E.98}$$

and let

$$\underline{S_2} = \underline{S} + \underline{S_1}. \tag{E.99}$$

Then by (E.94), (E.97), (E.98), and (E.99)

$$\underline{\sigma} \geq \mathcal{S}(\tau) \text{ via } \underline{S_2}. \tag{E.100}$$

Moreover, one can check that

$$\underline{S_2}(\vec{\rho}) = \mathcal{S}(\rho'_1), \ldots, \mathcal{S}(\rho'_{2^n}), \rho''_1, \ldots, \rho''_k \tag{E.101}$$

(there are two main cases: (i) $\rho \in \mathcal{S}(\tau')$ and (ii) $\rho \in \vec{\rho} \setminus \mathcal{S}(\tau')$ and for (i) there are again two cases: (i.i) primary and (i.ii) secondary variable).

Finally, by (E.80)

$$\mathrm{ML}(\mathcal{S}(\tau)) = \overline{\tau} \tag{E.102}$$

so by rule (3) on (E.88), (E.89), (E.91), (E.100), (E.101), and (E.102) we conclude

$$\begin{aligned} \mathit{TE} \vdash f : \overline{\tau} \Rightarrow\ & f\,\texttt{[}\mathcal{S}(\rho'_1), \ldots, \mathcal{S}(\rho'_{2^n}), \rho''_1, \ldots, \rho''_k\ \texttt{]}\ \texttt{at}\ \mathcal{S}(\rho) \\ & : (\mathcal{S}(\tau), \mathcal{S}(\rho)), \{\mathcal{S}(\rho'), \mathcal{S}(\rho)\} \end{aligned} \tag{E.103}$$

Let

$$\underline{\epsilon}.\underline{\varphi} = \mathcal{S}(\epsilon). \tag{E.104}$$

Then by $\Vdash_{\mathcal{S}} \mathbf{c_2}$ and (E.103) we can apply rule (8) to conclude

$$\begin{aligned} TE \vdash f : \overline{\tau} \Rightarrow\ & f\,\texttt{[}\mathcal{S}(\rho'_1), \ldots, \mathcal{S}(\rho'_{2^n}), \rho''_1, \ldots, \rho''_k\ \texttt{]}\ \texttt{at}\ \mathcal{S}(\rho) \\ & : (\mathcal{S}(\tau), \mathcal{S}(\rho)), \{\underline{\varphi}\} \end{aligned} \tag{E.105}$$

which together with (E.104) is the required, completing the proof of this case.

We have covered all cases, thus completing the proof of the theorem.

F Proof of Completeness of Constraint-based Formulation of Region Inference

In this section we prove the completeness of the constraint-based formulation of region inference with respect to the implementation oriented region inference rules. We first recall the theorem:

Theorem 70 (Completeness) *If*

(1) $\binom{A}{B}, TE \vdash e \Rightarrow e' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}$ *with rule 14 only applied immediately after rule 13 or 12;*
(2) $\Gamma \simeq_{\mathcal{S}_1} TE$*;*
(3) $\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_1)$*;*
(4) $\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_1)$*;*
(5) D_1 *is a simple basis;*
(6) $\mathrm{Dom}(D_1) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset$*;*
(7) $\binom{A'}{B} \uplus \binom{A}{B}$ *exists;*
(8) $\mathcal{S}_1(D_1) \subseteq A'$*;*
(9) $\mathcal{C}_1$ *is pre-well-tagged and* $\forall((\tau, \vec{m}), \rho) \in \mathrm{Rng}(\Gamma).\exists\, \mathrm{Norm}(\tau, \vec{m}) \in \mathcal{C}_1$*;*
(10) $\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_1)$*;*
(11) $\mathcal{S}_1\ \mathrm{solves}\ \mathcal{C}_1$

then there exist D_2*,* $\mathcal{S}_2$*,* $\mathcal{C}_2$*,* τ*,* ρ*, and* ϵ *such that*

(1) $\mathrm{Dom}(D_2) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset$*;*
(2) $\mathrm{Dom}(\mathcal{S}_2) \subseteq \mathrm{Dom}(D_2 \setminus D_1)$*;*
(3) $(\mathcal{S}_1 + \mathcal{S}_2)(D_2) \subseteq A' \cup A$*;*
(4) $\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_1 + \mathcal{S}_2} e \Rightarrow e' : (\tau, \rho), \epsilon, D_2, \mathcal{C}_2$*;*
(5) $(\mathcal{S}_1 + \mathcal{S}_2)(\tau) = \underline{\tau}$*;*
(6) $(\mathcal{S}_1 + \mathcal{S}_2)(\rho) = \underline{\rho}$*;*

(7) $(\mathcal{S}_1 + \mathcal{S}_2)(\epsilon) = \underline{\epsilon}.\underline{\varphi}$.

We shall make use of the following lemmas, whose proofs are straightforward.

Lemma 71

(1) *If* $\sigma \simeq_{\mathcal{S}_1} \underline{\sigma}$ *and* $\mathrm{frev}(\sigma) \cap \mathrm{Dom}(\mathcal{S}_2) = \emptyset$ *and* $\mathrm{Dom}(\mathcal{S}_2) \cap \mathit{TyVar} = \emptyset$*, then also* $\sigma \simeq_{\mathcal{S}_1+\mathcal{S}_2} \underline{\sigma}$.
(2) *If* $\Gamma \simeq_{\mathcal{S}_1} \mathit{TE}$ *and* $\mathrm{frev}(\Gamma) \cap \mathrm{Dom}(\mathcal{S}_2) = \emptyset$ *and* $\mathrm{Dom}(\mathcal{S}_2) \cap \mathit{TyVar} = \emptyset$*, then also* $\Gamma \simeq_{\mathcal{S}_1+\mathcal{S}_2} \mathit{TE}$.

Lemma 72 *If* $\mathcal{S}_1$ *solves* $\mathcal{C}$ *and* $\mathrm{frev}(\mathcal{C}) \subseteq \mathrm{Dom}(D_1)$ *and* $\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_1)$ *and* $D_1 \subseteq D_2$ *and* $\mathrm{Dom}(\mathcal{S}_2) \subseteq \mathrm{Dom}(D_2 \setminus D_1)$*, then* $\mathcal{S}_1 + \mathcal{S}_2$ *solves* $\mathcal{C}$.

Proof of Theorem 70

The proof is by rule induction on assumption (1), proceeding by cases on the last rule applied.

In the proof we shall abbreviate assumption (9) of the theorem to simply "$(\mathcal{C}, \Gamma)$ is pre-well-tagged."

Case (Rule (14) **preceded by rule** (13)**)** Then by assumption we have

$$\begin{aligned}\left(\begin{smallmatrix}A\\B\end{smallmatrix}\right), \mathit{TE} \vdash\; & \texttt{letrec}\ f : \forall\vec{\alpha}.\overline{\tau_0}\ \texttt{(}x : \overline{\tau_1}\texttt{)}\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\ & \texttt{letregion}\ \vec{\rho}\,'\ \texttt{in} \\ & \quad \texttt{letrec}\ f\texttt{[}\vec{\rho}\texttt{](}x\texttt{)}\ \texttt{at}\ \underline{\rho_0}\ \texttt{=}\ e_1'\ \texttt{in}\ e_2' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon}\end{aligned} \tag{F.1}$$

and by inversion of rules (14) and (13) we get

$$B \uplus B_1 \quad \text{exists} \tag{F.2}$$

$$\begin{aligned}\left(\begin{smallmatrix}A_4\\B\uplus B_1\end{smallmatrix}\right), \mathit{TE} \vdash\; & \texttt{letrec}\ f : \forall\vec{\alpha}.\overline{\tau_0}\ \texttt{(}x : \overline{\tau_1}\texttt{)}\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\ & \texttt{letrec}\ f\texttt{[}\vec{\rho}\texttt{](}x\texttt{)}\ \texttt{at}\ \underline{\rho_0}\ \texttt{=}\ e_1'\ \texttt{in}\ e_2' : (\underline{\tau}, \underline{\rho}), \underline{\varphi_4}, \underline{\epsilon_4}\end{aligned} \tag{F.3}$$

$$\underline{\varphi} = \mathrm{Observe}(\mathit{TE}, (\underline{\tau}, \underline{\rho}), \underline{\varphi_4}) \tag{F.4}$$

$$\vec{\rho}\,' = \mathrm{frv}(\underline{\varphi_4} \setminus \underline{\varphi}) \tag{F.5}$$

$$A = A_4 \uplus \{\underline{\epsilon}.\underline{\varphi}\} \quad \text{exists} \tag{F.6}$$

$$B \uplus B_1 \vdash \mathit{TE}, \underline{\varphi_1} \tag{F.7}$$

$$\underline{\sigma} = \forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau_0} \tag{F.8}$$

$$\underline{\hat{\sigma}} = \forall\vec{\rho}\,\vec{\epsilon}.\underline{\tau_0} \tag{F.9}$$

$$\mathrm{fv}(\vec{\alpha}\,) \cap \mathrm{ftv}(TE) = \emptyset \tag{F.10}$$
$$\mathrm{Dom}(B_2) = \vec{\rho} \cup \vec{\epsilon} \tag{F.11}$$
$$(B \uplus B_1) \uplus B_2 \quad \text{exists} \tag{F.12}$$
$$\left({}^{A_1}_{B\uplus B_1\uplus B_2}\right), TE + \{f \mapsto (\underline{\hat{\sigma}}, \rho_0)\} \vdash \lambda\, x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow \tag{F.13}$$
$$\lambda\, x.e_1' \ \mathtt{at}\ \underline{\rho_0} : (\underline{\tau_0}, \underline{\rho_0}), \underline{\varphi_1}, \underline{\epsilon_1}$$
$$\left({}^{A_2}_{B\uplus B_1}\right), TE + \{f \mapsto (\underline{\sigma}, \rho_0)\} \vdash e_2 \Rightarrow e_2' : (\underline{\tau}, \underline{\rho}), \underline{\varphi_2}, \underline{\epsilon_2} \tag{F.14}$$
$$\underline{\varphi_4} = \underline{\varphi_1} \cup \underline{\varphi_2} \tag{F.15}$$
$$\underline{\varphi_3} = \mathrm{frv}(\underline{\tau_0} \setminus \vec{\rho}\,) \cup \bigcup\{\{\epsilon\} \cup \underline{\varphi} \mid \epsilon \in (\mathrm{fev}(\underline{\tau_0}) \setminus \vec{\epsilon}\,) \wedge \epsilon.\underline{\varphi} = \widehat{\Phi \text{ of } B}(\epsilon)\} \tag{F.16}$$
$$\left({}^{A_3}_{B\uplus B_1}\right) = \left({}^{A_1}_{B\uplus B_1}\right) \uplus \left({}^{A_2}_{B\uplus B_1}\right) \tag{F.17}$$
$$A_4 = A_3 \uplus \{\underline{\epsilon_3}.\underline{\varphi_3}, \underline{\epsilon_4}.\underline{\varphi_4}\} \quad \text{exists} \tag{F.18}$$
$$\mathrm{ML}(\underline{\sigma}) = \forall\vec{\alpha}\,.\overline{\tau_0} \tag{F.19}$$
$$\mathrm{ML}(\underline{\tau}) = \overline{\tau} \tag{F.20}$$

Further by assumption we have

$$\Gamma \simeq_{\mathcal{S}_1} TE \tag{F.21}$$
$$\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_1) \tag{F.22}$$
$$\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.23}$$
$$D_1 \quad \text{is a simple basis} \tag{F.24}$$
$$Dom(D_1) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.25}$$
$$\left({}^{A'}_{B}\right) \uplus \left({}^{A}_{B}\right)\exists \tag{F.26}$$
$$\mathcal{S}_1(D_1) \subseteq A' \tag{F.27}$$
$$(\mathcal{C}_1, \Gamma) \quad \text{is pre-well-tagged} \tag{F.28}$$
$$\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.29}$$
$$\mathcal{S}_1 \text{ solves } \mathcal{C}_1 \tag{F.30}$$

Now let

$$(\tau_0, \rho_0, D_2) = \text{spread}(\overline{\tau_0}, D_1) \tag{F.31}$$
$$\text{s.t.} \quad \text{Dom}(D_2) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset$$
$$\vec{m_3} = \text{mono}(\Gamma) \tag{F.32}$$
$$\vec{\alpha_1} = \text{ftv}(\tau_0) \tag{F.33}$$
$$(\epsilon_{11}, \epsilon_3, D_3) = \text{freshEffVar}(D_2) \tag{F.34}$$
$$\text{s.t.} \quad \text{Dom}(D_3) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset$$
$$\vec{m_1} = \vec{\alpha_1}\, \rho_0 \epsilon_{11} \epsilon_3 \tag{F.35}$$
$$\mathcal{C}_2 = \mathcal{C}_1 \cup \{\text{Norm}(\tau_0, \vec{m_1}\, \vec{m_3}\,)\} \tag{F.36}$$
$$\mathcal{S}_2 = \text{mk\_}\mathcal{S}\left((\tau_0, \rho_0), (\underline{\tau_0}, \underline{\rho_0})\right) \tag{F.37}$$
$$\mathcal{S}_3 = \{\epsilon_{11} \mapsto \underline{\epsilon_1}.\underline{\varphi_1}, \epsilon_3 \mapsto \underline{\epsilon_3}.\underline{\varphi_3}\} \tag{F.38}$$
$$\mathcal{S}_4 = \mathcal{S}_2 + \mathcal{S}_3 \tag{F.39}$$
$$\mathcal{S}_5 = \mathcal{S}_1 + \mathcal{S}_4 \tag{F.40}$$
$$\vec{\alpha_2} = \text{ftv}(\tau_0) \setminus \vec{\alpha} \tag{F.41}$$
$$\vec{m_2} = \vec{\alpha_2}\, \rho_0 \epsilon_{11} \epsilon_3 \tag{F.42}$$
$$\vec{\alpha}'' = \text{ftv}(\mathcal{S}_5(\tau_0)) \setminus \text{ftv}(\mathcal{S}_5(\vec{m_2}\, \vec{m_3}\,)) \tag{F.43}$$
$$\vec{\rho}'' = \text{frv}(\mathcal{S}_5(\tau_0)) \setminus \text{frv}(\mathcal{S}_5(\vec{m_2}\, \vec{m_3}\,)) \tag{F.44}$$
$$\vec{\epsilon}'' = \text{fev}(\mathcal{S}_5(\tau_0)) \setminus \text{fev}(\mathcal{S}_5(\vec{m_2}\, \vec{m_3}\,)) \tag{F.45}$$

