Deciding Coproduct Equality with Focusing

Arbob Ahmad Daniel R. Licata Robert Harper
Carnegie Mellon University

May 10, 2010

Abstract

We describe a new decision procedure for equality in the simply-typed λ-calculi with finite product (pair) and coproduct (disjoint sum) types. Existing decision procedures for this problem use several techniques, such as normalization-by-evaluation and term rewriting. As this is a difficult problem that is both theoretically and practically interesting, we offer an alternative decision procedure, based on using the proof-theoretic notion of focusing to identify unique canonical representatives of equivalence classes. Focusing distinguishes this approach from prior work, and the proof of correctness employs straightforward structural induction arguments. Our decision procedure has two stages: First, we use Andreoli's notion of focusing to reduce the number of equivalent ways to write a λ-term, and then we compare focused terms for extensional equality. This second step is needed as focusing does not equate all terms that are coproduct-equal. For example, focusing does not dictate the order in which independent observations are made nor does it prevent redundant observations such as case analyzing the same variable twice. To decide the extensional equality of the focused terms, we employ higher-order focusing. Higher-order focusing provides a structured mechanism for generating all possible observations of a given type. This is possible since all of the types in our language are finitely inhabited so observations may be enumerated and performed. Therefore, two focused terms may be tested for equality by comparing them for each possible observation. Our decision procedure first translates the two terms into the higher order focusing language and then checks whether they are equal on all possible observations. We prove this alternative procedure for deciding equality is correct by showing it is complete and sound.

1 Introduction

In many applications of λ-calculi, it is essential to compare λ-terms for equality. For example, type checking a dependently-typed programming language requires deciding equality of the terms and types. While it is relatively simple to decide $\beta\eta$-equality for function and product types, deciding the full $\beta\eta$-equational theory for the finite simply-typed λ-calculus with coproducts (*i.e.*, disjoint sum types) is more complicated. We use the term *standard formulation* to refer to the λ-calculus with sum, product, and function types that are all finitely inhabited. The difficulty in deciding the equational theory for the standard formulation is evident in the coarsest η-rule (*i.e.*, the rule that equates as many terms as possible) for sum types, which is derived from the categorical universal mapping property for coproducts:

$$\begin{aligned}[t/x]t' \equiv \texttt{case}\ t\ \texttt{of}\ \texttt{inl}\ x_1 &\Rightarrow [\texttt{inl}\ x_1/x]t' \\ \mid \texttt{inr}\ x_2 &\Rightarrow [\texttt{inr}\ x_2/x]t' \qquad \text{if } t : \tau_1 + \tau_2\end{aligned}$$

The term t of sum type may appear multiple times in t', and each occurrence may be arbitrarily deeply nested within t'. In this paper we present a new algorithm for deciding definitional equality for the standard formulation.

Most existing decision procedures use the techniques of *normalization by evaluation* (NBE) [Altenkirch et al., 2001, Altenkirch and Uustalu, 2004, Balat et al., 2004] or *rewriting* [Ghani, 1995, Lindley, 2007]. Deciding coproduct equality is a difficult problem that is both theoretically and practically interesting. We offer a new approach that combines elements of both of these decision procedures. Unlike these other decision procedures, our approach employs *higher-order focusing* [Zeilberger, 2008] and *hereditary substitution* [Watkins et al., 2002]. This work provides another perspective to improve understanding of the coproduct equality problem by simplifying the correctness proof of the algorithm. The techniques of higher-order focusing and hereditary substitution facilitate a simple proof strategy using structural induction that avoids more mathematically sophisticated concepts from category theory.

Andreoli [1992] introduced focusing to restrict the number of nondeterministic choices made in a proof of a given proposition. The Curry-Howard interpretation of focusing in type theory restricts the number of terms of a given type. To limit the number of well-typed terms, focusing classifies the typing rules as *invertible* and *non-invertible* and restricts when and how the rules may be applied. The typing rules in Figure 1 illustrate the distinction between the two types of

Figure 1: Example typing rules

$$\frac{\gamma \vdash t_1 : \tau_1 \quad \gamma \vdash t_2 : \tau_2}{\gamma \vdash \langle t_1, t_2 \rangle : \tau_1 \times \tau_2} \times\mathtt{I} \qquad \frac{\gamma \vdash t : \tau_1 \times \tau_2}{\gamma \vdash \mathtt{fst}\, t : \tau_1} \times\mathtt{E1} \qquad \frac{\gamma \vdash t : \tau_1 \times \tau_2}{\gamma \vdash \mathtt{snd}\, t : \tau_2} \times\mathtt{E2}$$

typing rules. Invertible typing rules are those for which the conclusion implies the premises. For example, the conclusion of rule $\times\mathtt{I}$ implies the two premises because the first and second projections of the pair give terms of these types. Non-invertible typing rules are those for which the conclusion does not imply the premises. For example, the conclusion of rule $\times\mathtt{E1}$ does not imply its premise because the information about the second component of the term in the premise is lost. Therefore, the $\times\mathtt{I}$ rule is invertible, but the $\times\mathtt{E1}$ and $\times\mathtt{E2}$ rules are non-invertible.

Focusing assigns polarities to the types based on the distinction between invertible and non-invertible typing rules. The introduction rules for both the function and product types are invertible, and both types have non-invertible elimination rules. Focusing assigns negative polarity to types with this property. In contrast, the introduction rules for the sum type are non-invertible, and its elimination rule is invertible. Focusing assigns positive polarity to types with this property. The properties of the introduction and elimination rules of the types of each polarity form the basis for the focusing judgements to alternate between two phases, an inversion phase in which all possible invertible rules are applied and a focusing phase in which a particular chain of non-invertible rules is chosen. Higher-order focusing facilitates the application of all possible invertible rules by defining a term with a metafunction that maps each possible chain of rules to the resulting term.

The polarity of the type further divides the inversion phase into *right inversion* and *left inversion*. Right inversion introduces a term of negative type by applying all possible invertible introduction rules. In higher-order focusing, right inversion defines the behavior of the term with a metafunction that specifies the result of all possible observations for the given type. For example, the domain of the metafunction for a term of function type is any of the possible arguments to the function, and for each argument the metafunction specifies the term that is the result of applying the function to that argument. Left inversion eliminates a term

of positive type by applying all possible invertible elimination rules. In higher-order focusing, left inversion specifies the behavior on all possible results of the observation of the term of positive type.

The polarity of the type also divides the focusing phase into *right focus* and *left focus*. Right focus introduces a term of positive type by choosing a chain of non-invertible introduction rules. For example, a term of sum type specifies whether it gives a term of the type on the left or on the right of the sum type. The chain must continue applying non-invertible introduction rules until a term of negative type must be introduced. This term is introduced by right inversion. Left focus eliminates a term of negative type by choosing a chain of non-invertible elimination rules. The chain must continue applying non-invertible elimination rules until a term of positive type must be eliminated. This term is eliminated by left inversion.

The rigidity of higher-order focusing restricts the form of a term. The restrictions of focusing are illustrated by considering a program that represents the curried boolean "or" function where $\texttt{inl}\ \langle\rangle$ corresponds to true and $\texttt{inr}\ \langle\rangle$ corresponds to false. The following two programs are distinct possible representations of this function:

$$\begin{aligned}\lambda x : \mathbf{2}.\texttt{case}\ x\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow \lambda y : \mathbf{2}.\texttt{inl}\ \langle\rangle \\ \mid \texttt{inr}\ x_2 \Rightarrow \lambda y : \mathbf{2}.\texttt{case}\ y\ \texttt{of}\ \texttt{inl}\ y_1 \Rightarrow \texttt{inl}\ \langle\rangle \\ \mid \texttt{inr}\ y_2 \Rightarrow \texttt{inr}\ \langle\rangle\end{aligned}$$

and

$$\begin{aligned}\lambda x : \mathbf{2}.\lambda y : \mathbf{2}.\texttt{case}\ x\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow \texttt{inl}\ \langle\rangle \\ \mid \texttt{inr}\ x_2 \Rightarrow \texttt{case}\ y\ \texttt{of}\ \texttt{inl}\ y_1 \Rightarrow \texttt{inl}\ \langle\rangle \\ \mid \texttt{inr}\ y_2 \Rightarrow \texttt{inr}\ \langle\rangle\end{aligned}$$

As both of these programs represent the same function, they should be equal. We translate both programs to the same focusing term. The translation rearranges the first term since it applies an elimination rule (*i.e.*, the rule for eliminating sums) before all possible invertible introduction rules (*i.e.*, the two function introduction rules) have been applied. This corresponds to the fact that a term of negative type is introduced by right inversion so all possible invertible introduction rules must be applied before any elimination rules may be applied. In this way, the number of valid programs for the boolean "or" function is reduced.

The translation of the standard formulation to focusing employs a substitution judgement that preserves the invariants of focusing. Well-formed focusing

terms cannot have β-redices within the term. Ordinary substitution does not preserve this property. Substituting a well-formed term for a free variable in another well-formed term may not result in a well-formed term as the following example demonstrates:

$$[\langle\langle\rangle, \langle\rangle\rangle/x]\texttt{fst}\ x = \texttt{fst}\ \langle\langle\rangle, \langle\rangle\rangle$$

In this example, the substitution produces a term with a newly introduced β-redex. We need a substitution that does not introduce β-redices. To achieve this we use hereditary substitution, which performs any additional reduction required as a result of the substitution. In the above example, hereditary substitution reduces the projection leaving the first component of the pair. This maintains the structure of the focusing term.

We prove the correctness of our algorithm in two steps. First, we show that the equality we use to compare two focusing terms is decidable. The restrictions of focusing are not rigid enough to force all definitionally equal terms to translate to the same unique canonical focusing term. Therefore, we cannot simply translate two terms in the standard formulation into the higher-order focusing language and check that they are syntactically identical. Instead we use an equality on focusing terms. The equality we first propose is not obviously decidable because it compares two focusing terms under a possibly infinite set of observations; however, it is in fact decidable since it is equivalent to a second equality that only compares two focusing terms under a finite subset of the observations. Second, we show that our algorithm that translates standard formulation terms to focusing terms and checks this equality corresponds to definitional equality. To do this we must show completeness and soundness of our algorithm.

2 Related Work

We propose a new algorithm for deciding coproduct equality, but parts of our algorithm are similar to parts of both NBE and rewriting. Both of these techniques provide a valid solution to the problem. Our algorithm is intended as an alternative.

In NBE, terms in the standard formulation are interpreted in a finite model or denotational semantics. Equal terms have identical interpretations in the model. As with our approach, the proof of correctness is completely constructive. The interpretation of terms into the finite model is analogous to the translation in our approach with the finite model corresponding to our higher-order focusing language.

The finite model has simultaneous case splits on all necessary observations so that only identical elements of the model are equal. As our language only focuses on one observation at a time, we compare focusing terms for extensional equality. This allows us to avoid considering sophisticated category theory concepts such as sheaves over environments, which are used in NBE.

The rewriting algorithm of [Lindley, 2007] defines a clever series of term rewriting techniques to arrive at normal forms. The terms are considered modulo a congruence to resolve distinctions based on the order of observations rather than employing a simultaneous case split. In our setting, the terms that would be equated by the axioms of the rewriting algorithm are either resolved by focusing or by extensional equality. In particular, terms equated by one of the axioms corresponding to β-reduction are identified by the translation into the focusing language, and the terms equated by the three axioms corresponding to the η rule for sums (*i.e.*, *move-case*, *repeated-guard*, and *redundant-guard*) are translated to extensionally equal focusing terms. Unlike our approach, the proof of confluence of the rewrite rules requires a logical relations argument.

3 Standard formulation

Our algorithm is for the standard formulation of the finite simply-typed λ-calculus with sums that is presented in Figure 2. The context of typing assumptions is denoted by γ and types are indicated by τ. Note that `abort`, `inl`, and `inr` are tagged with their type for the purpose of type synthesis. The remaining typing rules should be relatively familiar.

The definitional equality that we are deciding for the standard formulation is presented in Figure 3. The first three rules state the reflexivity, transitivity, and symmetry of this equivalence relation. There are also two compatibility rules that state the applications of a term to equal arguments are equal and two functions are equal if their bodies are equal. It is not necessary to state the more general equality that two applications are equal if the respective function and argument terms are equal as this is derivable from the other rules. To see this consider the function, $\lambda x : \tau_1 \to \tau_2.x\ t_1$. Using this function and the compatibility rule for applications on arguments and the β rule for functions along with symmetry and transitivity, if we assume $\gamma \vdash t = t' : \tau_1 \to \tau_2$ we can show $\gamma \vdash t\ t_1 = t'\ t_1 : \tau_2$. Now assuming $\gamma \vdash t_1 = t_1' : \tau_1$ we can use the compatibility rules again to show $\gamma \vdash t'\ t_1 = t'\ t_1' : \tau_2$. Putting these equalities together using transitivity, we get a derivation of the more general compatibility equality for applications.

Figure 2: Standard formulation

$$\frac{}{\gamma, x : \tau, \gamma' \vdash x : \tau}$$

$$\frac{}{\gamma \vdash \langle\rangle : 1} \qquad \frac{\gamma \vdash t_1 : \tau_1 \quad \gamma \vdash t_2 : \tau_2}{\gamma \vdash \langle t_1, t_2 \rangle : \tau_1 \times \tau_2} \qquad \frac{\gamma \vdash t : \tau_1 \times \tau_2}{\gamma \vdash \mathtt{fst}\ t : \tau_1} \qquad \frac{\gamma \vdash t : \tau_1 \times \tau_2}{\gamma \vdash \mathtt{snd}\ t : \tau_2}$$

$$\frac{\gamma, x : \tau_1 \vdash t : \tau}{\gamma \vdash \lambda x : \tau_1 . t : \tau_1 \to \tau} \qquad \frac{\gamma \vdash t : \tau_2 \to \tau \quad \gamma \vdash t_2 : \tau_2}{\gamma \vdash t\ t_2 : \tau}$$

$$\frac{\gamma \vdash t : 0}{\gamma \vdash \mathtt{abort}^{\tau}\ t : \tau} \qquad \frac{\gamma \vdash t_1 : \tau_1}{\gamma \vdash \mathtt{inl}^{\tau_1+\tau_2}\ t_1 : \tau_1 + \tau_2} \qquad \frac{\gamma \vdash t_2 : \tau_2}{\gamma \vdash \mathtt{inr}^{\tau_1+\tau_2}\ t_2 : \tau_1 + \tau_2}$$

$$\frac{\gamma \vdash t : \tau_1 + \tau_2 \quad \gamma, x_i : \tau_i \vdash t_i : \tau \quad (i = 1, 2)}{\gamma \vdash \mathtt{case}(t, x_1.t_1, x_2.t_2) : \tau}$$

At first it may appear that additional compatibility rules are necessary, but these are actually derivable from the rules given and omitting them simplifies several of the later proofs. For example there is no need for the following compatibility rule:

$$\frac{\gamma \vdash t_1 = t_1' : \tau_1 \quad \gamma \vdash t_2 = t_2' : \tau_2}{\gamma \vdash \langle t_1, t_2 \rangle = \langle t_1', t_2' \rangle : \tau_1 \times \tau_2}$$

This rule can be derived using the function $\lambda x : \tau_1 . \lambda y : \tau_2 . \langle x, y \rangle$ and using the given compatibility and β rules as in the above derivation.

The remaining rules correspond to the $\beta\eta$-equalities of the different types. The η-equality for the sum type, which was given above in concrete syntax is of particular interest. The type annotations on the `inl` and `inr` have been omitted for brevity. Again, note that the term of sum type, t, may occur zero or more times in t' and may be arbitrarily deeply nested within the term. This is the major difficulty in deciding this equality.

The η-equality for the empty type may appear strange but can be better understood as the nullary version of the η-equality for the sum type. It states that in a context where there is a term of type 0, any well-typed term is equivalent to an `abort` at that term's type. From a logic perspective, this can be seen as equating

Figure 3: Definitional equality

$$\frac{\gamma \vdash t : \tau}{\gamma \vdash t = t : \tau} \qquad \frac{\gamma \vdash t' = t : \tau}{\gamma \vdash t = t' : \tau} \qquad \frac{\gamma \vdash t_1 = t_2 : \tau \qquad \gamma \vdash t_2 = t_3 : \tau}{\gamma \vdash t_1 = t_3 : \tau}$$

$$\frac{\gamma \vdash t : 1}{\gamma \vdash t = \langle\rangle : 1}$$

$$\frac{\gamma \vdash \langle t_1, t_2\rangle : \tau_1 \times \tau_2}{\gamma \vdash \mathtt{fst}\ \langle t_1, t_2\rangle = t_1 : \tau_1} \qquad \frac{\gamma \vdash \langle t_1, t_2\rangle : \tau_1 \times \tau_2}{\gamma \vdash \mathtt{snd}\ \langle t_1, t_2\rangle = t_2 : \tau_2}$$

$$\frac{\gamma \vdash t : \tau_1 \times \tau_2}{\gamma \vdash t = \langle \mathtt{fst}\ t, \mathtt{snd}\ t\rangle : \tau_1 \times \tau_2}$$

$$\frac{\gamma \vdash (\lambda x : \tau_1.t)\ t_1 : \tau}{\gamma \vdash (\lambda x : \tau_1.t)\ t_1 = [t_1/x]t : \tau} \qquad \frac{\gamma \vdash t : \tau_1 \to \tau_2 \quad x\#t}{\gamma \vdash t = \lambda x : \tau_1.t\ x : \tau_1 \to \tau_2}$$

$$\frac{\gamma \vdash t : \tau_1 \to \tau_2 \quad \gamma \vdash t_1 = t_1' : \tau_1}{\gamma \vdash t\ t_1 = t\ t_1' : \tau_2} \qquad \frac{\gamma, x : \tau_1 \vdash t = t' : \tau_2}{\gamma \vdash \lambda x : \tau_1.t = \lambda x : \tau_1.t' : \tau_1 \to \tau_2}$$

$$\frac{\gamma \vdash t_0 : 0 \quad \gamma \vdash t : \tau}{\gamma \vdash \mathtt{abort}^{\tau}\ t_0 = t : \tau}$$

$$\frac{\gamma \vdash t : \tau_1 \quad \gamma, x_i : \tau_i \vdash t_i : \tau \quad (i = 1, 2)}{\gamma \vdash \mathtt{case}(\mathtt{inl}^{\tau_1+\tau_2}\ t, x_1.t_1, x_2.t_2) = [t/x_1]t_1 : \tau}$$

$$\frac{\gamma \vdash t : \tau_2 \quad \gamma, x_i : \tau_i \vdash t_i : \tau \quad (i = 1, 2)}{\gamma \vdash \mathtt{case}(\mathtt{inr}^{\tau_1+\tau_2}\ t, x_1.t_1, x_2.t_2) = [t/x_2]t_2 : \tau}$$

$$\frac{\gamma \vdash t : \tau_1 + \tau_2 \quad \gamma, x : \tau_1 + \tau_2 \vdash t' : \tau}{\gamma \vdash \mathtt{case}(t, x_1.[\mathtt{inl}\ x_1/x]t', x_2.[\mathtt{inr}\ x_2/x]t') = [t/x]t' : \tau}$$

all proofs of a given proposition once a contradiction has been reached.

4 Focusing language

In this section, we present the focusing language that we use in our algorithm. The language is based on the focusing language presented in Licata et al. [2008]. We similarly define our focusing language in two stages. We first define the positive and negative types through the typing judgements of constructor and destructor patterns, respectively, and then give the general focusing framework. We then describe several other judgements in the focusing language including substitution and extensional equality.

4.1 Pattern Typing Judgements

The typing judgements of the constructor and destructor patterns are presented in Figure 4. We write A^+, B^+ and A^-, B^- to stand for positive and negative types, respectively. Explicit shifts denoted by $\downarrow A^-$ and $\uparrow A^+$ coerce a type from one polarity to another. The judgement $\Delta \Vdash p :: A^+$ defines the positive types by how they are introduced. The linear context, Δ, is the only output of this judgement. We call this a linear context as the pattern typing judgements mirror those of linear logic. For example, note the splitting of the context in the rule for the function type. As there are no constructor patterns for the empty type, in this setting the context for a constructor pattern will always contain exactly one variable since a constructor pattern must end in a variable when a shifted negative type, $\downarrow A^-$, is reached. The judgement $\Delta \Vdash d :: A^- > B^+$ defines the negative connectives by how they are decomposed. The linear context and the positive type B^+ are outputs. There is no destructor pattern for the type $\top$ just as there is no constructor pattern for the empty type. The destructor patterns for the product type, $A^- \,\&\, B^-$, are the familiar projections. The destructor pattern for the function type, $A^+ \rightarrow B^-$, consists of a constructor pattern for its argument of type A^+ and a destructor pattern to further decompose the result of type B^-. Every destructor pattern ends when it reaches a shifted positive type, $\uparrow A^+$, and the type A^+ is returned.

As noted in Licata et al. [2008], the typing judgements of the patterns yield a *strict subformula ordering* on the types of the focusing language.