Claim (I)

$$\vec{\rho}'' \subseteq \vec{\rho} \tag{F.46}$$
$$\vec{\epsilon}'' \subseteq \vec{\epsilon} \tag{F.47}$$

Proof of Claim I

$\vec{\rho}'' = \text{frv}(\mathcal{S}_5(\tau_0)) \setminus \text{frv}(\mathcal{S}_5(\vec{m_2}\, \vec{m_3}\,))$	by (F.44)
$\subseteq \text{frv}(\mathcal{S}_5(\tau_0)) \setminus \text{frv}(\mathcal{S}_5(\vec{m_2}\,))$	obvious
$= \text{frv}(\mathcal{S}_5(\tau_0)) \setminus \text{frv}(\mathcal{S}_4(\vec{m_2}\,))$	by (F.37)–(F.40), (F.31), (F.34), (F.23)
$\subseteq \text{frv}(\mathcal{S}_5(\tau_0)) \setminus \text{frv}(\mathcal{S}_4(\epsilon_3))$	obvious using (F.42)
$= \text{frv}(\mathcal{S}_5(\tau_0)) \setminus (\text{frv}(\underline{\tau_0}) \setminus \vec{\rho}\,)$	by (F.37)–(F.40), (F.16) and consistency of $\underline{\sigma}$ in $B \uplus B_1$, which in particular means that $\underline{\sigma}$ is well-formed, hence $\text{frv}(\underline{\varphi_3}) = \text{frv}(\underline{\tau_0}) \setminus \vec{\rho}$
$= \text{frv}(\underline{\tau}) \setminus (\text{frv}(\underline{\tau}) \setminus \vec{\rho}\,)$	by (F.37)–(F.40)
$\subseteq \vec{\rho}\,'$	obvious

(F.47) is shown analogously. This completes the proof of Claim **I**.

Claim (II)

$$\mathcal{S}_5 \text{ solves Norm}(\tau_0, \vec{m_2}\,\vec{m_3}\,) \tag{F.48}$$

Proof of Claim II By (F.37)–(F.40), $\mathcal{S}_5(\tau_0) = \underline{\tau_0}$. Moreover, by Lemma 28 on (F.14), $B \uplus B_1 \vdash \underline{\sigma}$, so $\underline{\sigma}$ is consistent, hence in normal form. By definition of solution and by definition of normal form, one sees that (F.48) follows from Claim **I**. This completes the proof of Claim **II**.

Now, by (F.48), $\text{mk\_}\underline{\sigma}(\mathcal{S}_5, (\tau_0, \vec{m_2}\,\vec{m_3}\,))$ is well-defined and we can let

$$\underline{\sigma'} = \text{mk\_}\underline{\sigma}(\mathcal{S}_5, (\tau_0, \vec{m_2}\,, \vec{m_3}\,)) \tag{F.49}$$

Claim (III)

$$\underline{\sigma} = \underline{\sigma}' \tag{F.50}$$

Proof of Claim III By definition of equality, we only need to show equality up to renaming of bound variables so, noting that $\underline{\tau_0} = \mathcal{S}_5(\tau_0)$, it suffices to show

(i) $\vec{\alpha} = \vec{\alpha}''$,
(ii) $\vec{\rho} \cap \text{frv}(\underline{\tau_0}) = \vec{\rho}''$,
(iii) $\vec{\epsilon} \cap \text{fev}(\underline{\tau_0}) = \vec{\epsilon}''$,
(iv) $\vec{\rho} \cap \text{frv}(\mathcal{S}_5(\vec{m_2}\,\vec{m_3}\,)) = \emptyset$,
(v) $\vec{\epsilon} \cap \text{fev}(\mathcal{S}_5(\vec{m_2}\,\vec{m_3}\,)) = \emptyset$.

Case (i) First note that

$$\vec{\alpha} \cap \text{ftv}(\mathcal{S}_1(\vec{m_3}\,)) = \emptyset \tag{F.51}$$

because by (F.10), $\mathrm{fv}(\vec{\alpha}\,) \cap \mathrm{ftv}(TE) = \emptyset$ and by Lemma 36 on (F.21), $\mathrm{ftv}(\vec{m_3}\,) \subseteq \mathrm{ftv}(TE)$ (and $\mathrm{ftv}(\vec{m_3}\,) = \mathrm{ftv}(\mathcal{S}_1(\vec{m_3}\,)))$. Then calculate

$$\begin{aligned}
\vec{\alpha}'' &= \mathrm{ftv}(\mathcal{S}_5(\tau_0)) \setminus \mathrm{ftv}(\mathcal{S}_5(\vec{m_2}\,\vec{m_3}\,)) && \text{by (F.43)}\\
&= \mathrm{ftv}(\underline{\tau_0}) \setminus \mathrm{ftv}(\mathcal{S}_5(\vec{m_2}\,\vec{m_3}\,)) &&\\
&= \Big(\mathrm{ftv}(\underline{\tau_0}) \setminus \mathrm{ftv}(\mathcal{S}_4(\vec{m_2}\,))\Big) \setminus \mathrm{ftv}(\mathcal{S}_1(\vec{m_3}\,)) &&\\
&= \Big(\mathrm{ftv}(\underline{\tau_0}) \setminus \vec{\alpha_2}\,\Big) \setminus \mathrm{ftv}(\mathcal{S}_1(\vec{m_3}\,)) &&\\
&= \Big(\mathrm{ftv}(\underline{\tau_0}) \setminus (\mathrm{ftv}(\underline{\tau_0}) \setminus \vec{\alpha}\,)\Big) \setminus \mathrm{ftv}(\mathcal{S}_1(\vec{m_3}\,)) && \text{by (F.41)}\\
&= \vec{\alpha} \setminus \mathrm{ftv}(\mathcal{S}_1(\vec{m_3}\,)) && \text{using that } \vec{\alpha} \subseteq \mathrm{ftv}(\underline{\tau_0})\\
&= \vec{\alpha} && \text{by (F.51).}
\end{aligned}$$

This completes the proof of Case (**i**).

Case (ii) The $\supseteq$ direction follows by (F.44) and (F.46). For the $\subseteq$ direction, note first that $\vec{\rho} \cap \mathrm{frv}(\underline{\tau_0}) \subseteq \mathrm{Dom}(B_2)$ by (F.11), and that by (F.13), (F.17), (F.18), (F.16), and (F.26) $\mathrm{Dom}(B_2) \cap \mathrm{frv}(A') = \emptyset$. Hence by (F.27) and (F.23), $\mathrm{Dom}(B_2) \cap \mathrm{Yield}(\mathcal{S}_1) = \emptyset$. Thus it suffices to show that $\Big(\vec{\rho} \cap \mathrm{frv}(\underline{\tau_0})\Big) \cap \mathrm{frv}(\mathcal{S}_5(\vec{m_2}\,)) = \emptyset$. Since by Lemma 28 on (F.14) $\underline{\rho_0} \in \mathrm{Dom}(B \uplus B_1)$ we get by (F.12) that $\underline{\rho_0} \notin \mathrm{Dom}(B_2)$, so $\underline{\rho_0} \notin \vec{\rho}$. Thus it suffices to show that $\Big(\vec{\rho} \cap \mathrm{frv}(\underline{\tau_0})\Big) \cap \mathrm{frv}(\mathcal{S}_5(\epsilon_{11}, \epsilon_3)) = \emptyset$. But $\mathcal{S}_5(\epsilon_{11}) = \underline{\epsilon_1}.\underline{\varphi_1}$ and by (F.7) and (F.12) $\mathrm{frv}(\underline{\epsilon_1}.\underline{\varphi_1}) \cap \mathrm{Dom}(B_2) = \emptyset$ so then it suffices to show that $\Big(\vec{\rho} \cap \mathrm{frv}(\underline{\tau_0})\Big) \cap \mathrm{frv}(\mathcal{S}_5(\epsilon_3)) = \emptyset$. But this is immediate by the definition of $\mathcal{S}_5$ ((F.37)–(F.40)) and again using (as in the proof of Case (**i**)) that $\mathrm{frv}(\underline{\varphi_3}) = \mathrm{frv}(\underline{\tau_0}) \setminus \vec{\rho}\,)$. This completes the proof of Case (**ii**).

Case (iii) Analogous to Case (**ii**) (one also uses that, by Lemma 28, $\underline{\epsilon_1} \notin \mathrm{Dom}(B_2)$).

Case (iv) By Case (**ii**) and (F.44), it suffices to show that $\Big(\vec{\rho} \setminus \mathrm{frv}(\underline{\tau_0})\Big) \cap \mathrm{frv}(\mathcal{S}_5(\vec{m_2}\,\vec{m_3}\,)) = \emptyset$. By definition of $\mathcal{S}_5$ ((F.37)–(F.40)), it suffices to show that $\Big(\vec{\rho} \setminus \mathrm{frv}(\underline{\tau_0})\Big) \cap \Big(\{\underline{\rho_0}\} \cup \mathrm{frv}(\underline{\varphi_1}) \cup \mathrm{frv}(\mathcal{S}_1(\vec{m_3}\,))\Big) = \emptyset$. But since $\vec{\rho} = \mathrm{frv}(\mathrm{Dom}(B_2))$ by (F.11), it suffices to show that $\mathrm{frv}(\mathrm{Dom}(B_2)) \cap \Big(\{\underline{\rho_0}\} \cup \mathrm{frv}(\underline{\varphi_1}) \cup \mathrm{frv}(\mathcal{S}_1(\vec{m_3}\,))\Big) = \emptyset$. This follows by an argument analogous to the one given in the proof of Case (**ii**). This completes the proof of Case (**iv**).

Case (v) Analogous to Case (**iv**) (also need to see that $\underline{\epsilon_1}, \underline{\epsilon_2} \notin \mathrm{Dom}(B_2)$, but that is easy).

This completes the proof of Claim **III**.

By (F.49) and (F.50) we get (easy to see) that

$$(\tau_0, \vec{m}_2\,\vec{m}_3\,) \simeq_{\mathcal{S}_5} \underline{\sigma}. \tag{F.52}$$

In a very similar way (only different for type variables) we get

$$\underline{\hat{\sigma}} = \mathrm{mk\_}\underline{\sigma}(\mathcal{S}_5, (\tau_0, \vec{m}_1\,\vec{m}_3\,)) \tag{F.53}$$

and

$$(\tau_0, \vec{m}_1\,\vec{m}_3\,) \simeq_{\mathcal{S}_5} \underline{\hat{\sigma}} \tag{F.54}$$

and

$$\mathcal{S}_5 \text{ solves } \mathrm{Norm}(\tau_0, \vec{m}_1\,\vec{m}_3\,). \tag{F.55}$$

By Lemma 71 on (F.21), (F.37)–(F.40), (F.31), and (F.34) we get that

$$\Gamma \simeq_{\mathcal{S}_5} TE \tag{F.56}$$

so by (F.56), (F.54), (F.37)–(F.40) we get that

$$\Gamma + \{f \mapsto ((\tau_0, \vec{m}_1\,\vec{m}_3\,), \rho_0)\} \simeq_{\mathcal{S}_5} TE + \{f \mapsto (\underline{\hat{\sigma}}, \underline{\rho_0})\}. \tag{F.57}$$

By (F.2), (F.6), (F.12), (F.13), (F.16), (F.17), and (F.18)

$$A_1' \stackrel{\text{def}}{=} A' \uplus B_1 \uplus B2 \uplus \{\underline{\epsilon_1}.\underline{\varphi_1}, \underline{\epsilon_3}.\underline{\varphi_3}\} \quad \text{exists} \tag{F.58}$$

and

$$\left(\begin{smallmatrix} A_1' \\ B\uplus B_1 \uplus B_2 \end{smallmatrix}\right) \text{ is a cone }. \tag{F.59}$$

Moreover, by (F.58),(F.13), (F.17), (F.18), (F.6), and (F.26)

$$\left(\begin{smallmatrix} A_1' \\ B\uplus B_1 \uplus B_2 \end{smallmatrix}\right) \uplus \left(\begin{smallmatrix} A_1 \\ B\uplus B_1 \uplus B_2 \end{smallmatrix}\right) \quad \text{exists.} \tag{F.60}$$

Furthermore,

$$\mathcal{S}_5(D_3) \subseteq A_1' \tag{F.61}$$

by (F.27), (F.37)–(F.38), (F.31), (F.34), and (F.13) (which by Lemma 28 gives $B \uplus B_1 \uplus B_2 \vdash (\underline{\tau_0}, \underline{\rho_0}), \underline{\varphi_1})$. Let

$$\mathcal{C}_2 = \mathcal{C}_1 \cup \{\mathrm{Norm}(\tau_0, \vec{m}_1\,\vec{m}_3\,)\}. \tag{F.62}$$