Definition 4.1. *(Strict subformula ordering). We define a well-founded ordering ($<$) between the types as the least relation closed under transitivity and the following properties:*

Figure 4: Focusing language patterns

Neg. types	A^-, B^-	$::=$	$A^- \mathbin{\&} B^- \mid \top \mid A^+ \to B^- \mid \uparrow A^+$
Pos. types	A^+, B^+	$::=$	$A^+ \oplus B^+ \mid 0 \mid \downarrow A^-$
Linear context	Δ	$::=$	$\cdot \mid \Delta, x : A^-$
Constructor pattern	p	$::=$	$x \mid \texttt{inl}\ p \mid \texttt{inr}\ p$
Destructor pattern	d	$::=$	$\epsilon \mid p; d \mid \texttt{fst}; d \mid \texttt{snd}; d$

$\boxed{\Delta \Vdash p :: A^+}$

$$\frac{\Delta \Vdash p :: A_1^+}{\Delta \Vdash \texttt{inl}\ p :: A_1^+ \oplus A_2^+}$$

$$\frac{\Delta \Vdash p :: A_2^+}{\Delta \Vdash \texttt{inr}\ p :: A_1^+ \oplus A_2^+}$$

$$\frac{}{x : A^- \Vdash x ::\downarrow A^-}$$

$\boxed{\Delta \Vdash d :: A^- > B^+}$

$$\frac{}{\cdot \Vdash \epsilon ::\uparrow A^+ > A^+}$$

$$\frac{\Delta_1 \Vdash p_1 :: A_1^+ \quad \Delta_2 \Vdash d :: A_2^- > B^+}{\Delta_1, \Delta_2 \Vdash p_1; d :: A_1^+ \to A_2^- > B^+}$$

$$\frac{\Delta \Vdash d :: A_1^- > B^+}{\Delta \Vdash \texttt{fst}; d :: A_1^- \mathbin{\&} A_2^- > B^+}$$

$$\frac{\Delta \Vdash d :: A_2^- > B^+}{\Delta \Vdash \texttt{snd}; d :: A_1^- \mathbin{\&} A_2^- > B^+}$$

Figure 5: Focusing language grammar

Pos. value	v^+	$::=$	$p[\sigma]$
Neg. value	v^-	$::=$	$\texttt{val}^-(\phi^-)$
			where $\phi^- ::= \{d_1 \mapsto e_1 \mid \cdots \mid d_n \mapsto e_n\}$
Expression	e	$::=$	$v^+ \mid x \bullet k^-$
Neg. Continuation	k^-	$::=$	$d[\sigma]; k^+$
Pos. Continuation	k^+	$::=$	$\texttt{cont}^+(\phi^+)$
			where $\phi^+ ::= \{p_1 \mapsto e_1 \mid \cdots \mid p_n \mapsto e_n\}$
Substitution	σ	$::=$	$\cdot \mid \sigma, v^-/x$
Unrestricted context	Γ	$::=$	$\cdot \mid \Gamma, \Delta$

- *If* $\Delta \Vdash d :: A_2^- > B^+$ *and* $A_1^- \in \Delta$ *then* $A_1^- < A_2^-$
- *If* $\Delta \Vdash d :: A_2^- > A_1^+$ *then* $A_1^+ < A_2^-$
- *If* $\Delta \Vdash p :: A_2^+$ *and* $A_1^- \in \Delta$ *then* $A_1^- < A_2^+$

We will frequently make use of this ordering in induction arguments.

4.2 Focusing Typing Judgements

The focusing terms are defined in Figure 5, and the focusing typing rules are presented in Figure 6. The unrestricted context Γ allows variables to be observed multiple times. Therefore, even within the focusing language there are multiple ways to write equivalent terms by making redundant or unnecessary observations on a variable. The typing rules for the focusing terms are divided into those for the six categories of terms, namely positive values, negative values, expressions, negative continuations, positive continuations, and substitutions.

The positive value judgement focuses on a particular constructor pattern and gives a substitution which maps its free variables to negative values. In contrast, the negative value judgement inverts on all destructor patterns of the given type. To each pattern decomposing A^- into B^+ in a context Δ, it assigns an expression of type B^+ in the unrestricted context extended with Δ. To represent this mapping we use the metafunction, ϕ^-, that maps destructor patterns to expressions. Note that there are only a finite number of destructor patterns for a given type so these

Figure 6: Focusing language terms

$\boxed{\Gamma \vdash v^+ :: A^+}$

$$\frac{\Delta \Vdash p :: A^+ \quad \Gamma \vdash \sigma : \Delta}{\Gamma \vdash p[\sigma] :: A^+}$$

$\boxed{\Gamma \vdash v^- : A^-}$

$$\frac{\forall (\Delta \Vdash d :: A^- > B^+) : \quad \Gamma, \Delta \vdash \phi^-(d) : B^+}{\Gamma \vdash \mathtt{val}^-(\phi^-) : A^-}$$

$\boxed{\Gamma \vdash e : A^+}$

$$\frac{\Gamma \vdash v^+ :: A^+}{\Gamma \vdash v^+ : A^+} \qquad \frac{x : A^- \in \Gamma \quad \Gamma \vdash k^- : A^- > B^+}{\Gamma \vdash x \bullet k^- : B^+}$$

$\boxed{\Gamma \vdash k^- : A^- > B^+}$

$$\frac{\Delta \Vdash d :: A^- > A_0^+ \quad \Gamma \vdash \sigma : \Delta \quad \Gamma \vdash k^+ : A_0^+ > B^+}{\Gamma \vdash d[\sigma]; k^+ : A^- > B^+}$$

$\boxed{\Gamma \vdash k^+ : A^+ > B^+}$

$$\frac{\forall (\Delta \Vdash p :: A^+) : \quad \Gamma, \Delta \vdash \phi^+(p) : B^+}{\Gamma \vdash \mathtt{cont}^+(\phi^+) : A^+ > B^+}$$

$\boxed{\Gamma \vdash \sigma : \Delta}$

$$\frac{}{\Gamma \vdash \cdot : \cdot} \qquad \frac{\Gamma \vdash \sigma : \Delta \quad \Gamma \vdash v^- : A^-}{\Gamma \vdash \sigma, v^-/x : \Delta, x : A^-}$$

functions are always defined on a finite domain. Therefore, the universal quantification in the premise of this rule corresponds to a finite number of premises.

An expression is either a positive value or a negative continuation that focuses on a single assumption in the unrestricted context. The continuations reason from some type to a resulting positive type. Negative continuations are dual to positive values as they correspond to left and right focus, respectively. A negative continuation focuses on an assumption by choosing a destructor pattern that decomposes the type together with a substitution for the pattern's free variables just as a positive value chooses a constructor pattern and substitution. However, the negative continuation also has a positive continuation from the resulting positive type of the destructor pattern to the final positive type. Positive continuations are similarly dual to negative values as they correspond to left and right inversion, respectively. A positive continuation inverts on all constructor patterns for the given type and assigns each of them to an expression of the desired type in the expanded unrestricted context.

Finally, a substitution simply associates a negative value of appropriate type to each variable. One of the elegant features of this framework is that these six typing judgements are completely independent of the particular positive and negative types we include in the language, making it robust.

4.3 η-expansion and identity substitutions and continuations

The restrictions on focusing terms makes the use of variables in the context more complex. Consequently, it is not immediately apparent how to construct a negative value of type A^- in a context containing $x : A^-$. The judgement for constructing such a negative value is presented in Figure 7. The invariant is that if $\texttt{id}\ x : A^- = v^-$ then $x : A^- \vdash v^- : A^-$. It relies on the judgements $\texttt{id}\ \Delta = \sigma$, which generates a substitution such that $\Delta \vdash \sigma : \Delta$, and $\texttt{id}\ A^+ = k^+$, which forms a positive continuation such that $\cdot \vdash k^+ : A^+ > A^+$.

To η-expand a variable, a negative value is created that assigns each negative pattern for the given type to an expression that performs that observation on the variable using the identity substitution and the identity positive continuation. We will use $\texttt{id}_\Delta$ to stand for the result of this operation on the context, Δ. The identity positive continuation assigns each constructor pattern to the positive value consisting of the same constructor pattern and the identity substitution. Componentwise η-expansion forms the identity substitution. Combining this information, we have the following lemma about these judgements:

Figure 7: η-expansion and identity

$\boxed{\mathtt{id}\ x : A^- = v^-}$

$$\frac{\forall(\Delta \Vdash d :: A^- > B^+).\ \forall\sigma.\ \forall k^+.\ (\mathtt{id}\ \Delta = \sigma, \mathtt{id}\ B^+ = k^+) : \quad \phi^-(d) := x \bullet d[\sigma]; k^+}{\mathtt{id}\ x : A^- = \mathtt{val}^-(\phi^-)}$$

$\boxed{\mathtt{id}\ A^+ = k^+}$

$$\frac{\forall(\Delta \Vdash p :: A^+).\ \forall\sigma.\ (\mathtt{id}\ \Delta = \sigma) : \quad \phi^+(p) := p[\sigma]}{\mathtt{id}\ A^+ = \mathtt{cont}^+(\phi^+)}$$

$\boxed{\mathtt{id}\ \Delta = \sigma}$

$$\frac{}{\mathtt{id}\ \cdot = \cdot} \qquad \frac{\mathtt{id}\ \Delta = \sigma \quad \mathtt{id}\ x : A^- = v^-}{\mathtt{id}\ \Delta, x : A^- = \sigma, v^-/x}$$

Lemma 4.1. *(η-expansion and identity)*

1. *If* $\texttt{id}\ x : A^- = v^-$ *then* $x : A^- \vdash v^- : A^-$
2. *If* $\texttt{id}\ \Delta = \sigma$ *then* $\Delta \vdash \sigma : \Delta$
3. *If* $\texttt{id}\ A^+ = k^+$ *then* $\cdot \vdash k^+ : A^+ > A^+$

Proof. Follows from straightforward induction on A^- in 1, Δ in 2, and A^+ in 3. For example, in the first case, according to the strict subformula ordering, all types in the context, Δ, as well as the type B^+ are smaller than the type A^- in the rule for η-expansion so the inductive calls to the other cases of the proof are justified. A weakening lemma is required in this proof. □

4.4 Substitution

The key substitution judgements are shown in Figure 8. The interesting case of substituting into an expression occurs when the variable observed by the expression is present in the simultaneous substitution. In this case the corresponding negative value in the substitution is cut against the negative continuation using a leftist substitution. The leftist substitution is defined inductively on the term that is being substituted rather than the term that is being substituted into as is the case for the usual substitution judgement. The other interesting cases occur when a negative value is substituted into a negative continuation and when a positive value is substituted into a positive continuation. In both cases, the metafunction is applied to the pattern and the substitution corresponding to the observation or positive value, respectively, is performed to eliminate the free variables from the expression. In the negative case, the resulting expression is then composed with the positive continuation.

The rest of the simultaneous substitution rules are given in the appendix. These rules just apply the substitution to the subterms. Note that no substitution is performed on patterns as their free variables are assigned values in the accompanying substitution. The rules for composing terms are also given in the appendix. These rules are defined by compositions of the subterms until an interesting cut is reached. For example, an expression that observes a variable is composed with a positive continuation by composing the associated negative continuation of the expression with that positive continuation.