By (F.22), (F.31), (F.34), and (F.35) we see that

$$\mathrm{frev}(\Gamma + \{f \mapsto ((\tau_0, \vec{m}_1\,\vec{m}_3\,), \rho_0)\}) \subseteq \mathrm{Dom}(D_3). \tag{F.63}$$

Also, by (F.23), (F.31), (F.34), and (F.37)–(F.40), we have

$$\mathrm{Dom}(\mathcal{S}_5) \subseteq \mathrm{Dom}(D_3) \tag{F.64}$$

and

$$D_3 \quad \text{is a simple basis} \tag{F.65}$$
$$\mathrm{Dom}(D_3) \cap \mathrm{Dom}(A'_1 \cup A_1) = \emptyset. \tag{F.66}$$

Clearly, by (F.62), (F.28), (F.55), (F.31), and (F.34)

$$(\mathcal{C}_2, \Gamma + \{f \mapsto ((\tau_0, \vec{m}_1\, \vec{m}_3\,), \rho_0)\}) \quad \text{is pre-well-tagged} \tag{F.67}$$

and

$$\mathcal{S}_5 \;\text{solves}\; \mathcal{C}_2 \tag{F.68}$$

and

$$\mathrm{frev}(\mathcal{C}_2) \subseteq \mathrm{Dom}(D_3). \tag{F.69}$$

Let

$$A''_1 = A' \cup A \cup A''. \tag{F.70}$$

Then we can apply induction on (F.13), (F.34), (F.57), (F.63), (F.64), (F.65), (F.66), (F.60), (F.61), (F.67), (F.69), (F.68), and (F.70) to get that there exist D_4, $\mathcal{S}_6$, $\mathcal{C}_3$, τ_1, ρ_1, and ϵ_1 such that

$$Dom(D_4) \cap \mathrm{Dom}(A'_1 \cup A_1 \cup A''_1) = \emptyset \tag{F.71}$$
$$Dom(\mathcal{S}_6) \subseteq \mathrm{Dom}(D_4 \setminus D_3) \tag{F.72}$$
$$(\mathcal{S}_5 + \mathcal{S}_6)(D_4) \subseteq A'_1 \cup A_1 \tag{F.73}$$

$$\mathcal{C}_2, D_3, \Gamma + \{f \mapsto ((\tau_0, \vec{m}_1\, \vec{m}_3\,), \rho_0)\} \Vdash_{\mathcal{S}_5+\mathcal{S}_6} \lambda\, x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow$$
$$\lambda\, x.e'_1 \;\texttt{at}\; \underline{\rho_0} : (\tau_1, \rho_1), \epsilon_1, D_4, \mathcal{C}_3 \tag{F.74}$$

$$(\mathcal{S}_5 + \mathcal{S}_6)(\tau_1) = \underline{\tau_0} \tag{F.75}$$
$$(\mathcal{S}_5 + \mathcal{S}_6)(\rho_1) = \underline{\rho_0} \tag{F.76}$$
$$(\mathcal{S}_5 + \mathcal{S}_6)(\epsilon_1) = \underline{\epsilon_1}.\underline{\varphi_1}. \tag{F.77}$$

By Lemma 54 on (F.74), (F.67), (F.68), (F.69), and (F.63) we also have that

$$D_4 \quad \text{is a simple basis} \tag{F.78}$$
$$D_3 \subseteq D_4 \tag{F.79}$$
$$\mathcal{C}_2 \subseteq \mathcal{C}_3 \tag{F.80}$$
$$(\mathcal{C}_3, \Gamma + \{f \mapsto ((\tau_0, \vec{m}_1\, \vec{m}_3\,), \rho_0)\}) \quad \text{is pre-well-tagged} \tag{F.81}$$
$$\mathcal{S}_5 + \mathcal{S}_6 \text{ solves } \mathcal{C}_3 \tag{F.82}$$
$$\text{frev}(\mathcal{C}_3) \subseteq \text{Dom}(D_4). \tag{F.83}$$

Let

$$\mathcal{S}_7 = \mathcal{S}_5 + \mathcal{S}_6 \tag{F.84}$$

and let

$$\mathcal{C}_4 = \mathcal{C}_3 \cup \{\text{Norm}(\tau_0, \vec{m}_2\, \vec{m}_3\,)\}. \tag{F.85}$$

By (F.72), (F.84), (F.64), (F.52), (F.56), (F.37)–(F.40) and Lemma 71 we get

$$\Gamma + \{f \mapsto ((\tau_0, \vec{m}_2\, \vec{m}_3\,), \rho_0)\} \simeq_{\mathcal{S}_7} TE + \{f \mapsto (\underline{\sigma}, \underline{\rho_0})\}. \tag{F.86}$$

Moreover, by (F.63), (F.35),(F.42), and (F.79) we have

$$\text{frev}(\Gamma + \{f \mapsto ((\tau_0, \vec{m}_2\, \vec{m}_3\,), \rho_0)\}) \subseteq \text{Dom}(D_4). \tag{F.87}$$

Now let

$$A_2' = A_1' \cup A_1. \tag{F.88}$$

Then by (F.60)

$$\binom{A_2'}{B \uplus B_1} \text{ is a cone} \tag{F.89}$$

and by (F.60), (F.17), (F.18), (F.6), and (F.26) we get that

$$\binom{A_2'}{B \uplus B_1} \uplus \binom{A_2}{B \uplus B_1} \quad \text{exists.} \tag{F.90}$$

By (F.73), (F.84), (F.61), and (F.88), we have

$$\mathcal{S}_7(D_4) \subseteq A_2'. \tag{F.91}$$

Moreover, by (F.81) and (F.85),

$$(\mathcal{C}_4, \Gamma + \{f \mapsto ((\tau_0, \vec{m}_2\, \vec{m}_3\,), \rho_0)\}) \quad \text{is pre-well-tagged.} \tag{F.92}$$

Also, by (F.83), (F.85), (F.42), (F.31), (F.34), and (F.79)

$$\text{frev}(\mathcal{C}_4) \subseteq \text{Dom}(D_4) \tag{F.93}$$

and by (F.82), (F.84), (F.48), and (F.72)

$$\mathcal{S}_7 \text{ solves } \mathcal{C}_4. \tag{F.94}$$

Let

$$A_2'' = A' \cup A \cup A''. \tag{F.95}$$

By induction on (F.14); (F.86); (F.84), (F.72), (F.79), (F.64); (F.78) (F.71); (F.90); (F.91); (F.92); (F.93); and (F.94) there exist D_5, $\mathcal{S}_8$, $\mathcal{C}_5$, τ_2, ρ_2 and ϵ_2 such that

$$\mathrm{Dom}(D_5) \cap \mathrm{Dom}(A_2' \cup A_2 \cup A_2'') = \emptyset \tag{F.96}$$
$$\mathrm{Dom}(\mathcal{S}_8) \subseteq \mathrm{Dom}(D_5 \setminus D_4) \tag{F.97}$$
$$(\mathcal{S}_7 + \mathcal{S}_8)(D_5) \subseteq A_2' \cup A_2 \tag{F.98}$$
$$\mathcal{C}_4, D_4, \Gamma + \{f \mapsto ((\tau_0, \vec{m}_2\, \vec{m}_3\,), \rho_0)\} \Vdash_{\mathcal{S}_7+\mathcal{S}_8} e_2 \Rightarrow e_2' : (\tau_2, \rho_2), \epsilon_2, D_5, \mathcal{C}_5 \tag{F.99}$$
$$(\mathcal{S}_7 + \mathcal{S}_8)(\tau_2) = \underline{\tau} \tag{F.100}$$
$$(\mathcal{S}_7 + \mathcal{S}_8)(\rho_2) = \underline{\rho} \tag{F.101}$$
$$(\mathcal{S}_7 + \mathcal{S}_8)(\epsilon_2) = \underline{\epsilon_2}.\underline{\varphi_2}. \tag{F.102}$$

By Lemma 54 on (F.99), (F.92), (F.93), (F.94), (F.87), and (F.78) we also have that

$$D_5 \quad \text{is a simple basis} \tag{F.103}$$
$$D_4 \subseteq D_5 \tag{F.104}$$
$$\mathcal{C}_4 \subseteq \mathcal{C}_5 \tag{F.105}$$
$$(\mathcal{C}_5, \Gamma + \{f \mapsto ((\tau_0, \vec{m}_2\, \vec{m}_3\,), \rho_0)\}) \quad \text{is pre-well-tagged} \tag{F.106}$$
$$\mathcal{S}_7 + \mathcal{S}_8 \text{ solves } \mathcal{C}_5 \tag{F.107}$$
$$\mathrm{frev}(\mathcal{C}_5) \subseteq \mathrm{Dom}(D_5). \tag{F.108}$$

Let

$$\mathcal{S}_9 = \mathcal{S}_7 + \mathcal{S}_8. \tag{F.109}$$

By (F.109), (F.97), (F.104), (F.79), and (F.72) we get

$$\mathcal{S}_9 \restriction (\mathrm{Dom}(D_4)) = \mathcal{S}_7 \tag{F.110}$$
$$\mathcal{S}_9 \restriction (\mathrm{Dom}(D_3)) = \mathcal{S}_5 \tag{F.111}$$
$$\mathcal{S}_9 \restriction (\mathrm{Dom}(D_1)) = \mathcal{S}_1. \tag{F.112}$$

Let

$$(\epsilon_4, \epsilon, D_6) = \mathrm{freshEffVar}(D_5) \quad \text{s.t.} \quad \mathrm{Dom}(D_6) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset. \tag{F.113}$$

(This is possible by (F.96) and (F.95). Note that the use of A''_1, A''_2 and A'' ensures, e.g., that D_5 does not contain $\underline{\epsilon}$, i.e., ensures that syntactic and semantic variables are kept distinct.) Define

$$\mathcal{S}_{10} = \{\epsilon_4 \mapsto \underline{\epsilon_4}.\underline{\varphi_4}, \epsilon \mapsto \underline{\epsilon}.\underline{\varphi}\}. \tag{F.114}$$

Further let

$$\mathcal{S}_{11} = \mathcal{S}_4 + \mathcal{S}_6 + \mathcal{S}_8 + \mathcal{S}_{10} \tag{F.115}$$

and

$$\mathcal{S}_{12} = \mathcal{S}_1 + \mathcal{S}_{11} = \mathcal{S}_9 + \mathcal{S}_{10}. \tag{F.116}$$

Note that, by (F.97), (F.114), (F.113), (F.72), (F.37), (F.38), (F.39), (F.41), and (F.34)

$$\mathrm{Dom}(\mathcal{S}_{11}) \subseteq \mathrm{Dom}(D_6 \setminus D_1). \tag{F.117}$$

Also

$$\mathcal{S}_{12}(D_6) \subseteq A' \cup A \tag{F.118}$$

by (F.114), (F.116), and (F.98).

Moreover, by Lemma 55 on (F.74); (F.69); (F.64), (F.72); and (F.117) (with $\mathcal{S}_1 = \mathcal{S}_5 + \mathcal{S}_6$) and $\mathcal{S}_2 = \mathcal{S}_8 + \mathcal{S}_{10}$ in the lemma) and noting that $\mathcal{S}_5 + \mathcal{S}_6 + \mathcal{S}_8 + \mathcal{S}_{10} = \mathcal{S}_{12}$ we get

$$\begin{aligned}\mathcal{C}_2, D_3, \Gamma + f \mapsto ((\tau, \vec{m}_1\, \vec{m}_3\,), \rho_0) \Vdash_{\mathcal{S}_{12}} \ &\lambda\, x : \overline{\tau_1}.e_1 : \overline{\tau_0} \Rightarrow \\ &\lambda\, x : e_1' \ \texttt{at} \ \underline{\rho_0} : (\tau_1, \rho_1), \epsilon_1, D_4, \mathcal{C}_3\end{aligned} \tag{F.119}$$

Also by Lemma 55 on (F.99); (F.93); (F.97), (F.84), (F.72), (F.79), (F.64); and (F.117) (with $\mathcal{S}_1 = \mathcal{S}_7 + \mathcal{S}_8$ and $\mathcal{S}_2 = \mathcal{S}_{10}$ in the lemma) and noting that $\mathcal{S}_7 + \mathcal{S}_8 + \mathcal{S}_{10} = \mathcal{S}_{12}$ we get

$$\mathcal{C}_4, D_4, \Gamma + \{f \mapsto ((\tau_0, \vec{m}_2\, \vec{m}_3\,), \rho_0)\} \Vdash_{\mathcal{S}_{12}} e_2 \Rightarrow e_2' : (\tau_2, \rho_2), \epsilon_2, D_5, \mathcal{C}_5 \tag{F.120}$$

Further, by (F.37), (F.75), (F.76), and by definition of $\mathcal{S}_{12}$,

$$\mathcal{S}_{12}(\tau_0, \rho_0) = (\underline{\tau_0}, \underline{\rho_0}) = \mathcal{S}_{12}(\tau_1, \rho_1). \tag{F.121}$$

Also, by (F.15), (F.77), (F.102), (F.114), and by definition of $\mathcal{S}_{12}$

$$\mathcal{S}_{12} \text{ satisfies } \epsilon_4 \mathrel{\dot{\supseteq}} \epsilon_1 \epsilon_2. \tag{F.122}$$