These judgements yield the following substitution lemma:

Lemma 4.2. *(substitution)*

Figure 8: Key substitution judgements

$\boxed{[\sigma]_\Delta e = e'}$

$$\frac{[\sigma]^{\mathrm{v}}_\Delta v^+ = v_0^+}{[\sigma]_\Delta v^+ = v_0^+} \qquad \frac{x \notin \sigma \quad [\sigma]_\Delta k^- = k_0^-}{[\sigma]_\Delta x \bullet k^- = x \bullet k_0^-}$$

$$\frac{((v^-/x : A^-) \in \sigma : \Delta) \quad [\sigma]_\Delta k^- = k_0^- \quad \langle v^- \rangle_{A^-} k_0^- = e}{[\sigma]_\Delta x \bullet k^- = e}$$

$\boxed{\langle v^- \rangle_{A^-} k^- = e}$

$$\frac{\Delta \Vdash d :: A^- > B^+ \quad [\sigma]_\Delta \phi^-(d) = e \quad (\mathtt{case}\ e : B^+\ \mathtt{of}\ k^+) = e'}{\langle \mathtt{val}^-(\phi^-) \rangle_{A^-}(d[\sigma]; k^+) = e'}$$

$\boxed{\langle v^+ \rangle_{A^+} k^+ = e}$

$$\frac{\Delta \Vdash p :: A^+ \quad [\sigma]_\Delta \phi^+(p) = e}{\langle p[\sigma] \rangle_{A^+} \mathtt{cont}^+(\phi^+) = e}$$

1. *If* $\Gamma, \Delta \vdash e : A^+$, $\Gamma \vdash \sigma : \Delta$, *and* $[\sigma]_\Delta e = e'$ *then* $\Gamma \vdash e' : A^+$
2. *If* $\Gamma, \Delta \vdash v^+ :: A^+$, $\Gamma \vdash \sigma : \Delta$, *and* $[\sigma]^{\mathtt{v}}_\Delta v^+ = v_0^+$ *then* $\Gamma \vdash v_0^+ :: A^+$
3. *If* $\Gamma, \Delta \vdash v^- : A^-$, $\Gamma \vdash \sigma : \Delta$, *and* $[\sigma]_\Delta v^- = v_0^-$ *then* $\Gamma \vdash v_0^- : A^-$
4. *If* $\Gamma, \Delta \vdash k^- : A^- > B^+$, $\Gamma \vdash \sigma : \Delta$, *and* $[\sigma]_\Delta k^- = k_0^-$ *then* $\Gamma \vdash k_0^- : A^- > B^+$
5. *If* $\Gamma, \Delta \vdash k^+ : A^+ > B^+$, $\Gamma \vdash \sigma : \Delta$, *and* $[\sigma]_\Delta k^+ = k_0^+$ *then* $\Gamma \vdash k_0^+ : A^+ > B^+$
6. *If* $\Gamma, \Delta \vdash \sigma_0 : \Delta_0$, $\Gamma \vdash \sigma : \Delta$, *and* $[\sigma]_\Delta \sigma_0 = \sigma_0'$ *then* $\Gamma \vdash \sigma_0' : \Delta_0$
7. *If* $\Gamma \vdash v^- : A^-$, $\Gamma \vdash k^- : A^- > B^+$, *and* $\langle v^- \rangle_{A^-} k^- = e$ *then* $\Gamma \vdash e : B^+$
8. *If* $\Gamma \vdash v^+ :: A^+$, $\Gamma \vdash k^+ : A^+ > B^+$, *and* $\langle v^+ \rangle_{A^+} k^+ = e$ *then* $\Gamma \vdash e : B^+$
9. *If* $\Gamma \vdash e : A^+$, $\Gamma \vdash k^+ : A^+ > B^+$, *and* $\mathtt{case}\ e : A^+\ \mathtt{of}\ k^+ = e'$ *then* $\Gamma \vdash e' : B^+$
10. *If* $\Gamma \vdash k^- : A_1^- > A^+$, $\Gamma \vdash k^+ : A^+ > B^+$, *and* $k^- \circ_{A^+} k^+ = k_0^-$ *then* $\Gamma \vdash k_0^- : A_1^- > B^+$
11. *If* $\Gamma \vdash k^+ : A_1^+ > A^+$, $\Gamma \vdash k_0^+ : A^+ > B^+$, *and* $k^+ \circ_{A^+} k_0^+ = k_1^+$ *then* $\Gamma \vdash k_0^+ : A_1^+ > B^+$

Proof. The proof is by induction on the relevant types, terms, and contexts. It again uses the strict subformula ordering. For example, in the case for leftist substitution of a negative value into a negative continuation, the types in the context, Δ, and type, B^+, at which the substitutions in the premises are performed are both less than A^- in the strict subformula ordering. □

The following lemma relating substitution to the η-expansion and identity judgements expresses the expected behavior of the identity substitution and continuation:

Lemma 4.3. *(substitution-identity)*

1. *If* $\mathtt{id}\ x : A^- = v^-$, $\Gamma \vdash \sigma : \Delta$, *and* $v_0^-/x : A^- \in \sigma : \Delta$ *then* $[\sigma]_\Delta v^- = v_0^-$
2. *If* $\mathtt{id}\ \Delta = \sigma$ *and* $\Gamma \vdash \sigma_0 : \Delta$ *then* $[\sigma_0]_\Delta \sigma = \sigma_0$

3. *If* $\mathtt{id}\ \Delta = \sigma$ *and* $\Gamma, \Delta \vdash e : A^+$ *then* $[\sigma]_\Delta e = e$.

4. *Similarly for the remaining simultaneous substitution judgements*

5. *If* $\mathtt{id}\ A^+ = k^+$ *and* $\Gamma \vdash k_0^+ : A^+ > B^+$ *then* $k^+ \circ_{A^+} k_0^+ = k_0^+$

Proof. We use induction on the relevant types and contexts. For example, in case 5, the induction hypothesis corresponding to case 3 is appealed to at a context less than A^+. □

4.5 Extensional equality

The extensional equality for terms in the focusing language is presented in Figure 9. The equality of expressions is defined as equality of all closed instances of those expressions as mentioned above. Note that a closed expression is necessarily a positive value as there are no variables to observe in the empty context. Therefore, the equality of the closed instances is checked as equality of positive values. Two positive values are equal if they both choose the same constructor pattern and their respective substitutions for the free variables of that pattern are equal. Equality of substitutions is checked componentwise. Equality of negative values is verified by checking the equality of the expressions in the image of the metafunction for each destructor pattern.

To see that this equality is well defined, we must look at the types and contexts to verify that they get smaller. The extensional equality for expressions relies on the equality of positive values at the same type in the empty context. The equality of positive values depends on the equality of a substitution for a context that is smaller according to the strict subformula ordering. Therefore, the negative types in this context are smaller than the positive type that we started with. So the destructor patterns for this type produce contexts and types that are again smaller than the original type. Consequently, when we go full circle back to the extensional equality of expressions, the context and type are smaller than the original type. So this definition is well founded.

Note that it is not immediately apparent that extensional equality of focusing terms is decidable. When we quantify over all substitutions for a context, this is a potentially infinite set as there are many distinct ways to represent equivalent substitutions. For example, an arbitrary number of redundant observations of a variable may be added to create multiple equivalent substitutions.

It must also be shown that this equality actually corresponds to a congruence. That is, it interacts with substitution as expected. To show this, we will also need

Figure 9: Extensional equality of focusing terms

$\boxed{\Gamma \vdash e_1 \equiv e_2 : A^+}$

$$\frac{\forall(\sigma : \Gamma) : \quad \cdot \vdash [\sigma]e_1 \equiv [\sigma]e_2 :: A^+}{\Gamma \vdash e_1 \equiv e_2 : A^+}$$

$\boxed{\Gamma \vdash v_1^+ \equiv v_2^+ :: A^+}$

$$\frac{\Delta \Vdash p :: A^+ \quad \Gamma \vdash \sigma_1 \equiv \sigma_2 : \Delta}{\Gamma \vdash p[\sigma_1] \equiv p[\sigma_2] :: A^+}$$

$\boxed{\Gamma \vdash v_1^- \equiv v_2^- : A^-}$

$$\frac{\forall(\Delta \Vdash d :: A^- > B^+) : \quad \Gamma, \Delta \vdash \phi_1^-(d) \equiv \phi_2^-(d) : B^+}{\Gamma \vdash \mathtt{val}^-(\phi_1^-) \equiv \mathtt{val}^-(\phi_2^-) : A^-}$$

$\boxed{\Gamma \vdash \sigma_1 \equiv \sigma_2 : \Delta}$

$$\frac{\Gamma \vdash v_1^- \equiv v_2^- : A^- \quad \Gamma \vdash \sigma_1 \equiv \sigma_2 : \Delta}{\Gamma \vdash \sigma_1, v_1^-/x \equiv \sigma_2, v_2^-/x : \Delta, x : A^-} \qquad \frac{}{\Gamma \vdash \cdot \equiv \cdot : \cdot}$$

to know that the various substitution judgements commute with one another. That is, the following lemma holds:

Lemma 4.4. *(substitution commutes)*

1. *If* $\Gamma, \Delta \vdash e : A_0^+$, $\Gamma, \Delta \vdash k^+ : A_0^+ > A^+$, $\Gamma \vdash \sigma : \Delta$, $[\sigma]_\Delta e = e_\mathrm{s}$, $[\sigma]_\Delta k^+ = k_\mathrm{s}^+$, *and* $\mathtt{case}\ e : A_0^+\ \mathtt{of}\ k^+ = e'$ *then* $\Gamma \vdash [\sigma]_\Delta e' \equiv \mathtt{case}\ e_\mathrm{s} : A_0^+\ \mathtt{of}\ k_\mathrm{s}^+ : A^+$

2. *All corresponding cases of simultaneous substitution into a leftist substitution.*

3. *If* $\Gamma \vdash e : A_1^+$, $\Gamma \vdash k_1^+ : A_1^+ > A_0^+$, *and* $\Gamma \vdash k^+ : A_0^+ > A^+$ *then* $\Gamma \vdash e_1 \equiv e_2 : A^+$ *where* $e_1 := \mathtt{case}\ (\mathtt{case}\ e : A_1^+\ \mathtt{of}\ k_1^+) : A_0^+\ \mathtt{of}\ k^+$ *and* $e_2 := \mathtt{case}\ e : A_1^+\ \mathtt{of}\ (k_1^+ \circ_{A_0^+} k^+)$

4. *All corresponding cases of composing cuts.*

Proof. The proof is complicated by the fact that in several cases, multiple induction hypotheses are applied resulting in an intricate induction order. □

Now, using this lemma, we can show functionality, namely, that extensional equality is a congruence.