(to be really pedantic we should write, in one go, that $\mathcal{S}_{12}$solves$\{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}\} \cup \mathcal{C}_5$ (this we conclude in (F.130) below) because "solves/satisfies" is only defined for pre-well-tagged constraint sets — but this should not cause any confusion). By (F.21), (F.22), (F.23), (F.117) we get

$$\Gamma \simeq_{\mathcal{S}_{12}} TE \tag{F.123}$$

and by (F.100), (F.101), (F.114), and the definition of $\mathcal{S}_{12}$

$$\mathcal{S}_{12}(\tau_2) = \underline{\tau} \tag{F.124}$$

$$\mathcal{S}_{12}(\rho_2) = \underline{\rho} \tag{F.125}$$

$$\mathcal{S}_{12}(\epsilon_2) = \underline{\epsilon}.\underline{\varphi} \tag{F.126}$$

so by (F.123)–(F.126), (F.114), and (F.4)

$$\mathcal{S}_{12} \text{ satisfies } \epsilon \mathrel{\dot{\supseteq}} \mathrm{Obs}(\Gamma, (\tau_2, \rho_2), \epsilon_4). \tag{F.127}$$

Moreover, by (F.38), (F.77), and the definition of $\mathcal{S}_12$ we get

$$\mathcal{S}_{12} \text{ satisfies } \epsilon_{11} \doteq \epsilon_1. \tag{F.128}$$

Further, by (F.107), (F.108), (F.113), and the definition of $\mathcal{S}_{12}$ we get by Lemma 72 that

$$\mathcal{S}_{12} \text{ solves } \mathcal{C}_5 \tag{F.129}$$

and thus by (F.121), (F.122), (F.127), (F.128), and (F.129) we get

$$\mathcal{S}_{12} \text{ solves } \{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}\} \cup \mathcal{C}_5. \tag{F.130}$$

By Lemma 71 on (F.49), (F.50), and the fact that $\mathrm{Dom}(\mathcal{S}_{12}) \cap \mathrm{TyVar} = \emptyset$ we get that

$$\underline{\sigma} = \mathrm{mk}\_\underline{\sigma}(\mathcal{S}_{12}, (\tau_0, \vec{m}_2\, \vec{m}_3)). \tag{F.131}$$

By rule (19) on (F.31), (F.32), (F.33), (F.34), (F.35), (F.36), (F.119), (F.41), (F.42), (F.120), (F.130), (F.114), (F.126), (F.5), and (F.131) we get, using that $\mathcal{S}_{12}(\rho_0) = \underline{\rho_0}$,

$$\begin{aligned} \mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_{12}} & \ \texttt{letrec}\ f : \forall\vec{\alpha}.\overline{\tau_0}\ \texttt{(}x : \overline{\tau_1}\texttt{)}\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \overline{\tau} \Rightarrow \\ & \texttt{letregion}\ \vec{\rho}\,'\ \texttt{in letrec}\ f\texttt{[}\vec{\rho}\texttt{](}x\texttt{)}\ \texttt{at}\ \underline{\rho_0}\ \texttt{=}\ e_1'\ \texttt{in}\ e_2' : \\ & (\tau_2, \rho_2), \epsilon, D_6, \mathcal{C}_6 \end{aligned} \tag{F.132}$$

By (F.113), (F.117), (F.132), (F.124), (F.125), (F.126) we have the required, completing the proof of this case.

Case (Rule (13)**)** The proof for this case is essentially just a part of the proof for the preceding case.

Case (Rule (9)**)** Then

$$\binom{A}{B}, TE \vdash x : \overline{\tau} \Rightarrow x : (\underline{\tau}, \underline{\rho}), \emptyset, \underline{\epsilon} \tag{F.133}$$

and by inversion

$$TE(x) = (\underline{\tau}, \underline{\rho}) \tag{F.134}$$
$$\mathrm{ML}(\underline{\tau}) = \overline{\tau} \tag{F.135}$$
$$B \vdash (TE, (\underline{\tau}, \underline{\rho})) \tag{F.136}$$
$$A = B \uplus \{\underline{\epsilon}.\emptyset\} \quad \text{exists.} \tag{F.137}$$

Further by assumption

$$\Gamma \simeq_{\mathcal{S}_1} TE \tag{F.138}$$
$$\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_1) \tag{F.139}$$
$$\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.140}$$
$$D_1 \quad \text{is a simple basis} \tag{F.141}$$
$$\mathrm{Dom}(D_1) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.142}$$
$$\binom{A'}{B} \uplus \binom{A}{B} \quad \text{exists} \tag{F.143}$$
$$\mathcal{S}_1(D_1) \subseteq A' \tag{F.144}$$
$$(\mathcal{C}_1, \Gamma) \quad \text{is pre-well-tagged} \tag{F.145}$$
$$\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.146}$$
$$\mathcal{S}_1 \text{ solves } \mathcal{C}_1. \tag{F.147}$$

Let

$$(\tau, \rho) = \Gamma(x) \tag{F.148}$$
$$(\epsilon, D_2) = \mathrm{freshEffVar}(D_1) \quad \text{s.t.} \quad \mathrm{Dom}(D_2) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.149}$$
$$\mathcal{S}_2 = \{\epsilon \mapsto \underline{\epsilon}.\emptyset\}. \tag{F.150}$$

Then

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_1 + \mathcal{S}_2} x : \overline{\tau} \Rightarrow x : (\tau, \rho), \epsilon, D_2, \mathcal{C}_3$$

and it is easy to see that the remaining required items all also hold, completing the proof of this case.

Case (Rule (10)**)** Then

$$\binom{A}{B}, TE \vdash f : \overline{\tau} \Rightarrow f\,\texttt{[}\underline{S}(\underline{\rho}_1), \ldots, \underline{S}(\underline{\rho}_k)\texttt{]}\ \texttt{at}\ \underline{\rho} : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon} \tag{F.151}$$

and by inversion

$$TE(f) = (\underline{\sigma}, \underline{\rho'}) \tag{F.152}$$
$$\underline{\sigma} = \forall\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}.\underline{\tau'} \qquad \vec{\rho} = \underline{\rho}_1 \cdots \underline{\rho}_k \qquad \vec{\epsilon} = eps_1 \cdots \epsilon_m \tag{F.153}$$
$$\underline{\sigma} \geq \underline{\tau} \text{ via } \underline{S} \tag{F.154}$$
$$\mathrm{ML}(\underline{\tau}) = \overline{\tau} \tag{F.155}$$
$$B \vdash (TE, (\underline{\tau}, \underline{\rho})) \tag{F.156}$$
$$\underline{\varphi} = \{\underline{\rho'}, \underline{\rho}\} \tag{F.157}$$
$$A_1 = \Big(\{\underline{S}(\rho) \mid \rho \in \vec{\rho} \setminus \mathrm{frv}(\underline{\tau'})\}, \{\underline{S}(\epsilon) \mid \epsilon \in \vec{\epsilon} \setminus \mathrm{fev}(\underline{\tau'})\}\Big) \tag{F.158}$$
$$A_2 = B \uplus A_1 \quad \text{exists} \tag{F.159}$$
$$A = A_2 \uplus \{\underline{\epsilon}.\underline{\varphi}\} \quad \text{exists.} \tag{F.160}$$

Further by assumption

$$\Gamma \simeq_{\mathcal{S}_1} TE \tag{F.161}$$
$$\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_1) \tag{F.162}$$
$$\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.163}$$
$$D_1 \quad \text{is a simple basis} \tag{F.164}$$
$$\mathrm{Dom}(D_1) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.165}$$
$$\left(\begin{smallmatrix} A' \\ B \end{smallmatrix}\right) \uplus \left(\begin{smallmatrix} A \\ B \end{smallmatrix}\right) \quad \text{exists} \tag{F.166}$$
$$\mathcal{S}_1(D_1) \subseteq A' \tag{F.167}$$
$$(\mathcal{C}_1, \Gamma) \quad \text{is pre-well-tagged} \tag{F.168}$$
$$\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.169}$$
$$\mathcal{S}_1 \text{ solves } \mathcal{C}_1. \tag{F.170}$$

Now let

$$((\tau', \vec{m_1}), \rho') = \Gamma(f) \tag{F.171}$$
$$(\tau, \rho, D_2) = \mathrm{spread}(\overline{\tau}, D_1) \quad \text{s.t.} \quad \mathrm{Dom}(D_2) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.172}$$
$$n = \text{number of arrows in } \overline{\tau}' \tag{F.173}$$
$$\begin{aligned}(\vec{\rho}\,' = \rho'_1, \ldots, \rho'_{2^n}, D_3) &= \mathrm{freshRegVar}(D_2) \\ \text{s.t.} \quad \mathrm{Dom}(D_3) &\cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset\end{aligned} \tag{F.174}$$
$$\begin{aligned}(\vec{\epsilon}\,' = \epsilon'_1, \ldots, \epsilon'_{2^n}, D_4) &= \mathrm{freshEffVar}(D_3) \\ \text{s.t.} \quad \mathrm{Dom}(D_4) &\cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset\end{aligned} \tag{F.175}$$
$$\begin{aligned}(\epsilon, D_5) &= \mathrm{freshRegVar}(D_4) \\ \text{s.t.} \quad \mathrm{Dom}(D_5) &\cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset.\end{aligned} \tag{F.176}$$

Further let

$$\mathcal{S}_2 = \text{mk\_}\mathcal{S}\left((\tau,\rho),(\underline{\tau},\underline{\rho})\right) \tag{F.177}$$

(well-defined by (F.155) and (F.172)) and let

$$\mathcal{S}_3 = \{\rho'_j \mapsto \underline{S}(\underline{\rho_j}) \mid j \in 1..2^n\} \cup \{\epsilon'_j \mapsto \underline{S}(\epsilon_j) \mid j \in 1..2^n\} \tag{F.178}$$

(well-defined as, by (F.156) and (F.152), $\underline{\sigma}$ is consistent in B, hence in normal form, hence $k \geq 2^n$ with n as in (F.173) using (F.161) and (F.152) which gives that the number of arrows in $\underline{\tau'}$ is n). Moreover, let

$$\mathcal{S}_4 = \{\epsilon \mapsto \underline{\epsilon}.\underline{\varphi}\} \tag{F.179}$$

$$\mathcal{S}_5 = \mathcal{S}_2 + \mathcal{S}_3 + \mathcal{S}_4 \tag{F.180}$$

$$\mathcal{S}_6 = \mathcal{S}_2 + \mathcal{S}_5. \tag{F.181}$$

Claim (I)

$$\mathcal{S}_6 \text{ solves } \mathcal{C}_2 = \{(\tau',\vec{m}) \stackrel{\cdot}{\succeq} (\vec{\rho}\,' \cup \text{frv}(\tau), \vec{\epsilon}\,' \cup \text{fev}(\tau), \tau), \epsilon \stackrel{\cdot}{\supseteq} \rho'\rho\} \cup \mathcal{C}_1 \tag{F.182}$$

Proof of Claim I First note that $\mathcal{S}_6$ solves $\mathcal{C}_1$ by Lemma 72 on (F.169) and the definition of $\mathcal{S}_6$ ($\mathcal{S}_6 \restriction \text{Dom}(D_1) = \mathcal{S}_1$). Next, $\mathcal{S}_6$ solves $\epsilon \stackrel{\cdot}{\supseteq} \rho'\rho$ by (F.157) and the definition of $\mathcal{S}_6$. It remains to show that $\mathcal{S}_6$ solves $(\tau',\vec{m}) \stackrel{\cdot}{\succeq} (\vec{\rho}\,' \cup \text{frv}(\tau), \vec{\epsilon}\,' \cup \text{fev}(\tau), \tau)$. Let $\underline{S'} = \underline{S} \restriction \vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}$. This $\underline{S'}$ is a *finite* substition with domain included in $\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}$. Now extend it to have exactly domain $\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}$; call the resulting finite substitution $\underline{S''}$. We show that $(\mathcal{S}_6, (\underline{S''}))$ satisfy $(\tau',\vec{m}) \stackrel{\cdot}{\succeq} (\vec{\rho}\,' \cup \text{frv}(\tau), \vec{\epsilon}\,' \cup \text{fev}(\tau), \tau)$ from which the required is immediate. So we are to show that the four requirements of item (8) in Definition 49 are all met. First we show that $(\mathcal{S}_6, (\underline{S''}))$ satisfy $m \stackrel{\cdot}{\succeq}^1 m$, for all $m \in \vec{m}$. Recall that $\text{Dom}(\underline{S''}) = \vec{\alpha}\,\vec{\rho}\,\vec{\epsilon}$. Further, by Lemma 71 on (F.161), (F.162), and the definition of $\mathcal{S}_6$, we have $(\tau',\vec{m}) \simeq_{\mathcal{S}_6} \underline{\sigma}$, so $\vec{\alpha}\,\vec{\rho}\,\vec{\epsilon} \cap \text{fv}(\mathcal{S}_6(\vec{m})) = \emptyset$. Thus $\text{Dom}(\underline{S''}) \cap \text{fv}(\mathcal{S}_6(\vec{m})) = \emptyset$, from which the required follows. The remaining three requirements are all easily seen to hold by the definition of $\mathcal{S}_6$ (in particular, $\mathcal{S}_6(\tau) = \underline{\tau}$) and again using that $(\tau',\vec{m}) \simeq_{\mathcal{S}_6} \underline{\sigma}$, so $\mathcal{S}_6(\tau') = \underline{\tau'}$, and, by (F.154), $\underline{S''}(\underline{\tau'}) = \underline{\tau}$. This completes the proof of Claim **I**.

Let

$$\rho''_1,\ldots,\rho''_{k'} = \mathit{actuals}_{\mathcal{S}_6}(\tau',\tau,\vec{m}) \setminus (\mathcal{S}_6(\rho'_1),\ldots,\mathcal{S}_6(\rho'_{2^n})). \tag{F.183}$$

Then by normal form of $\underline{\sigma}$ (from (F.155) and (F.152)), and the definition of *actuals* and the definition of $\mathcal{S}_6$ we get that

$$\mathcal{S}_6(\rho'_1),\ldots,\mathcal{S}_6(\rho'_{2^n}),\rho''_1,\ldots,\rho''_{k'} = \underline{S}(\underline{\rho_1}),\ldots,\underline{S}(\underline{\rho_k}). \tag{F.184}$$

Now note that, by (F.176),

$$\mathrm{Dom}(D_5) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.185}$$

and by (F.177)–(F.180), (F.172); (F.174); (F.175); and (F.176)

$$\mathrm{Dom}(\mathcal{S}_5) \subseteq \mathrm{Dom}(D_5 \setminus D_4) \tag{F.186}$$

and, by (F.156), (F.151), (F.158), (F.159), (F.160), and (F.177)–(F.180)

$$\mathcal{S}_6(D_5) \subseteq A' \cup A. \tag{F.187}$$

Moreover, by (F.171), (F.172), (F.173), (F.174), (F.175), (F.176), (F.182), (F.183), and (F.184) we can apply rule 16 to get (using that, by (F.177), $\underline{\rho} = \mathcal{S}_6(\rho)$)

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_6} f : \overline{\tau} \Rightarrow f\,[\underline{S}(\underline{\rho_1}), \ldots, \underline{S}(\underline{\rho_k})] \;\texttt{at}\; \underline{\rho} : (\tau, \rho), \epsilon, D_5, \mathcal{C}_2 \tag{F.188}$$

Further, we have by definition of $\mathcal{S}_6$ that

$$\mathcal{S}_6(\tau) = \underline{\tau} \tag{F.189}$$
$$\mathcal{S}_6(\rho) = \underline{\rho} \tag{F.190}$$
$$\mathcal{S}_6(\epsilon) = \underline{\epsilon}.\underline{\varphi}. \tag{F.191}$$

Now (F.185)–(F.191) together with (F.181) is the required, completing this case of the proof.

Case (Rule (11)**)** Then

$$\binom{A}{B}, TE \vdash \lambda\, x : \overline{\tau_1}.e \Rightarrow \lambda\, x.e' \;\texttt{at}\; \underline{\rho} : ((\underline{\tau_1}, \underline{\rho_1}) \xrightarrow{\underline{\epsilon_3}.\underline{\varphi_3}} (\underline{\tau_2}, \underline{\rho_2}), \underline{\rho}), \{\underline{\rho}\}, \underline{\epsilon} \tag{F.192}$$

and by inversion we get that

$$\binom{A_1}{B}, TE + \{x \mapsto (\underline{\tau_1}, \underline{\rho_1})\} \vdash e \Rightarrow e' : (\underline{\tau_2}, \underline{\rho_2}), \underline{\varphi_2}, \underline{\epsilon_2} \tag{F.193}$$
$$\underline{\varphi_2} \subseteq \underline{\varphi_3} \tag{F.194}$$
$$\mathrm{ML}(\underline{\tau_1}) = \overline{\tau_1} \tag{F.195}$$
$$\mathrm{ML}((\underline{\tau_1}, \underline{\rho_1}) \xrightarrow{\underline{\epsilon_3}.\underline{\varphi_3}} (\underline{\tau_2}, \underline{\rho_2})) = \overline{\tau} \tag{F.196}$$
$$B \vdash TE \tag{F.197}$$
$$(\{\underline{\rho}\}, \{\underline{\epsilon_3}.\underline{\varphi_3}\}) \subseteq B \tag{F.198}$$
$$A = A_1 \uplus \{\underline{\epsilon}.\{\underline{\rho}\}\} \quad \text{exists.} \tag{F.199}$$

Further by assumption

$$\Gamma \simeq_{\mathcal{S}_1} TE \tag{F.200}$$
$$\text{frev}(\Gamma) \subseteq \text{Dom}(D_1) \tag{F.201}$$
$$\text{Dom}(\mathcal{S}_1) \subseteq \text{Dom}(D_1) \tag{F.202}$$
$$D_1 \quad \text{is a simple basis} \tag{F.203}$$
$$\text{Dom}(D_1) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.204}$$
$$\begin{pmatrix} A' \\ B \end{pmatrix} \uplus \begin{pmatrix} A \\ B \end{pmatrix} \quad \text{exists} \tag{F.205}$$
$$\mathcal{S}_1(D_1) \subseteq A' \tag{F.206}$$
$$(\mathcal{C}_1, \Gamma) \quad \text{is pre-well-tagged} \tag{F.207}$$
$$\text{frev}(\mathcal{C}_1) \subseteq \text{Dom}(D_1) \tag{F.208}$$
$$\mathcal{S}_1 \text{ solves } \mathcal{C}_1. \tag{F.209}$$

Let

$$(\tau_1, \rho_1, D_2) = \text{spread}(\overline{\tau_1}, D_1) \tag{F.210}$$
$$\text{s.t.} \quad \text{Dom}(D_2) \cap \text{Dom}(A' \cup A \cup A''\emptyset$$
$$\mathcal{S}_2 = \text{mk\_}\mathcal{S}\left((\tau_1, \rho_1), (\underline{\tau_1}, \underline{\rho_1})\right) \tag{F.211}$$
$$\mathcal{S}_3 = \mathcal{S}_1 + \mathcal{S}_2 \tag{F.212}$$

Then by Lemma 71 on (F.200), (F.210), (F.211), and (F.212) we get

$$\Gamma + \{x \mapsto (\tau_1, \rho_1)\} \simeq_{\mathcal{S}_3} TE + \{x \mapsto (\underline{\tau_1}, \underline{\rho_1})\}. \tag{F.213}$$

Moreover, by (F.201) and (F.210), we have

$$\text{frev}(\Gamma + \{x \mapsto (\tau_1, \rho_1)\}) \subseteq \text{Dom}(D_2) \tag{F.214}$$

and by (F.202), (F.210); (F.211); and (F.212)

$$\text{Dom}(\mathcal{S}_3) \subseteq \text{Dom}(D_2). \tag{F.215}$$

Further, by (F.203) and (F.210)

$$D_2 \quad \text{is a simple basis.} \tag{F.216}$$

Let

$$A_1'' = A' \cup A \cup A'' \tag{F.217}$$

and let

$$A_1' = A'. \tag{F.218}$$

Then by (F.210), (F.217), and (F.218)

$$\mathrm{Dom}(D_2) \cap \mathrm{Dom}(A_1' \cup A_1 \cup A_1'') = \emptyset. \tag{F.219}$$

Further, by (F.205) and (F.199), we have

$$\binom{A_1'}{B} \uplus \binom{A_1}{B} \quad \text{exists} \tag{F.220}$$

and by (F.210), (F.211), (F.212), Lemma 28 on (F.192) (which in particular gives that $B \vdash (\underline{\tau_1}, \underline{\rho_1})$), and (F.206) we get

$$\mathcal{S}_3(D_2) \subseteq A_1'. \tag{F.221}$$

Moreover, by (F.207)

$$(\mathcal{C}_1, \Gamma + \{x \mapsto (\tau_1, \rho_1)\}) \quad \text{is pre-well-tagged} \tag{F.222}$$

and by (F.208) and (F.210)

$$\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_2) \tag{F.223}$$

and by Lemma72 on (F.208), (F.209), (F.210), (F.211), and (F.212)

$$\mathcal{S}_3 \,\text{solves}\, \mathcal{C}_1. \tag{F.224}$$

Now we can apply induction on (F.193), (F.213), (F.214), (F.215), (F.216), (F.219), (F.220), (F.221), (F.222), (F.223), (F.224) to get that there exist D_3, $\mathcal{S}_4$, $\mathcal{C}_2$, τ_2, ρ_2, and ϵ_2 such that

$$\mathrm{Dom}(D_3) \subseteq \mathrm{Dom}(A_1' \cup A_1 \cup A_1'') = \emptyset \tag{F.225}$$
$$\mathrm{Dom}(\mathcal{S}_4) \subseteq \mathrm{Dom}(D_3 \setminus D_2) \tag{F.226}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(D_3) \subseteq A_1' \cup A_1 \tag{F.227}$$
$$\mathcal{C}_1, D_2, \Gamma + \{x \mapsto (\tau_1, \rho_1)\} \Vdash_{\mathcal{S}_3 + \mathcal{S}_4} e \Rightarrow e' : (\tau_2, \rho_2), \epsilon_2, D_3, \mathcal{C}_2 \tag{F.228}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(\tau_2) = \underline{\tau_2} \tag{F.229}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(\rho_2) = \underline{\rho_2} \tag{F.230}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(\epsilon_2) = \underline{\epsilon_2}.\underline{\varphi_2} \tag{F.231}$$

Moreover, by Lemma 54 on (F.207), (F.223), (F.214), (F.224), and (F.228) we get

$$\mathcal{C}_1 \subseteq \mathcal{C}_2 \tag{F.232}$$
$$\mathcal{C}_2 \quad \text{is pre-well-tagged} \tag{F.233}$$
$$D_2 \subseteq D_3 \tag{F.234}$$
$$D_3 \quad \text{is a simple basis} \tag{F.235}$$
$$\mathrm{frev}(\mathcal{C}_2) \subseteq \mathrm{Dom}(D_3) \tag{F.236}$$
$$\mathcal{S}_3 + \mathcal{S}_4 \,\text{solves}\, \mathcal{C}_2. \tag{F.237}$$

Let

$$\mathcal{S}_5 = \mathcal{S}_3 + \mathcal{S}_5 \tag{F.238}$$

$$(\rho, D_4) = \text{freshRegVar}(D_3) \tag{F.239}$$
$$\text{s.t.} \quad \text{Dom}(D_4) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset$$

$$(\epsilon, \epsilon_3, D_5) = \text{freshEffVar}(D_4) \tag{F.240}$$
$$\text{s.t.} \quad \text{Dom}(D_5) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset$$

$$\mathcal{S}_6 = \{\epsilon_3 \mapsto \underline{\epsilon_3}.\underline{\varphi_3}, \epsilon \mapsto \underline{\epsilon}.\{\underline{\rho}\}, \rho \mapsto \underline{\rho}\} \tag{F.241}$$

Then by Lemma 55 on (F.228); (F.214), (F.234); (F.236); (F.239); (F.240); (F.241) we have

$$\mathcal{C}_1, D_2, \Gamma + \{x \mapsto (\tau_1, \rho_1)\} \Vdash_{\mathcal{S}_5 + \mathcal{S}_6} e \Rightarrow e' : (\tau_2, \rho_2), \epsilon_2, D_3, \mathcal{C}_2 \tag{F.242}$$

Let

$$\mathcal{S}_7 = \mathcal{S}_5 + \mathcal{S}_6 \tag{F.243}$$

By Lemma 72 on (F.239), (F.240), (F.241), (F.243), (F.238), (F.237), and (F.236) we get

$$\mathcal{S}_7 \text{ solves } \mathcal{C}_2. \tag{F.244}$$

Let

$$\mathcal{C}_3 = \{\epsilon_3 \mathrel{\dot{\supseteq}} \epsilon_2, \epsilon \mathrel{\dot{\supseteq}} \rho\} \cup \mathcal{C}_2. \tag{F.245}$$

Then by (F.194), (F.238)–(F.241), (F.243), and (F.244) we also have

$$\mathcal{S}_7 \text{ solves } \mathcal{C}_3. \tag{F.246}$$

Now by (F.210), (F.242), (F.239), (F.240), (F.246) we can apply rule (18) to conclude (using that, by (F.241), $\mathcal{S}_7(\rho) = \underline{\rho}$)

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_7} \lambda\, x : \overline{\tau_1}.e : \overline{\tau} \Rightarrow \tag{F.247}$$
$$\lambda\, x.e' \ \texttt{at} \ \underline{\rho} : ((\tau_1, \rho_1) \xrightarrow{\epsilon_3} (\tau_2, \rho_2), \rho), \epsilon, D_5, \mathcal{C}_3$$

Further by (F.240) we have

$$\text{Dom}(D_5) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.248}$$

and by (F.211), (F.226), (F.241), (F.210), (F.234), (F.239), and (F.240) we get

$$\text{Dom}(\mathcal{S}_2 + \mathcal{S}_4 + \mathcal{S}_6) \subseteq \text{Dom}(D_5 \setminus D_1) \tag{F.249}$$

and by (F.227), (F.239), (F.240), (F.241), (F.198), and (F.199) we have

$$\mathcal{S}_7(D_5) \subseteq A' \cup A \tag{F.250}$$

and by definition of $\mathcal{S}_7$ we have

$$\mathcal{S}_7\Big((\tau_1, \rho_1) \xrightarrow{\epsilon_3} (\tau_2, \rho_2)\Big) = (\underline{\tau_1}, \underline{\rho_1}) \xrightarrow{\underline{\epsilon_3}.\underline{\varphi_3}} (\underline{\tau_2}, \underline{\rho_2}) \tag{F.251}$$
$$\mathcal{S}_7(\rho) = \underline{\rho} \tag{F.252}$$
$$\mathcal{S}_7(\epsilon) = \underline{\epsilon}.\{\underline{\rho}\} \tag{F.253}$$

so noting that $\mathcal{S}_7 = \mathcal{S}_1 + \mathcal{S}_2 + \mathcal{S}_4 + \mathcal{S}_6$ we have the required by (F.247)–(F.253), completing the proof of this case.