Lemma 4.5. *(functionality)*

1. *If* $\Gamma, \Delta \vdash e_1 \equiv e_2 : A^+$ *and* $\Gamma \vdash \sigma_1 \equiv \sigma_2 : \Delta$ *then* $\Gamma \vdash [\sigma_1]_\Delta e_1 \equiv [\sigma_2]_\Delta e_2 : A^+$

2. *Similarly for the other substitution judgements*

Proof. The proof is by lexicographic induction. First, we induct on the critical types and contexts. Then, we induct on the terms. Finally, we induct on the ordering of the cases. The proof relies on the previous lemma and certain properties of simultaneous substitution. □

5 Translation

To compare terms in the standard formulation, we define a translation to the higher-order focusing language. The translation is specified in two parts, one for the types and another for the terms. Both of these parts are split into positive and

negative translations. A type τ can be translated to either a positive type A^+ or a negative type A^-, and similarly a term can be translated to either an expression or a negative value.

We begin by describing the type translation as this is simpler than the term translation but follows the same pattern. The classification of types as positive and negative as specified in Figure 10 informs the type translation given in Figure 11 yielding positive and negative translation judgements. This classification looks only at the outermost type constructor. Both the positive and negative translations are defined componentwise on types with connectives. The translation of the type consists of the translations of its components. Both translations appeal to the other judgement when the type has the wrong polarity. For example, the negative translation of a function type refers to both translation judgements in its premises. The argument type of the function is translated with the positive judgement, and the result type is translated with the negative judgement. This corresponds to the expected polarities for a function type in the focusing language. Each translation produces a unique type in the focusing language for each type in the standard formulation as stated in the following lemma:

Lemma 5.1. *(type translation) For all types, τ,*

1. $\exists! A^-$ *such that* $\tau \overset{-}{\rightsquigarrow} A^-$
2. $\exists! A^+$ *such that* $\tau \overset{+}{\rightsquigarrow} A^+$

Proof. The induction argument for this lemma allows the negative case to appeal to the positive case with the same type, τ, as long as the type is positive, and similarly, the positive case may appeal to the negative case with the same type as long as it is negative. A similar argument will be used in proving properties of the translation. □

The term translation is defined by the two judgements:

$$\gamma|\Gamma \vdash (t[\theta] : \tau) \overset{-}{\rightsquigarrow} v^- : A^- \text{ and } \gamma|\Gamma \vdash (t[\theta] : \tau) \overset{+}{\rightsquigarrow} e : A^+$$

Both judgements contain a positive substitution, θ, that associates each variable, $x : \tau$, in the context γ with a constructor pattern, p, such that $\Delta \Vdash p_i :: A^+$ where $\tau \overset{+}{\rightsquigarrow} A^+$. This is necessary as there is not a direct correspondence between the variables in γ and the variables in Γ. Instead, each variable in γ is associated with a constructor pattern, and the free variables, Δ, of that constructor pattern are in the context Γ. One use of the positive substitution is shown in the rule for the

Figure 10: Polarity of the types in the standard formulation

$\boxed{\tau \text{ pos}}$

$$\frac{}{\tau_1 + \tau_2 \text{ pos}} \qquad \frac{}{0 \text{ pos}}$$

$\boxed{\tau \text{ neg}}$

$$\frac{}{\tau_1 \to \tau_2 \text{ neg}} \qquad \frac{}{\tau_1 \times \tau_2 \text{ neg}} \qquad \frac{}{\top \text{ neg}}$$

Figure 11: Type translation

$\boxed{\tau \overset{+}{\rightsquigarrow} A^+}$

$$\frac{\tau \text{ neg} \quad \tau \overset{-}{\rightsquigarrow} A^-}{\tau \overset{+}{\rightsquigarrow} \downarrow A^-} \qquad \frac{\tau_1 \overset{+}{\rightsquigarrow} A_1^+ \quad \tau_2 \overset{+}{\rightsquigarrow} A_2^+}{\tau_1 + \tau_2 \overset{+}{\rightsquigarrow} A_1^+ \oplus A_2^+} \qquad \frac{}{0 \overset{+}{\rightsquigarrow} 0}$$

$\boxed{\tau \overset{-}{\rightsquigarrow} A^-}$

$$\frac{\tau \text{ pos} \quad \tau \overset{+}{\rightsquigarrow} A^+}{\tau \overset{-}{\rightsquigarrow} \uparrow A^+} \qquad \frac{\tau_1 \overset{+}{\rightsquigarrow} A_1^+ \quad \tau_2 \overset{-}{\rightsquigarrow} A_2^-}{\tau_1 \to \tau_2 \overset{-}{\rightsquigarrow} A_1^+ \to A_2^-}$$

$$\frac{\tau_1 \overset{-}{\rightsquigarrow} A_1^- \quad \tau_2 \overset{-}{\rightsquigarrow} A_2^-}{\tau_1 \times \tau_2 \overset{-}{\rightsquigarrow} A_1^- \,\&\, A_2^-} \qquad \frac{}{1 \overset{-}{\rightsquigarrow} \top}$$

Figure 12: Representative cases of the translation to the focusing language

$\boxed{\gamma|\Gamma \vdash (t[\theta] : \tau) \mathrel{\overset{-}{\rightsquigarrow}} v^- : A^-}$

$$\frac{\tau \text{ pos} \quad \gamma|\Gamma \vdash (t[\theta] : \tau) \overset{+}{\rightsquigarrow} e : A^+}{\gamma|\Gamma \vdash (t[\theta] : \tau) \mathrel{\overset{-}{\rightsquigarrow}} \mathtt{val}^-(\{\epsilon \mapsto e\}) : \uparrow A^+}$$

$$\frac{\tau \text{ neg} \quad \tau \mathrel{\overset{-}{\rightsquigarrow}} A^- \quad \mathtt{id}\, y : A^- = v^-}{\gamma_1, x : \tau, \gamma_2|\Gamma \vdash (x[\theta_1, y/x, \theta_2] : \tau) \mathrel{\overset{-}{\rightsquigarrow}} v^- : A^-}$$

$$\frac{\gamma|\Gamma \vdash (t_i[\theta] : \tau_i) \mathrel{\overset{-}{\rightsquigarrow}} \mathtt{val}^-(\phi_i^-) : A_i^- \ (i = 1, 2) \quad \langle \phi_1^-, \phi_2^- \rangle_{A_1^- \,\&\, A_2^-} = \phi^-}{\gamma|\Gamma \vdash (\langle t_1, t_2 \rangle[\theta] : \tau_1 \times \tau_2) \mathrel{\overset{-}{\rightsquigarrow}} \mathtt{val}^-(\phi^-) : A_1^- \,\&\, A_2^-}$$

$\boxed{\gamma|\Gamma \vdash (t[\theta] : \tau) \overset{+}{\rightsquigarrow} e : A^+}$

$$\frac{\tau \text{ neg} \quad \gamma|\Gamma \vdash (t[\theta] : \tau) \mathrel{\overset{-}{\rightsquigarrow}} v^- : A^-}{\gamma|\Gamma \vdash (t[\theta] : \tau) \overset{+}{\rightsquigarrow} x[v^-/x] : \downarrow A^-}$$

$$\frac{\begin{array}{c}\tau \text{ pos} \quad \gamma|\Gamma \vdash (t[\theta] : \tau_1 + \tau_2) \overset{+}{\rightsquigarrow} e : A_1^+ \oplus A_2^+ \\ \forall (\Delta_i \Vdash p_i :: A_i^+) : \ \gamma, x_i : \tau_i|\Gamma, \Delta_i \vdash (t_i[\theta, p_i/x_i] : \tau) \overset{+}{\rightsquigarrow} e_{p_i} : A^+ \ (i = 1, 2) \\ [\{p_1 \mapsto e_{p_1}\}, \{p_2 \mapsto e_{p_2}\}]_{A_1^+ \oplus A_2^+} = \phi^+\end{array}}{\gamma|\Gamma \vdash (\mathtt{case}(t, x_1.t_1, x_2.t_2)[\theta] : \tau) \overset{+}{\rightsquigarrow} \mathtt{case}\, e : A_1^+ \oplus A_2^+ \ \mathtt{of}\ \mathtt{cont}^+(\phi^+) : A^+}$$

negative translation of a variable in Figure 12. In this case the variable $x : \tau$ must be associated with a pattern, y, as when τ is negative, $\tau \overset{+}{\rightsquigarrow} \downarrow A^-$, and the variable pattern is the only one of type $\downarrow A^-$. Therefore, we may form a negative value v^- of type A^- from y and give this as the result of the translation.

New patterns are added to the positive substitution when the translation goes under a binder. This is illustrated in the rule for the positive translation of a case statement. For each constructor pattern of type A_1^+, the left branch of the case is translated with the positive substitution extended to associate that pattern with x_1 and Γ extended with the free variables of the pattern. The right branch of the case is translated in the same way. The results of these translations along with the translation of the scrutinized term are combined with a substitution to form the translation of the case.

The rule for the negative translation of a term of positive type performs the positive translation of the term and then forms the trivial negative value of type $\uparrow A^+$ from the resulting expression. Similarly, the positive translation of a term of negative type forms the trivial expression of type $\downarrow A^-$ from the result of the negative translation. The case for the negative translation of a pair illustrates the general pattern found in many of the remaining cases. The two subterms are translated with the resulting negative values combined to form the translation of the pair. In this way, the negative translation produces a unique negative value for each term in the standard formulation. Similarly, the positive translation produces a unique expression for each term in the standard formulation. Moreover, the type of the negative value produced by the translation is the result of the negative translation of the type of the term in the standard formulation, and the type of the expression produced is the result of the positive translation of the type of the term. This is stated in the following lemma:

Lemma 5.2. *(translation) If* $\cdot \vdash t : \tau$*,* $\tau \overset{-}{\rightsquigarrow} A^-$*, and* $\tau \overset{+}{\rightsquigarrow} A^+$ *then*

1. $\exists! v^-$ *such that* $\cdot|\cdot \vdash (t[\cdot] : \tau) \overset{-}{\rightsquigarrow} v^- : A^-$
2. $\exists! e$ *such that* $\cdot|\cdot \vdash (t[\cdot] : \tau) \overset{+}{\rightsquigarrow} e : A^+$

Proof. As with the type translation proof, we allow inductive calls between the two cases on the same term as long as the type has the right polarity. Otherwise, the argument is a straightforward induction on the typing derivation in the standard formulation. See the appendix for details. □

The translation rules eliminate all of the β-redices through the use of focusing. Therefore, comparing two terms for equality is simplified by translating both