Case (Rule (14) **preceded by rule** (12)**)** Then

$$\binom{A}{B}, TE \vdash e_1\ e_2 : \overline{\tau} \Rightarrow \texttt{letregion}\ \vec{\rho}\ \texttt{in}\ e_1'\ e_2' : (\underline{\tau}, \underline{\rho}), \underline{\varphi}, \underline{\epsilon} \tag{F.254}$$

and by inversion

$$B \uplus B_1 \quad \text{exists} \tag{F.255}$$
$$\underline{\varphi} = \text{Observe}(TE, \underline{\tau}, \underline{\rho}), \underline{\varphi_4}) \tag{F.256}$$
$$\vec{\rho} = \text{frv}(\underline{\varphi_4} \setminus \underline{\varphi}) \tag{F.257}$$
$$A = A_4 \uplus \{\underline{\epsilon}.\underline{\varphi}\} \tag{F.258}$$
$$\binom{A_4}{B \uplus B_1}, TE \vdash e_1\ e_2 \Rightarrow e_1'\ e_2' : (\underline{\tau}, \underline{\rho}), \underline{\varphi_4}, \underline{\epsilon_4} \tag{F.259}$$
$$\binom{A_1}{B \uplus B_1}, TE \vdash e_1 \Rightarrow e_1' : \Big((\underline{\tau_2}, \underline{\rho_2}) \xrightarrow{\underline{\epsilon_3}.\underline{\varphi_3}} (\underline{\tau}, \underline{\rho}), \underline{\rho_1}\Big), \underline{\varphi_1}, \underline{\epsilon_1} \tag{F.260}$$
$$\binom{A_2}{B \uplus B_1}, TE \vdash e_2 \Rightarrow e_2' : (\underline{\tau_2}, \underline{\rho_2}), \underline{\varphi_2}, \underline{\epsilon_2} \tag{F.261}$$
$$\text{ML}(\underline{\tau}) = \overline{\tau} \tag{F.262}$$
$$\binom{A_3}{B \uplus B_1} = \binom{A_1}{B \uplus B_1} + \binom{A_2}{B \uplus B_1} \tag{F.263}$$
$$\underline{\varphi_4} = \underline{\varphi_1} \cup \underline{\varphi_2} \cup \underline{\varphi_3} \cup \{\underline{\epsilon_3}, \underline{\rho_1}\} \tag{F.264}$$
$$A_4 = A_3 \uplus \{\underline{\epsilon_4}.\underline{\varphi_4}\} \quad \text{exists} \tag{F.265}$$
$$B \vdash TE, (\underline{\tau}, \underline{\rho}) \tag{F.266}$$

Further by assumption

$$\Gamma \simeq_{\mathcal{S}_1} TE \tag{F.267}$$
$$\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_1) \tag{F.268}$$
$$\mathrm{Dom}(\mathcal{S}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.269}$$
$$D_1 \quad \text{is a simple basis} \tag{F.270}$$
$$\mathrm{Dom}(D_1) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \tag{F.271}$$
$$\left(\begin{smallmatrix} A' \\ B \end{smallmatrix}\right) \uplus \left(\begin{smallmatrix} A \\ B \end{smallmatrix}\right) \quad \text{exists} \tag{F.272}$$
$$\mathcal{S}_1(D_1) \subseteq A' \tag{F.273}$$
$$(\mathcal{C}_1, \Gamma) \quad \text{is pre-well-tagged} \tag{F.274}$$
$$\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_1) \tag{F.275}$$
$$\mathcal{S}_1 \text{ solves } \mathcal{C}_1. \tag{F.276}$$

Let

$$\begin{aligned} &\left((\tau_{11}, \rho_{11}) \xrightarrow{\epsilon_3} (\tau_{12}, \rho_{12}), \_, D_2\right) = \mathrm{spread}(\mathrm{MLTypeOf}(e_1), D_1) \\ &\quad \text{s.t.} \quad \mathrm{Dom}(D_2) \cap \mathrm{Dom}(A' \cup A \cup A'') = \emptyset \end{aligned} \tag{F.277}$$

and let

$$\mathcal{S}_2 = \mathrm{mk\_}\mathcal{S}\left((\tau_{11}, \rho_{11}) \xrightarrow{\epsilon_3} (\tau_{12}, \rho_{12}), (\underline{\tau_2}, \underline{\rho_2}) \xrightarrow{\underline{\epsilon_3 . \varphi_3}} (\underline{\tau}, \underline{\rho})\right) \tag{F.278}$$

By (F.258), (F.265), (F.263), and (F.272)

$$A'_1 = A' \uplus B_1 \quad \text{exists} \tag{F.279}$$

and

$$\left(\begin{smallmatrix} A'_2 \\ B \uplus B_1 \end{smallmatrix}\right) = \left(\begin{smallmatrix} A'_1 \\ B \uplus B_1 \end{smallmatrix}\right) \uplus \left(\begin{smallmatrix} A_1 \\ B \uplus B_1 \end{smallmatrix}\right) \quad \text{exists.} \tag{F.280}$$

Let

$$A''_1 = A' \cup A \cup A'' \tag{F.281}$$

and let

$$\mathcal{S}_3 = \mathcal{S}_1 + \mathcal{S}_2. \tag{F.282}$$

By Lemma 71 on (F.267), (F.268), (F.269), (F.278), and (F.282) we get that

$$\Gamma \simeq_{\mathcal{S}_3} TE \tag{F.283}$$

and by (F.268) and (F.277)

$$\mathrm{frev}(\Gamma) \subseteq \mathrm{Dom}(D_2). \tag{F.284}$$

Further, by (F.269), (F.278), and (F.282), we get

$$\mathrm{Dom}(\mathcal{S}_3) \subseteq \mathrm{Dom}(D_2) \tag{F.285}$$

and

$$D_2 \quad \text{is a simple basis.} \tag{F.286}$$

Also, by (F.273) and Lemma 28 on (F.260) (which in particular gives that $B \uplus B_1 \vdash (\underline{\tau_2}, \underline{\rho_2}) \xrightarrow{\epsilon_3.\varphi_3} (\underline{\tau}, \underline{\rho})$), we get

$$\mathcal{S}_3(D_2) \subseteq A_1' \tag{F.287}$$

and by (F.275) and (F.277)

$$\mathrm{frev}(\mathcal{C}_1) \subseteq \mathrm{Dom}(D_2). \tag{F.288}$$

Further, by Lemma 72 on (F.276), (F.275), (F.269), (F.277), and (F.278), we get

$$\mathcal{S}_3 \text{ solves } \mathcal{C}_1. \tag{F.289}$$

By induction on (F.259), (F.283), (F.284), (F.285), (F.286), (F.277), (F.280), (F.287), (F.274), (F.288), and (F.289) there exist D_3, $\mathcal{S}_4$, $\mathcal{C}_2$, τ_1, ρ_1, and ϵ_1 such that

$$\mathrm{Dom}(D_3) \cap \mathrm{Dom}(A_1' \cup A_1 \cup A_1'') = \emptyset \tag{F.290}$$
$$\mathrm{Dom}(\mathcal{S}_4) \subseteq \mathrm{Dom}(D_3 \setminus D_2) \tag{F.291}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(D_3) \subseteq A_1' \cup A_1 \tag{F.292}$$
$$\mathcal{C}_1, D_2, \Gamma \Vdash_{\mathcal{S}_3+\mathcal{S}_4} e_1 \Rightarrow e_1' : (\tau_1, \rho_1), \epsilon_1, D_4, \mathcal{C}_2 \tag{F.293}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(\tau_1) = (\underline{\tau_2}, \underline{\rho_2}) \xrightarrow{\epsilon_3.\varphi_3} (\underline{\tau}, \underline{\rho}) \tag{F.294}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(\rho_1) = \underline{\rho_1} \tag{F.295}$$
$$(\mathcal{S}_3 + \mathcal{S}_4)(\epsilon_1) = \underline{\epsilon_1}.\underline{\varphi_1}. \tag{F.296}$$

Moreover, by Lemma 54 on (F.288), (F.284), (F.274), (F.289), (F.293), and (F.286) we have that

$$\mathcal{C}_1 \subseteq \mathcal{C}_2 \tag{F.297}$$
$$\mathcal{C}_2 \quad \text{is pre-well-tagged} \tag{F.298}$$
$$D_2 \subseteq D_3 \tag{F.299}$$
$$D_3 \quad \text{is a simple basis} \tag{F.300}$$
$$\mathcal{S}_3 + \mathcal{S}_4 \text{ solves } \mathcal{C}_2 \tag{F.301}$$
$$\mathrm{frev}(\mathcal{C}_2) \subseteq \mathrm{Dom}(D_3). \tag{F.302}$$
$$\tag{F.303}$$

Let

$$\mathcal{S}_5 = \mathcal{S}_3 + \mathcal{S}_4. \tag{F.304}$$

Then by Lemma 71 on (F.283), (F.291), and (F.284) we have that

$$\Gamma \simeq_{\mathcal{S}_5} TE. \tag{F.305}$$

Let

$$A_2'' = A' \cup A \cup A''. \tag{F.306}$$

By (F.284) and (F.299) we have that

$$\text{frev}(\Gamma) \subseteq \text{Dom}(D_3). \tag{F.307}$$

Also by (F.291), (F.285), and (F.304) we get that

$$\text{Dom}(\mathcal{S}_5) \subseteq \text{Dom}(D_3) \tag{F.308}$$

and by (F.290), (F.281), (F.306), (F.263), (F.265), (F.258)

$$\text{Dom}(D_3) \cap \text{Dom}(A_2' \cup A_2 \cup A_2'') = \emptyset. \tag{F.309}$$

By (F.263), (F.280), (F.279), (F.265), and (F.269)

$$\left(\begin{smallmatrix} A_2' \\ B \uplus B_1 \end{smallmatrix}\right) \uplus \left(\begin{smallmatrix} A_2 \\ B \uplus B_1 \end{smallmatrix}\right) \quad \text{exists.} \tag{F.310}$$

By (F.292) and (F.280)

$$\mathcal{S}_5(D_3) \subseteq A_2'. \tag{F.311}$$

By (F.298) and (F.274) we get

$$(\mathcal{C}_2, \Gamma) \quad \text{is pre-well-tagged.} \tag{F.312}$$

By induction on (F.261), (F.305), (F.307), (F.308), (F.300), (F.309), (F.310), (F.311), (F.312), (F.302), and (F.301), (F.304) there exist D_4, $\mathcal{S}_6$, $\mathcal{C}_3$, τ_2, ρ_2, and ϵ_2 such that

$$\text{Dom}(D_4) \cap \text{Dom}(A_2' \cup A_2 \cup A_2'') = \emptyset \tag{F.313}$$
$$\text{Dom}(\mathcal{S}_6) \subseteq \text{Dom}(D_4 \setminus D_3) \tag{F.314}$$
$$(\mathcal{S}_5 + \mathcal{S}_6)(D_4) \subseteq A_2' \cup A_2 \tag{F.315}$$
$$\mathcal{C}_2, D_3, \Gamma \Vdash_{\mathcal{S}_5+\mathcal{S}_6} e_2 \Rightarrow e_2' : (\tau_2, \rho_2), \epsilon_2, D_4, \mathcal{C}_4 \tag{F.316}$$
$$(\mathcal{S}_5 + \mathcal{S}_6)(\tau_2) = \underline{\tau_2} \tag{F.317}$$
$$(\mathcal{S}_5 + \mathcal{S}_6)(\rho_2) = \underline{\rho_2} \tag{F.318}$$
$$(\mathcal{S}_5 + \mathcal{S}_6)(\epsilon_2) = \underline{\epsilon_2}.\underline{\varphi_2}. \tag{F.319}$$

Moreover, by Lemma 54 on (F.312); (F.302); (F.307); (F.301), (F.304); (F.316); and (F.300) we have that

$$\mathcal{C}_2 \subseteq \mathcal{C}_3 \tag{F.320}$$
$$\mathcal{C}_3 \quad \text{is pre-well-tagged} \tag{F.321}$$
$$D_3 \subseteq D_4 \tag{F.322}$$
$$D_4 \quad \text{is a simple basis} \tag{F.323}$$
$$\mathcal{S}_5 + \mathcal{S}_6 \,\text{solves}\, \mathcal{C}_4 \tag{F.324}$$
$$\text{frev}(\mathcal{C}_3) \subseteq \text{Dom}(D_4). \tag{F.325}$$

Let

$$(\epsilon_4, \epsilon, D_5) = \text{freshEffVar}(D_4) \tag{F.326}$$
$$\text{s.t.} \quad \text{Dom}(D_5) \cap \text{Dom}(A' \cup A \cup A'') = \emptyset$$

and let

$$\mathcal{S}_7 = \mathcal{S}_5 + \mathcal{S}_6 \tag{F.327}$$
$$\mathcal{S}_8 = \{\epsilon_4 \mapsto \underline{\epsilon_4}.\underline{\varphi_4}, \epsilon \mapsto \underline{\epsilon}.\underline{\varphi}\} \tag{F.328}$$
$$\mathcal{S}_9 = \mathcal{S}_7 + \mathcal{S}_8. \tag{F.329}$$

By Lemma 71 on (F.305), (F.314), (F.307), (F.326), (F.328), and (F.329) we get that

$$\Gamma \simeq_{\mathcal{S}_9} TE. \tag{F.330}$$

Let

$$\mathcal{C}_4 = \{\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, \mathbf{c_4}\} \cup \mathcal{C}_3 \tag{F.331}$$

with $\mathbf{c_1}, \ldots, \mathbf{c_4}$ as in rule 20. Then by Lemma 72 on (F.324), (F.326), and (F.325)

$$\mathcal{S}_9 \,\text{solves}\, \mathcal{C}_3 \tag{F.332}$$

and it is now easy to see that

$$\mathcal{S}_9 \,\text{solves}\, \mathcal{C}_4. \tag{F.333}$$

Finally by Lemma 55 on (F.293); (F.284); (F.285), (F.291); (F.314); (F.326)–(F.329) we get

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_9} e_1 \Rightarrow e_1' : (\tau_1, \rho_1), \epsilon_1, D_2, \mathcal{C}_2. \tag{F.334}$$

Likewise by Lemma 55 on (F.316); (F.307); (F.308), (F.314); (F.326)–(F.329) we get

$$\mathcal{C}_2, D_3, \Gamma \Vdash_{\mathcal{S}_9} e_1 \Rightarrow e_1' : (\tau_2, \rho_2), \epsilon_2, D_4, \mathcal{C}_3. \tag{F.335}$$

By rule 20 on (F.277), (F.334), (F.335), (F.326), (F.333), (F.328), and (F.257) we conclude

$$\mathcal{C}_1, D_1, \Gamma \Vdash_{\mathcal{S}_9} e_1\ e_2 \Rightarrow \texttt{letregion}\ \vec{\rho}\ \texttt{ in }\ e_1'\ e_2' : (\tau_{12}, \rho_{12}), \epsilon, D_5, \mathcal{C}_4. \quad \text{(F.336)}$$

Letting

$$\mathcal{S}_{10} = \mathcal{S}_2 + \mathcal{S}_4 + \mathcal{S}_6 + \mathcal{S}_8 \quad \text{(F.337)}$$

and noting that

$$\mathcal{S}_9 = \mathcal{S}_1 + \mathcal{S}_{10} \quad \text{(F.338)}$$

we get that

$$\mathrm{Dom}(\mathcal{S}_{10}) \subseteq D_5 \setminus D_1 \quad \text{(F.339)}$$

by (F.277), (F.278), (F.291), (F.314), (F.326), and (F.328). Furthermore, by (F.315), (F.328), (F.280), (F.263), (F.265), and (F.258)

$$\mathcal{S}_9(D_5) \subseteq A' \cup A. \quad \text{(F.340)}$$

Also

$$\mathcal{S}_9(\tau_{12}) = \underline{\tau} \quad \text{(F.341)}$$
$$\mathcal{S}_9(\rho_{12}) = \underline{\rho} \quad \text{(F.342)}$$
$$\mathcal{S}_9(\epsilon) = \underline{\epsilon}.\underline{\varphi} \quad \text{(F.343)}$$
$$\text{(F.344)}$$

Finally note that (F.326), (F.339), (F.338), (F.336), (F.341), (F.342), and (F.343) is the required, completing the proof of this case.

Case (Rule (12)**)** The proof for this case is essentially just a part of the proof for the preceding case.

Case (Rule 15) This case is obvious.

We have now covered all cases, completing the proof of the theorem.

G Proof of Solution to Normal Form

In this section we complete the proof of Theorem 61 by showing that

$$(\mathcal{S}, (\underline{S}_1, \ldots, \underline{S}_k))$$

(defined in the part of the proof in the main text) satisfies all the constraints in $\mathcal{C}$.

We proceed in the order of the enumeration of the requirements in the definition of solution (Definition 49).

(1) Immediate by item (1) in the definition of normal form.
(2) By definition of $\underline{S}_i$.
(3) By definition of $\underline{S}_i$ (and using that $\mathcal{S}_D$ is the identity on type variables so in case $\tau = \alpha$, the required also holds as then also $\underline{S}_i(\alpha) = \alpha$).
(4) By item (6) in the definition of normal form and the definition of $\underline{S}_i$.
(5) Let $\epsilon.\underline{\varphi} = \mathcal{S}_D(\epsilon)$ and let $\epsilon'.\underline{\varphi'} = \mathcal{S}_D(\epsilon')$.
  (a) We are to show that $\underline{S}_i(\epsilon.\underline{\varphi}) = \epsilon'.\underline{\varphi'}$. By the definition of $\underline{S}_i$ this is equivalent to showing $\epsilon'.\underline{\varphi'} \cup \underline{S}_i(\underline{\varphi}) = \epsilon'.\underline{\varphi'}$ and thus it suffices to show that $\underline{S}_i(\underline{\varphi}) \subseteq \underline{\varphi'}$. There are four cases to consider:

  Case (i) $\rho'' \in \underline{\varphi} \wedge \rho'' \in \mathrm{frv}(\mathcal{S}_D(\vec{m}))$. Then by items (8b) and (7) in the definition of normal form, $\rho'' \notin \mathrm{Dom}(\underline{S}_i)$ (we have showed in the part of the proof in the main text that $\mathrm{Dom}(\underline{S}_i) \subseteq \mathrm{fv}(\mathcal{S}_D(\tau)) \setminus \mathrm{fv}(\mathcal{S}_D(\vec{m})))$. Thus $\underline{S}_i(\rho'') = \rho'' \in \underline{\varphi'}$.

  Case (ii) $\rho'' \in \underline{\varphi} \wedge \rho'' \notin \mathrm{frv}(\mathcal{S}_D(\vec{m}))$. Then by item (8a) in the definition of normal form, letting $j = \mathit{index}(\rho'', \mathcal{S}_D(\tau))$, $\rho'' \dot{\succeq}^i \rho_j \in \mathcal{C}$ and $\epsilon' \dot{\supseteq} \rho_j \in \mathcal{C}$. Thus by definition of $\underline{S}_i$ and by item (4) in the definition of normal form, $\underline{S}_i(\rho'') = \rho_j$ and by item (7) in the definition of normal form, $\underline{S}_i(\rho'') = \rho_j \in \underline{\varphi'}$.

  Case (iii) $\epsilon'' \in \underline{\varphi} \wedge \epsilon'' \in \mathrm{fev}(\mathcal{S}_D(\vec{m}))$. Then as in Case (i), we get $\underline{S}_i(\epsilon'') = \epsilon''.\emptyset$ and $\epsilon'' \in \underline{\varphi'}$.

  Case (iv) $\epsilon'' \in \underline{\varphi} \wedge \epsilon'' \notin \mathrm{fev}(\mathcal{S}_D(\vec{m}))$. Then by item (8c) in the definition of normal form, letting $j = \mathit{index}(\epsilon'', \mathcal{S}_D(\tau))$, $\epsilon'' \dot{\succeq}^i \epsilon_j \in \mathcal{C}$ and $\epsilon' \dot{\supseteq} \overline{\epsilon_j} \in \mathcal{C}$. By item (5) and the definition of $\underline{S}_i$, $\underline{S}_i(\epsilon'') = \epsilon_j.\underline{\varphi_j}$, for some $\underline{\varphi_j}$ [in case $\epsilon'' = \epsilon_j$, $\underline{\varphi_j} = \emptyset$] and we are to show that $\{\epsilon_j\} \cup \underline{\varphi_j} \subseteq \underline{\varphi'}$. But this follows by item (7) in the definition of normal form since, as mentioned, $\epsilon' \dot{\supseteq} \overline{\epsilon_j} \in \mathcal{C}$.

  By cases **(i)**–**(iv)** we have that $\underline{S}_i(\underline{\varphi}) \subseteq \underline{\varphi'}$, as required.
  (b) By items (8a) and (4) in the definition of normal form and the definition of $\underline{S}_i$.
  (c) By items (8c) and (5) in the definition of normal form and the definition of $\underline{S}_i$.
(6) By item (7) in the definition of normal form.
(7) By items (9) and (1) in the definition of normal form.
(8) By item (2) in the definition of normal form.

This completes the proof of Theorem 61. □

H Proof of Soundness of Normalization

Theorem 73 (Soundness of Normalization) *If*

$$((\mathcal{C}_1, I_1, k_1), D_1) \Longrightarrow_{\underline{S}} ((\mathcal{C}_2, I_2, k), D_2)$$

and $\mathcal{S}_2$ is a solution to $((\mathcal{C}_2, I_2, k), D_2)$, then $\mathcal{S}_1 = \mathcal{S}_2 \circ \underline{S}$ is a solution to $((\mathcal{C}_1, I_1, k_1), D_1)$.

PROOF. By assumption $\mathrm{Dom}(\mathcal{S}_2) \subseteq D_2$ and there exists a consistent basis A and $\underline{S}_1, \ldots, \underline{S}_k$ semantic substitutions such that

A0. $\mathcal{S}_2(\mathcal{S}_{D_2}(\epsilon_2)) = \mathcal{S}_2(\epsilon_2)$, for all $\epsilon_2 \in \mathrm{Dom}(D_2)$;
A1. $\mathcal{S}_2(D_2) \subseteq A$;
A2. $\forall i \in \{1, \ldots, k\}. \mathrm{Dom}(\underline{S}_i) \subseteq \mathrm{fv}(\mathcal{S}_2(\tau_i^2)) \backslash \mathrm{fv}(\mathcal{S}_2(\vec{m}_i{}^2)$, where $(\tau_i^2, \vec{m}_i{}^2, \_, \_, \_) = I_2(i)$;
A3. $(\mathcal{S}_2, (\underline{S}_1, \ldots, \underline{S}_k))$ satisfies all the constraints in $\mathcal{C}_2$.

We proceed by cases of the rewriting rule applied. In each case we will show that

C0. $\mathcal{S}_1(\mathcal{S}_{D_1}(\epsilon_1)) = \mathcal{S}_1(\epsilon_1)$, for all $\epsilon_1 \in \mathrm{Dom}(D_1)$;
C1. $\mathcal{S}_1(D_1) \subseteq A$;
C2. $\forall i \in \{1, \ldots, k\}. \mathrm{Dom}(\underline{S}_i) \subseteq \mathrm{fv}(\mathcal{S}_1(\tau_i^1)) \backslash \mathrm{fv}(\mathcal{S}_1(\vec{m}_i{}^1)$, where $(\tau_i^1, \vec{m}_i{}^1, \_, \_, \_) = I_1(i)$;
C3. $(\mathcal{S}_1, (\underline{S}_1, \ldots, \underline{S}_k))$ satisfies all the constraints in $\mathcal{C}_1$.

Now we consider each rewriting rule in turn.

R1. Then $D_2 = D_1$ and $\mathcal{S}_2 = \mathcal{S}_1$, so C0, C1, and C2 reduce to A0, A1, and A2. $C3$ follows by $A3$.
R2. Then $D_2 = D_1$ and $\mathcal{S}_2 = \mathcal{S}_1$, so C0, C1, and C2 are again obvious. $C3$ follows as $\mathcal{S}_1(\tau 1) = \underline{S}_i(\alpha) = \mathcal{S}_1(\tau_2)$ by assumption $A3$.
R3. Then $\underline{S} = \{\rho \mapsto \rho'\}$, $\mathcal{S}_1 = \mathcal{S}_2 \circ \underline{S}$, $D_2 = \underline{S}(D_1)$, $\underline{S} \circ \mathcal{S}_{D_1} = \mathcal{S}_{D_2}$, $\mathcal{C}_2 = \underline{S}(\mathcal{C}_1 \setminus \{\rho \doteq \rho'\})$. Thus $\mathcal{S}_1(D1) = \mathcal{S}_2(\underline{S}(D_1)) = \mathcal{S}_2(D_2) \subseteq A$ by A1. Hence C1 holds. For C2, note that $\mathcal{S}_1(\tau_i^1) = \mathcal{S}_2 \circ \underline{S}(\tau_i^1) = \mathcal{S}_2(\tau_i^2)$ and likewise $\mathcal{S}_1(\vec{m}_i{}^1) = \mathcal{S}_2(\vec{m}_i{}^2)$ so the required follows by A2. For C3, one checks that all constraints in $\mathcal{C}_1 \setminus \{\rho \doteq \rho'\}$ are satisfied using A3 and finally one notes that $\rho \doteq \rho'$ of course is satisfied. For C0, let $\epsilon \in \mathrm{Dom}(D_1)$; then $\mathcal{S}_1(\mathcal{S}_{D_1}(\epsilon)) = \mathcal{S}_2(\underline{S}(\mathcal{S}_{D_1}(\epsilon))) = \mathcal{S}_2(\mathcal{S}_{D_2}(\epsilon)) = \mathcal{S}_2(\epsilon) = \mathcal{S}_2(\underline{S}(\epsilon)) = \mathcal{S}_1(\epsilon)$.
R4. As R2.
R5. For C0, we are to show

$$\forall \epsilon_1 \in \mathrm{Dom}(D_1). \mathcal{S}_1(\mathcal{S}_{D_1}(\epsilon_1)) = \mathcal{S}_1(\epsilon_1).$$

Suppose first that $\epsilon_1 = \epsilon$. Then

$$\begin{aligned}
\mathcal{S}_1(\mathcal{S}_{D_1}(\epsilon)) &= \mathcal{S}_1(\epsilon.\underline{\varphi}) \\
&= \mathcal{S}_2(\underline{S}(\epsilon.\underline{\varphi})) \\
&= \mathcal{S}_2(\epsilon'.\{\epsilon \mapsto \epsilon'\}(\underline{S}(\underline{\varphi}) \cup \underline{\varphi} \cup \underline{\varphi'})) \\
&= \mathcal{S}_2(\epsilon'.\{\epsilon \mapsto \epsilon'\}(\underline{\varphi} \cup \underline{\varphi'})) \\
&= \mathcal{S}_2(\underline{S}(\epsilon)) \\
&= \mathcal{S}_1(\epsilon),
\end{aligned}$$

so we have the desired. The two other cases, when $\epsilon_1 = \epsilon'$ and when $\epsilon_1 \neq \epsilon \wedge \epsilon_1 \neq \epsilon'$, are dealt with by analogous calculations. This completes the proof of C0. For C1, $\mathcal{S}_1(D_1) = \mathcal{S}_2(\underline{S}(D_1)) = \mathcal{S}_2(D_2) \subseteq A$ by A0. Finally, C2 and C3 can be shown to follow from

$$\forall \vartheta \in \mathrm{fv}(\mathcal{C}_1).\mathcal{S}_1(\vartheta) = \mathcal{S}_2(\underline{S_3}(\vartheta)),$$

which can be shown by three simple calculations, one for each of the cases: $\vartheta \neq \epsilon \wedge \vartheta \neq \epsilon'$, $\vartheta = \epsilon$, and $\vartheta = \epsilon'$.