Figure 13: Judgements that generate all patterns of a given type

$$\frac{\boxed{\mathcal{L} \Vdash \mathcal{P} :: A^+} \quad \overline{[x : A^-] \Vdash [x] ::\downarrow A^-} \qquad \overline{[] \Vdash [] :: 0}}{\mathcal{L}_1 \Vdash \mathcal{P}_1 :: A_1^+ \quad \mathcal{L}_2 \Vdash \mathcal{P}_2 :: A_2^+}$$
$$\mathcal{L}_1 \mathbin{@} \mathcal{L}_2 \Vdash \texttt{inl}\ \mathcal{P}_1 \mathbin{@} \texttt{inr}\ \mathcal{P}_2 :: A_1^+ \oplus A_2^+$$

$$\frac{\boxed{\mathcal{L} \Vdash \mathcal{D} :: A^- > \mathcal{B}^+} \quad \overline{[\cdot] \Vdash [\epsilon] ::\uparrow A^+ > [A^+]} \qquad \overline{[] \Vdash [] :: \top > []}}{\mathcal{L}_1 \Vdash \mathcal{D}_1 :: A_1^- > \mathcal{B}_1^+ \quad \mathcal{L}_2 \Vdash \mathcal{D}_2 :: A_2^- > \mathcal{B}_2^+}$$
$$\mathcal{L}_1 \mathbin{@} \mathcal{L}_2 \Vdash \texttt{fst}; \mathcal{D}_1 \mathbin{@} \texttt{snd}; \mathcal{D}_2 :: A_1^- \mathbin{\&} A_2^- > \mathcal{B}_1^+ \mathbin{@} \mathcal{B}_1^+$$
$$\frac{\mathcal{L}_1 \Vdash \mathcal{P} :: A_1^+ \quad \mathcal{L}_2 \Vdash \mathcal{D} :: A_2^- > \mathcal{B}^+}{(\prod_{\Delta_1 \in \mathcal{L}_1} \Delta_1, \mathcal{L}_2) \Vdash (\prod_{p_1 \in \mathcal{P}} p_1; \mathcal{D}) :: A_1^+ \to A_2^- > (\prod_{\Delta_1 \in \mathcal{L}_1} \mathcal{B}^+)}$$

to the focusing language. However, the translations of equal terms are not necessarily identical focusing terms so we still must compare the focusing terms for equality.

6 Decidability of equality

As already noted, the extensional equality that we defined on focusing expressions quantifies over all closed substitutions, and this is a potentially infinite set so it is not immediately clear that this equality is decidable. To give a decision procedure for this equality we define the alternative finite extensional equality that only quantifies over the canonical substitutions. We give a way to enumerate all canonical substitutions to show that the finite extensional equality is decidable. We then show that the finite extensional equality is equivalent to the original extensional equality.

First, we show how to enumerate all patterns of a given type in Figure 13. This enumeration is a direct consequence of the fact that all of our types are finitely inhabited. We use calligraphy font letters to indicate a list. For example, $\mathcal{D}$ indicates a list of destructor patterns, and $\mathcal{L}$ indicates a list of linear contexts. We use the notation $\texttt{inl}\ \mathcal{P}$ to indicate the mapping of the function $p \mapsto \texttt{inl}\ p$ to all of the members of the list $\mathcal{P}$. We also use $\prod$ to denote the appending of several lists.

The enumeration of all terms of a given type is shown in Figure 14. To enumerate all expressions of a given type in a non-empty context, all observations of the variables in the context are performed, where an observation consists of a

Figure 14: Judgements that generate all terms of a given type

$\boxed{\mathcal{V}^- : A^-}$

$$\frac{\forall(\Delta \Vdash d :: A^- > B^+) :\ f^-(d) := \mathcal{E}_d \text{ where } \Delta \vdash \mathcal{E}_d :: B^+ \quad \texttt{all\_funs}^-_{A^-} f^- = \mathcal{F}^-}{\texttt{val}^-(\mathcal{F}^-) : A^-}$$

$\boxed{\mathcal{V}^+ :: A^+}$

$$\frac{\mathcal{L} \Vdash \mathcal{P} :: A^+ \quad \forall \Delta \in \mathcal{L} \quad \mathcal{S}_\Delta : \Delta \quad (\prod_{(\Delta,p)\in(\mathcal{L},\mathcal{P})} p[\mathcal{S}_\Delta]) = \mathcal{V}^+}{\mathcal{V}^+ :: A^+}$$

$\boxed{\Delta \vdash \mathcal{E} : A^+}$

$$\frac{\begin{array}{c}\mathcal{L}_i \Vdash \mathcal{D}_i :: A_i^- > \mathcal{B}_i^+ \\ \forall(\Delta_i \in \mathcal{L}_i) : \quad \mathcal{S}_{\Delta_i} : \Delta_i \quad (\prod_{(\Delta,d)\in(\mathcal{L}_i,\mathcal{D}_i)} d[\mathcal{S}_\Delta]) = \mathcal{D}'_i[\mathcal{S}_i] \quad (i \in 1..n) \\ \texttt{make\_obsvs}_{A^+}(x_1 : A_1^-, \ldots, x_n : A_n^-, [\mathcal{D}'_1[\mathcal{S}_1], \ldots, \mathcal{D}'_n[\mathcal{S}_n]], \cdot) = \mathcal{E}\end{array}}{x_1 : A_1^-, \ldots, x_n : A_n^- \vdash \mathcal{E} : A^+}$$

$$\frac{\mathcal{V}^+ :: A^+}{\cdot \vdash \mathcal{V}^+ : A^+}$$

$\boxed{\mathcal{S} : \Delta}$

$$\frac{}{[\cdot] : \cdot}$$

$$\frac{\mathcal{S} : \Delta \quad \mathcal{V}^- : A^- \quad (\prod_{\sigma\in\mathcal{S}}[\sigma, v^-/x \mid v^- \in \mathcal{V}^-]) = \mathcal{S}'}{\mathcal{S}' : \Delta, x : A^-}$$

Figure 15: Key rule of finite extensional equality

$$\frac{\mathcal{S} : \Gamma \quad \forall(\sigma \in \mathcal{S}) : \quad \cdot \vdash [\sigma]e_1 \equiv_{\mathrm{f}} [\sigma]e_2 :: A^+}{\Gamma \vdash e_1 \equiv_{\mathrm{f}} e_2 : A^+}$$

destructor pattern and a canonical substitution for its free variables. The results of these observations are then further scrutinized so that a full decision tree on the context is created. The auxiliary judgements for making all observations are given in a lexicographic order are given in the appendix. We also need a way to give a lexicographic order to the metafunctions ϕ^- and ϕ^+. The definition of this ordering is also given in the appendix.

For this enumeration to be correct, it must enumerate all canonical terms. This means that for any well typed term there must be an extensionally equal term in the enumeration. This fact is expressed in the following lemma:

Lemma 6.1. *(canonical terms)*

1. *If* $\Delta \vdash e : A^+$ *and* $\Delta \vdash \mathcal{E} : A^+$ *then* $\exists e_0 \in \mathcal{E}$ *such that* $\Delta \vdash e \equiv e_0 : A^+$
2. *If* $\cdot \vdash \sigma : \Delta$ *and* $\mathcal{S} : \Delta$ *then* $\exists \sigma_0 \in \mathcal{S}$ *such that* $\cdot \vdash \sigma \equiv \sigma_0 : \Delta$
3. *Similarly for remaining enumerations*

Proof. The proof is by induction on the relevant types and contexts. In the first case, each expression, e, is associated with a canonical term that is equal as follows. The canonical term is generated that corresponds to this context, Δ, and at the leaf corresponding to the substitution, σ_i, is the positive value, $p[\sigma_0]$, where $[\sigma_i]_\Delta e = p[\sigma]$ and σ_0 is the canonical substitution corresponding to σ given by the induction hypothesis. It is then demonstrated that this results in a extensionally equal expression using functionality. □

Using this enumeration, we can now define the finite extensional equality. The only rule that is significantly modified is presented in Figure 15. Rather than quantify over all substitutions, we enumerate the canonical substitutions for these closed instances.

Using the canonical terms and the extensional equality-substitution we can show that the two definitions of equality are equivalent. That is, two expressions are extensionally equal if and only if they are finite extensionally equal as stated in the following lemma:

Lemma 6.2. *(finite extensional equality) If* $\Gamma \vdash e_1 : A^+$ *and* $\Gamma \vdash e_2 : A^+$ *then* $\Gamma \vdash e_1 \equiv e_2 : A^+$ *if and only if* $\Gamma \vdash e_1 \equiv_{\mathrm{f}} e_2 : A^+$

Proof. The fact that extensional equality implies finite extensional equality is immediate as the set of canonical substitutions is a subset of the set of all substitutions. The other direction results from the previous lemma combined with the functionality lemma. □

7 Completeness

To show completeness, we show that each of the equalities given in the definition of definitional equality results in extensionally equal terms in the focusing language when translated. In this section we present a few representative cases of these proofs. First we prove several properties of the translation to the focusing language and relate it to substitution in the standard formulation.

7.1 Properties of the translation

A key property of the translation is that it preserves substitution. That is, a substitution in the standard formulation such as $[t_1/x]t$ translates to a term that corresponds to a substitution in the focusing language of the the translations of t_1 and t. While this cannot be stated in exactly this manner due to the way that the translation is defined, the following lemma expresses this idea.

Lemma 7.1. *(translation-substitution)*

1. *If* $\gamma, x : \tau_1 \vdash t : \tau$, $\gamma \vdash t_1 : \tau_1$, τ_1 pos, $\gamma|\Gamma \vdash (t_1[\theta] : \tau_1) \stackrel{+}{\rightsquigarrow} e_1 : A_1^+$, $\gamma|\Gamma \vdash ([t_1/x]t[\theta] : \tau) \stackrel{+}{\rightsquigarrow} e' : A^+$, *and for all* $\Delta \Vdash p :: A_1^+$, $\gamma, x : \tau_1|\Gamma, \Delta \vdash (t[\theta, p/x] : \tau) \stackrel{+}{\rightsquigarrow} e_p : A^+$ *and* $\phi^+(p) := e_p$ *then* $\Gamma \vdash e' \equiv \mathtt{case}\ e_1 : A_1^+\ \mathtt{of}\ \mathtt{cont}^+(\phi^+) : A^+$

2. *If* $\gamma, x : \tau_1 \vdash t : \tau$, $\gamma \vdash t_1 : \tau_1$, τ_1 neg, $\gamma|\Gamma \vdash (t_1[\theta] : \tau_1) \stackrel{-}{\rightsquigarrow} v_1^- : A_1^-$, $\gamma|\Gamma \vdash ([t_1/x]t[\theta] : \tau) \stackrel{+}{\rightsquigarrow} e' : A^+$, *and* $\gamma, x : \tau_1|\Gamma, x : A_1^- \vdash (t[\theta] : \tau) \stackrel{+}{\rightsquigarrow} e : A^+$ *then* $\Gamma \vdash e' \equiv [v_1^-/x]e : A^+$

3. *Additional two cases for* τ_1 pos *with the negative translation of* t_1 *and for* τ_1 neg *with the positive translation of* t_1

Proof. The proof is by induction on the derivation of the translation. Several simple lemmas about substitution's interactions with judgements such as `inject` are required. □

7.2 Completeness of equalities

To show that the translations of definitionally equal terms are extensionally equal, we first cover the base cases. Namely, we look at those rules of definitional equality which do not have premises in terms of the equality. There is then an inductive argument for the remaining cases.

The β-equality for conjunction is a representative case of many of the equalities which result in identical terms in the focusing language.

Lemma 7.2. *(β-$\times$-1) If* $\gamma \vdash t_1 : \tau_1$, $\gamma \vdash t_2 : \tau_2$, $\gamma|\Gamma \vdash (t_1[\theta] : \tau_1) \mathrel{\bar{\rightsquigarrow}} v_1^- : A_1^-$, *and* $\gamma|\Gamma \vdash (\mathtt{fst}\ \langle t_1, t_2\rangle[\theta] : \tau_1) \mathrel{\bar{\rightsquigarrow}} v^- : A_1^-$ *then* $v^- = v_1^-$.

Proof. We split the proof into two cases.
Case 1: τ_1 neg

1. Let $v_1^- = \mathtt{val}^-(\phi_1^-)$
2. By the definition of the translation,
   $\gamma|\Gamma \vdash (\langle t_1, t_2\rangle[\theta] : \tau_1 \times \tau_2) \mathrel{\bar{\rightsquigarrow}} \mathtt{val}^-(\phi^-) : A_1^-$ where for all
   $\Delta \Vdash d :: A_1^- > B^+$, $\phi^-(\mathtt{fst}; d) := \phi_1^-(d)$
3. By the definition of the translation, $v^- = \mathtt{val}^-(\phi_1^-) = v_1^-$ as
   $\mathtt{fst}_{A_1^-}(\phi^-) = \phi_1^-$

The proof of the case for τ_1 pos is similar. □

The case for the β-equality of the function type is more complicated. As it involves substitution, we use the translation-substitution lemma.

Lemma 7.3. *(β-$\rightarrow$) If* $\gamma, x : \tau_1 \vdash t : \tau$, $\gamma \vdash t_1 : \tau_1$,
$\gamma|\Gamma \vdash ((\lambda x : \tau_1.t)\ t_1[\theta] : \tau) \mathrel{\overset{+}{\rightsquigarrow}} e : A^+$, *and* $\gamma|\Gamma \vdash ([t_1/x]t[\theta] : \tau) \mathrel{\overset{+}{\rightsquigarrow}} e' : A^+$ *then* $\Gamma \vdash e \equiv e' : A^+$

Proof. Again, we only consider the case when τ_1 neg and τ pos.

1. By lemma, there exist unique e_0 and $\mathtt{val}^-(\phi_1^-)$ such that
   $\gamma, x : \tau_1|\Gamma, x : A_1^- \vdash (t[\theta] : \tau) \mathrel{\overset{+}{\rightsquigarrow}} e_0 : A^+$ and
   $\gamma|\Gamma \vdash (t_1[\theta] : \tau_1) \mathrel{\bar{\rightsquigarrow}} \mathtt{val}^-(\phi_1^-) : A_1^-$

2. By the definition of the translation,
   $\gamma, x : \tau_1 | \Gamma, x : A_1^- \vdash (t[\theta] : \tau) \overset{-}{\rightsquigarrow} \mathtt{val}^-(\phi_0^-) : \uparrow A^+$ where $\phi_0^-(\epsilon) := e_0$

3. By the definition of the translation,
   $\gamma | \Gamma \vdash (\lambda x : \tau_1 . t[\theta] : \tau_1 \to \tau) \overset{-}{\rightsquigarrow} \mathtt{val}^-(\phi_2^-) : \downarrow A_1^- \to \uparrow A^+$ where
   $\phi_2^-(x; \epsilon) := e_0$

4. By the definition of the translation,
   $\gamma | \Gamma \vdash (t_1[\theta] : \tau_1) \overset{+}{\rightsquigarrow} y[\mathtt{val}^-(\phi_1^-)/y] : \downarrow A_1^-$

5. By the definition of the translation,
   $e = \mathtt{case}\ y[\mathtt{val}^-(\phi_1^-)/y] : \downarrow A_1^-\ \mathtt{of}\ \mathtt{cont}^+(\phi_2^+) = [\mathtt{val}^-(\phi_1^-)/x]_{x:A_1^-} e_0$
   where $\phi_2^+(x) := \phi_2^-(x; \epsilon)$

6. By the translation-substitution lemma, $\Gamma \vdash e \equiv e' : A^+$

□

The case for the η-equality of the sum type is similar to the case for the β-equality of the function type in that it also involves substitution. The permissiveness of the extensional equality greatly simplifies this case.

Lemma 7.4. *(η-+) If* $\gamma \vdash t_0 : \tau_1 + \tau_2$, $\gamma, x : \tau_1 + \tau_2 \vdash t : \tau$,
$\gamma | \Gamma \vdash ([t_0/x]t[\theta] : \tau) \overset{+}{\rightsquigarrow} e : A^+$, *and*
$\gamma | \Gamma \vdash (\mathtt{case}(t_0, x_1.([\mathtt{inl}^{\tau_1+\tau_2}\ x_1/x]t), x_2.([\mathtt{inr}^{\tau_1+\tau_2}\ x_2]t))[\theta] : \tau) \overset{+}{\rightsquigarrow} e' : A^+$
then $\Gamma \vdash e \equiv e' : A^+$

Proof. As in the previous lemma, the proof relies on the translation-substitution lemma.

1. By lemma, there exists a unique e_0 such that
   $\gamma | \Gamma \vdash (t_0[\theta] : \tau_1 + \tau_2) \overset{+}{\rightsquigarrow} e_0 : A_1^+ \oplus A_2^+$

2. By lemma, for all $\Delta \Vdash p_1 :: A_1^+$ there exists a unique e_{p_1} such that
   $\gamma, x_1 : \tau_1 | \Gamma, \Delta \vdash ([\mathtt{inl}^{\tau_1+\tau_2}\ x_1/x]t[\theta, p_1/x_1] : \tau_1 + \tau_2) \overset{+}{\rightsquigarrow} e_{p_1} : A^+$ and similarly for A_2^+

3. By the translation-substitution lemma,
   $\Gamma \vdash \mathtt{case}\ e_0 : A_1^+ \oplus A_2^+\ \mathtt{of}\ \mathtt{cont}^+(\phi^+) \equiv e : A^+$ where for all
   $\Delta \Vdash p :: A_1^+ \oplus A_2^+$, $\phi^+(p) := e_p$ where
   $\gamma, x : \tau_1 + \tau_2 | \Gamma, \Delta \vdash (t[\theta, p/x] : \tau) \overset{+}{\rightsquigarrow} e_p : A^+$

4. By the definition of the translation, for all $\Delta \Vdash p_1 :: A_1^+$,
   $\gamma, x_1 : \tau_1 | \Gamma, \Delta \vdash (\texttt{inl}\ x_1[\theta, p_1/x_1] : \tau_1 + \tau_2) \overset{+}{\rightsquigarrow} \texttt{inl}\ p_1[\texttt{id}_\Delta] : A_1^+ \oplus A_2^+$
   and similarly for A_2^+

5. By the translation-substitution lemma,
   $\Gamma, \Delta \vdash \texttt{case inl}\ p_1[\texttt{id}_{\Delta_1}] : A_1^+ \oplus A_2^+\ \texttt{of cont}^+(\phi_{p_1}^+) \equiv e_{p_1} : A^+$ where for all $\Delta \Vdash p :: A_1^+ \oplus A_2^+$, $\phi_{p_1}^+(p) := e_{p_1,p}$ where
   $\gamma, x_1 : \tau_1, x : \tau_1 + \tau_2 | \Gamma, \Delta \vdash (t[\theta, p_1/x_1, p/x] : \tau) \overset{+}{\rightsquigarrow} e_{p_1,p} : A^+$

6. Note that x_1 does not appear in t so $e_{p_1,p} = e_p$ and therefore $\phi_{p_1}^+ = \phi^+$

7. By the definition of substitution,
   $\texttt{case inl}\ p_1[\texttt{id}_{\Delta_1}] : A_1^+ \oplus A_2^+\ \texttt{of cont}^+(\phi^+) = [\texttt{id}_{\Delta_1}]_{\Delta_1}\phi^+(\texttt{inl}\ p_1) = e_{\texttt{inl}\ p_1}$ and similarly for $\texttt{inr}\ p_2$

8. By inversion on the translation, $\Gamma \vdash e \equiv e' : A^+$

□

The remaining cases follow a similar pattern. For rules such as the compatibility rule for applications, the result follows using induction and functionality as the two resulting terms differ only in that extensionally equal terms are substituted. The induction argument for this and the other cases uses these base cases to give the general completeness result:

Lemma 7.5. *(completeness) If* $\cdot \vdash t_1 = t_2 : \tau$, $\cdot|\cdot \vdash (t_1[\cdot] : \tau) \overset{+}{\rightsquigarrow} e_1 : A^+$, *and* $\cdot|\cdot \vdash (t_2[\cdot] : \tau) \overset{+}{\rightsquigarrow} e_2 : A^+$ *then* $\cdot \vdash e_1 \equiv e_2 : A^+$

8 Soundness

It is not enough to simply equate all definitionally equal terms, we must also show that we only equate those terms that are actually definitionally equal. Otherwise, our algorithm could just say that all terms are equal. In order to show this we define a backward translation from the focusing language to the standard formulation and prove several theorems about this translation and the extensional equality.

8.1 Backward translation

We define a mapping from the focusing language back to the standard formulation in Figures **??** and 17. Note that we use ρ to denote a simultaneous substitution in the standard formulation. In addition to relying on a backward mapping for patterns, the backward translation of terms depends on the componentwise backward translation of the image of a metafunction and auxiliary judgements such as `grow` defined in the appendix. These auxiliary judgements convert the meta functions of the focusing language back into the more familiar syntax of the standard formulation.