R6. As R2.

R7, R8, R11, R12. Obvious: $I_2 = I_1$, $D_2 = D_1$, $\mathcal{S}_1 = \mathcal{S}_2$, and $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

R10. Clear using that $I_2 = I_1$, $D_2 = D_1$, $\mathcal{S}_1 = \mathcal{S}_2$, and the definition of satisfaction of $(\tau, \vec{m}) \dot{\succeq}^i (\vec{\rho}, \vec{\epsilon}, \tau')$.

R9. For C0 we are to show that

$$\forall \epsilon_1 \in \mathrm{Dom}(D_1).\mathcal{S}_1(\mathcal{S}_{D_1}(\epsilon_1)) = \mathcal{S}_1(\epsilon_1).$$

Suppose first that $\epsilon_1 = \epsilon$. Then

$$\begin{aligned}
\mathcal{S}_1(\mathcal{S}_{D_1}(\epsilon)) &= \mathcal{S}_2(\underline{S}(\epsilon.\underline{\varphi})) \\
&= \mathcal{S}_2(\epsilon.\underline{\varphi} \cup \underline{\varphi'}) \\
&= \mathcal{S}_2(\underline{S}(\epsilon)) \\
&= \mathcal{S}_1(\epsilon),
\end{aligned}$$

as desired. Suppose then that $\epsilon_1 \neq \epsilon$. Then

$$\begin{aligned}
\mathcal{S}_1(\mathcal{S}_{D_1}(\epsilon_1)) &= \mathcal{S}_1(\epsilon_1.\underline{\varphi_1}), \qquad \text{where } \epsilon_1.\underline{\varphi_1} = \mathcal{S}_{D_1}(\epsilon_1) \\
&= \mathcal{S}_2(\underline{S}(\epsilon_1.\underline{\varphi_1})) \\
&= \mathcal{S}_2(\mathcal{S}_{D_2}(\epsilon_1)) \\
&= \mathcal{S}_2(\epsilon_1), \qquad \text{by A0,}
\end{aligned}$$

and

$$\mathcal{S}_1(\epsilon_1) = \mathcal{S}_2(\underline{S}(\epsilon_1)) = \mathcal{S}_2(\epsilon_1),$$

as desired. This completes the proof for C0. For C1, note that $\mathcal{S}_1(D_1) =$

$\mathcal{S}_2(\underline{S}(D_1)) = \mathcal{S}_2(D_2) \subseteq A$. For C2 and C3, use (as for R5) that

$$\forall\vartheta \in \mathrm{fv}(\mathcal{C}_1).\mathcal{S}_1(\vartheta) = \mathcal{S}_2(\vartheta),$$

which holds because:

Case $(\vartheta = \epsilon)$

$$\mathcal{S}_1(\epsilon) = \mathcal{S}_2(\underline{S}(\epsilon)) = \mathcal{S}_2(\epsilon.\underline{\varphi} \cup \underline{\varphi'}) = \mathcal{S}_2(\mathcal{S}_{D_2}(\epsilon)) = \mathcal{S}_2(\epsilon),$$

Case $(\vartheta \neq \epsilon)$

$$\mathcal{S}_1(\epsilon) = \mathcal{S}_2(\underline{S}(\vartheta)) = \mathcal{S}_2(\vartheta).$$

All rewriting rules have now been considered, completing the proof. □

I Proof of Strong Normalization of Rewriting

In this Appendix, we prove Theorem 66. To this end we need a couple of definitions.

Let c be an equality or instantiation constraint tagged with colours. Then the depth of c, denoted depth(c) is a natural number; it is defined as follows:

$$\begin{aligned}
\mathrm{depth}(\rho \doteq \rho') &= 1 \\
\mathrm{depth}(\epsilon \doteq \epsilon') &= 1 \\
\mathrm{depth}(\alpha \doteq \alpha) &= 1 \\
\mathrm{depth}((\tau_1, \rho_1) \xrightarrow{\epsilon_1} (\tau_1', \rho_1') & \\
\doteq (\tau_2, \rho_2) \xrightarrow{\epsilon_2} (\tau_2', \rho_2')) &= 4 + \mathrm{depth}(\tau_1 \doteq \tau_2) + \mathrm{depth}(\tau_1' \doteq \tau_2')
\end{aligned}$$

$$\begin{aligned}
\mathrm{depth}(\rho \mathrel{\dot{\succeq}^i} \rho') &= 4 \\
\mathrm{depth}(\epsilon \mathrel{\dot{\succeq}^i} \epsilon') &= 4 \\
\mathrm{depth}(\alpha \mathrel{\dot{\succeq}^i} \tau) &= 4 + \mathrm{depth}(\tau \doteq \tau) \\
\mathrm{depth}((\tau, \vec{m}) \mathrel{\dot{\succeq}^i} (\vec{\rho}, \vec{\epsilon}, \tau')) &= 1 + \Sigma_{m \in \vec{m}}\, \mathrm{depth}(m \mathrel{\dot{\succeq}^i} m) \\
&\quad + \Sigma_{(\alpha_0, \tau_0) \in \text{tyvar-pairs}(\tau, \tau')}\, \mathrm{depth}(\alpha_0 \mathrel{\dot{\succeq}^i} \tau_0) \\
&\quad + \Sigma_{(\rho, \rho') \in \text{regvar-pairs}(\tau, \tau')}\, \mathrm{depth}(\rho \mathrel{\dot{\succeq}^i} \rho') \\
&\quad + \Sigma_{(\epsilon, \epsilon') \in \text{effectvar-pairs}(\tau, \tau')}\, \mathrm{depth}(\epsilon \mathrel{\dot{\succeq}^i} \epsilon').
\end{aligned}$$

For a finite set X, we write $|X|$ for the cardinality of X. The size of a consistent basis $D = (Q, \Phi)$, denoted size(D), is defined to be (A, B) where $A = |Q| + |\mathrm{Dom}(\Phi)|$ and $B = A^2 - \Sigma_{\epsilon \in \mathrm{Dom}(\Phi)} |\widehat{\Phi}(\epsilon)|$ (since D is a consistent basis, all the relevant sets are finite so size(D) is well-defined). Thus A is the number of free region and effect variables in D, and A^2 is an upper bound for the sum of the cardinality of all the possible effect sets given by Φ.

Define the size of a constraint system $((\mathcal{C}, I, k), D)$, denoted $\text{size}((\mathcal{C}, I, k), D)$, to be a tuple (A, B, C) of natural numbers, where

$$\begin{array}{rcl} (A, B) & = & \text{size}(D) \\ C & = & 10 \times A^2 - 2X - 5Y - 3Z \\ & + & \text{the sum of the depth of all } \doteq \text{ and } \dot{\succeq}^i \text{ constraints in } \mathcal{C} \end{array}$$

and where

- X is the number of different $\alpha \dot{\succeq}^i \alpha$, $\rho \dot{\succeq}^i \rho$, and $\epsilon \dot{\succeq}^i \epsilon$ constraints in $\mathcal{C}$;
- Y is the number of different $\epsilon \dot{\supseteq} \rho$ and $\epsilon \dot{\supseteq} \overline{\epsilon'}$ constraints in $\mathcal{C}$;
- Z is the number of different $\alpha \doteq \alpha$, $\rho \doteq \rho'$, and $\epsilon \doteq \epsilon'$ constraints in $\mathcal{C}$.

The number $10 \times A^2$ is an upper bound for the sum $2X + 5Y + 3Z$; the X, Y, and Z can be thought of as the "good" constraints (leading to increased effect sets or to more and more variables becoming equal or monomorphic).

Theorem 74 (Rewriting is Strongly Normalizing) *Let $((\mathcal{C}_1, I_1, k), D_1)$ be a constraint system. There is no infinite sequence of rewriting steps*

$$(\mathcal{C}_1, I_1, k), D_1) \Longrightarrow_{\underline{S}_1} (\mathcal{C}_2, I_2, k), D_2) \Longrightarrow_{\underline{S}_2} \cdots$$

PROOF. Consider an arbitrary rewriting step

$$((\mathcal{C}, I, k), D) \Longrightarrow_{(Ri, \underline{S})} ((\mathcal{C}', I', k), D')$$

and let

$$\begin{array}{c} (A, B, C) = \text{size}((\mathcal{C}, I, k), D) \\ (A', B', C') = \text{size}((\mathcal{C}', I', k), D'). \end{array}$$

We now argue that $(A', B', C') < (A, B, C)$ in the standard lexicographic order on N^3 (where N denotes the set of natural numbers); since $<$ is a well-founded relation on N^3 this completes the proof. We proceed by cases on the rewriting rule applied.

R1. $A' = A$, $B' = B$, $C' < C$.
R2. $A' = A$, $B' = B$, $C' < C$. Here we have used that $\text{ML}(\tau_1) = \text{ML}(\tau_2)$ by well-taggedness of $\mathcal{C}$ and thus, as is easily shown by induction on $\text{ML}(\tau_1)$, that $\text{depth}(\tau_2 \doteq \tau_2) = \text{depth}(\tau_1 \doteq \tau_2) = \text{depth}(\tau_1 \doteq \tau_1)$.
R3. $A' < A$ (recall that by definition of constraint system $\text{frev}(\mathcal{C}) \subseteq \text{Dom}(D)$.
R4. $A' = A$, $B' = B$, $C' < C$.

R5. $A' < A$.
R6. $A' = A$, $B' = B$, $C' < C$.
R7. $A' = A$, $B' = B$, $C' < C$.
R8. $A' = A$, $B' = B$, $C' < C$.
R9. $A' = A$, $B' < B$.
R10. $A' = A$, $B' = B$, $C' < C$.
R11. $A' = A$, $B' = B$, $C' < C$.
R12. $A' = A$, $B' = B$, $C' < C$. $\square$

References

[1] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, 1977.

[2] L. Birkedal, M. Tofte, and M. Vejlstrup. From region inference to von Neumann machines via region representation inference. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 171–183. ACM Press, January 1996.

[3] J. M. Lucassen D. K. Gifford, P. Jouvelot and M.A. Sheldon. FX-87 reference manual. Technical Report MIT/LCS/TR-407, MIT Laboratory for Computer Science, Sept 1987.

[4] L. Damas and R. Milner. Principal type schemes for functional programs. In *Proc. 9th Annual ACM Symp. on Principles of Programming Languages*, pages 207–212, Jan. 1982.

[5] F. Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):253, April 1993.

[6] P. Jouvelot and D.K. Gifford. Algebraic reconstruction of types and effects. In *Proceedings of the 18th ACM Symposium on Principles of Programming Languages (POPL)*, 1991.

[7] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. Type reconstruction in the presence of polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):290–311, April 1993.

[8] R. Milner. A theory of type polymorphism in programming. *J. Computer and System Sciences*, 17:348–375, 1978.

[9] A. Mycroft. Polymorphic type schemes and recursive definitions. In *Proc. 6th Int. Conf. on Programming, LNCS 167*, 1984.

[10] P. Naur. Revised report on the algorithmic language Algol 60. *Comm. ACM*, 1:1–17, 1963.

[11] A.M. Pitts. Some notes on inductive and co-inductive techniques in the semantics of functional programs. BRICS Notes Series NS–94–5, BRICS, December 1994.

[12] D. Rémy. Extending ML type system with a sorted equational theory. Technical Report 1766, INRIA-Rocquencourt, BP 105, F-78 153 Le Chesnay Cedex, 1992.

[13] J-P. Talpin and P. Jouvelot. Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3), 1992.

[14] M. Tofte and L. Birkedal. A region inference algorithm. *ACM Transactions on Programming Languages and Systems*, 20(4):734–767, July 1998. (plus 24 pages of electronic appendix).

[15] M. Tofte, L. Birkedal, M. Elsman, N. Hallenberg, T.H. Olesen, P. Sestoft, and P. Bertelsen. Programming with regions in the ML Kit. Technical Report DIKU-TR-97/12, Dept. of Computer Science, University of Copenhagen, 1997. (http://www.diku.dk/research-groups/ topps/activities/kit2).

[16] M. Tofte and J-P. Talpin. Implementing the call-by-value lambda-calculus using a stack of regions. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 188–201. ACM Press, January 1994.

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

[18] P.R. Wilson. Uniprocessor garbage collection techniques. In Y. Bekkers and J. Cohen, editors, *Memory Management, Proceedings, International Workshop IWMM92*, pages 1–42. Springer-Verlag, September 1992.

Index of Notation and Definitions