The translation assigns a unique expression to each well-typed term in the standard formulation as stated in the following lemma holds:

Lemma 8.1. *(backward translation)*

1. *If* $\Gamma \vdash e : A^+$ *then* $\exists! t$ *such that* $\Gamma|\gamma \vdash e : A^+ \leftarrow t : \tau$ *and* $\gamma \vdash t : \tau$
2. *If* $\Gamma \vdash v^- : A^-$ *then* $\exists! t$ *such that* $\Gamma|\gamma \vdash v^- : A^- \leftarrow t : \tau$ *and* $\gamma \vdash t : \tau$

Proof. By lexicographic induction on the typing derivation and then on an ordering of the cases so that the case for the positive translation can appeal to the case for the negative translation if the type is negative and the case for the negative translation can appeal to the case for the positive translation if the type is positive. □

8.2 Properties of the back translation

There are several key properties of the translation that are essential to showing soundness. First we consider its interaction with substitution. Namely, we prove that the back translation commutes with substitution. That is, the following lemma holds:

Lemma 8.2. *(back translation-substitution)*

1. *If* $\Gamma, \Delta \vdash e : A^+$, $\Gamma \vdash \sigma : \Delta$, $\Gamma, \Delta|\gamma, \gamma' \vdash e : A^+ \leftarrow t : \tau$, $\Gamma|\gamma \vdash \sigma : \Delta \leftarrow \rho : \gamma'$, *and* $\Gamma|\gamma \vdash [\sigma]_\Delta e : A^+ \leftarrow t' : \tau$ *then* $\gamma \vdash [\rho]_{\gamma'} t = t' : \tau$
2. *If* $\Gamma, \Delta \vdash v^- : A^-$, $\Gamma \vdash \sigma : \Delta$, $\Gamma, \Delta|\gamma, \gamma' \vdash v^- : A^- \leftarrow t : \tau$, $\Gamma|\gamma \vdash \sigma : \Delta \leftarrow \rho : \gamma'$, *and* $\Gamma|\gamma \vdash [\sigma]_\Delta v^- : A^- \leftarrow t' : \tau$ *then* $\gamma \vdash [\rho]_{\gamma'} t = t' : \tau$

Figure 16: Backward translation of types and contexts

$\boxed{A^- \overset{-}{\leftarrow} \tau}$

$$\frac{}{\top \overset{-}{\leftarrow} 1} \qquad \frac{A_1^- \overset{-}{\leftarrow} \tau_1 \quad A_2^- \overset{-}{\leftarrow} \tau_2}{A_1^- \,\&\, A_2^- \overset{-}{\leftarrow} \tau_1 \times \tau_2}$$

$$\frac{A_1^+ \overset{+}{\leftarrow} \tau_1 \quad A_2^- \overset{-}{\leftarrow} \tau_2}{A_1^+ \to A_2^- \overset{-}{\leftarrow} \tau_1 \to \tau_2} \qquad \frac{A^+ \overset{+}{\leftarrow} \tau}{\uparrow A^+ \overset{-}{\leftarrow} \tau}$$

$\boxed{A^+ \overset{+}{\leftarrow} \tau}$

$$\frac{}{0 \overset{-}{\leftarrow} 0} \qquad \frac{A_1^+ \overset{+}{\leftarrow} \tau_1 \quad A_2^+ \overset{+}{\leftarrow} \tau_2}{A_1^+ \oplus A_2^+ \overset{+}{\leftarrow} \tau_1 + \tau_2} \qquad \frac{A^- \overset{-}{\leftarrow} \tau}{\downarrow A^- \overset{+}{\leftarrow} \tau}$$

$\boxed{\Delta | \gamma \Vdash p :: A^+ \leftarrow t : \tau}$

$$\frac{A^- \overset{-}{\leftarrow} \tau}{x : A^- | x : \tau \Vdash x ::\downarrow A^- \leftarrow x : \tau}$$

$$\frac{\Delta | \gamma \Vdash p :: A_1^+ \leftarrow t : \tau_1 \quad A_2^+ \overset{+}{\leftarrow} \tau_2}{\Delta | \gamma \Vdash \mathtt{inl}\ p :: A_1^+ \oplus A_2^+ \leftarrow \mathtt{inl}^{\tau_1+\tau_2} t : \tau_1 + \tau_2}$$

$$\frac{\Delta | \gamma \Vdash p :: A_2^+ \leftarrow t : \tau_2 \quad A_1^+ \overset{+}{\leftarrow} \tau_1}{\Delta | \gamma \Vdash \mathtt{inr}\ p :: A_1^+ \oplus A_2^+ \leftarrow \mathtt{inr}^{\tau_1+\tau_2} t : \tau_1 + \tau_2}$$

Figure 17: Backward translation of terms

$\boxed{\Gamma|\gamma \vdash e : A^+ \leftarrow t : \tau}$

$$\frac{\Gamma|\gamma \vdash v^+ :: A^+ \leftarrow t : \tau}{\Gamma|\gamma \vdash v^+ : A^+ \leftarrow t : \tau}$$

$$\frac{x : A^-|x : \tau \in \Gamma|\gamma \quad \mathtt{obsv}(d, A^-, x) = (t_0, \Delta_0, A_0^+, \gamma_0) \quad \Gamma|\gamma \vdash \sigma : \Delta_0 \leftarrow \rho : \gamma_0 \quad \Gamma|\gamma \vdash \phi^+ : A_0^+ > A^+ \leftarrow \psi^+ : A_0^+ > \tau \quad \mathtt{grow}(\psi^+, A_0^+, A^+, [\rho]t_0) = t}{\Gamma|\gamma \vdash x \bullet d[\sigma]; \mathtt{cont}^+(\phi^+) : A^+ \leftarrow t : \tau}$$

$\boxed{\Gamma|\gamma \vdash v^+ :: A^+ \leftarrow t : \tau}$

$$\frac{\Delta_0|\gamma_0 \Vdash p :: A^+ \leftarrow t_0 : \tau \quad \Gamma|\gamma \vdash \sigma : \Delta_0 \leftarrow \rho : \gamma_0 \quad [\rho]t_0 = t}{\Gamma|\gamma \vdash p[\sigma] :: A^+ \leftarrow t : \tau}$$

$\boxed{\Gamma|\gamma \vdash \sigma : \Delta_0 \leftarrow \rho : \gamma_0}$

$$\frac{}{\Gamma|\gamma \vdash \cdot : \cdot \leftarrow \cdot : \cdot}$$

$$\frac{\Gamma|\gamma \vdash \sigma_0 : \Delta_0 \leftarrow \rho_0 : \gamma_0 \quad \Gamma|\gamma \vdash v^- : A^- \leftarrow t : \tau}{\Gamma|\gamma \vdash \sigma_0, v^-/x : \Delta_0, x : A^- \leftarrow \rho_0, t/x : \gamma_0, x : \tau}$$

3. *If* $\Gamma \vdash e : A^+$, $\Gamma|\gamma \vdash e : A^+ \leftarrow t : \tau$, $\gamma, x : \tau \vdash t' : \tau_0$,
$\Gamma|\gamma \vdash \texttt{case}\ e : A^+\ \texttt{of}\ \texttt{cont}^+(\phi^+) : A_0^+ \leftarrow t_{\mathrm{s}} : \tau_0$ *where* $\phi^+(p) := e_p$, *and for all* $\Delta \Vdash p :: A^+$, $\Gamma, \Delta \vdash e_p : A_0^+$, $\Gamma, \Delta|\gamma, \gamma' \vdash e_p : A_0^+ \leftarrow t_p : \tau_0$,
$\Delta|\gamma' \Vdash p :: A^+ \leftarrow t_{0,p} : \tau$, *and* $\gamma, \gamma' \vdash t_p = [t_{0,p}/x]_\tau t' : \tau_0$ *then*
$\gamma \vdash [t/x]_\tau t_0 = t_{\mathrm{s}} : \tau_0$

Proof. The proof is by induction on the typing judgement and relies on corresponding lemmas about `grow` and the other auxiliary judgements. □

We will also use the fact that every term in the standard formulation is equivalent to a decision tree whose branching depends only on the context, γ, where the leaves are of the form $[\rho]_\gamma t$. We will denote this term t_{tree}. Moreover, there is such a t_{tree} where all the substitutions, ρ, are in the image of the back translation. That is, there exists σ such that $\cdot \vdash \sigma : \Delta$ and $\cdot|\cdot \vdash \sigma : \Delta \leftarrow \rho : \gamma$. In this way we get a close correspondence between the extensional equality in the focusing language and the terms in the standard formulation as we have a way to express both in terms of closed instances.

It is also the case that the round trip of the translation to the focusing language and the back translation to the standard formulation results in a definitionally equal term. That is, the following lemma holds:

Lemma 8.3. *(round trip) If* $\cdot \vdash t : \tau$ *then*

1. *if* $\cdot|\cdot \vdash (t[\cdot] : \tau) \overset{-}{\rightsquigarrow} v^- : A^-$ *and* $\cdot|\cdot \vdash v^- : A^- \leftarrow t' : \tau$ *then* $\cdot \vdash t = t' : \tau$
2. *if* $\cdot|\cdot \vdash (t[\cdot] : \tau) \overset{+}{\rightsquigarrow} e : B^+$ *and* $\cdot|\cdot \vdash e : B^+ \leftarrow t' : \tau$ *then* $\cdot \vdash t = t' : \tau$

Proof. The lemma is proved by proving a stronger lemma about non-empty contexts that uses simultaneous substitutions in the standard formulation. The proof is by induction on the forward translation and uses the back translation-substitution lemma. □

8.3 Soundness Proof

To show soundness we will first prove the following key lemma:

Lemma 8.4. *(soundness of definitional equality) If* $\Gamma \vdash e_1 \equiv e_2 : A^+$,
$\Gamma|\gamma \vdash e_1 : A^+ \leftarrow t_1 : \tau$. *and* $\Gamma|\gamma \vdash e_2 : A^+ \leftarrow t_2 : \tau$ *then* $\gamma \vdash t_1 = t_2 : \tau$.

Proof. The proof is by induction on the definition of extensional equality. The interesting case is for expressions.

1. Assume $\Gamma \vdash e_1 \equiv e_2 : A^+$, $\Gamma|\gamma \vdash e_1 : A^+ \leftarrow t_1 : \tau$. and $\Gamma|\gamma \vdash e_2 : A^+ \leftarrow t_2 : \tau$

2. By the tree lemma, we have $\gamma \vdash t_1 = t_{1,\mathrm{tree}} : \tau$ and $\gamma \vdash t_2 = t_{2,\mathrm{tree}} : \tau$

3. As the branching structure of $t_{1,\mathrm{tree}}$ and $t_{2,\mathrm{tree}}$ depends only on the context, γ, it is enough to compare these terms at the leaves. At each leaf we have $[\rho]_\gamma t_1$ and $[\rho]_\gamma t_2$, respectively, where there exists σ such that $\cdot|\cdot \vdash \sigma : \Gamma \leftarrow \rho : \gamma$

4. By the definition of extensional equality, $\cdot \vdash [\sigma]_\Gamma e_1 \equiv [\sigma]_\Gamma e_2 :: A^+$. So we can appeal to the induction hypothesis to get that the backward translations of $[\sigma]_\Gamma e_1$ and $[\sigma]_\Gamma e_2$ are definitionally equal.

5. By the back translation-substitution lemma and the transitivity of definitional equality, $[\rho]_\gamma t_1 = [\rho]_\gamma t_2$

6. Therefore, again by transitivity, we have $\gamma \vdash t_1 = t_2 : \tau$

□

We can now prove soundness as a corollary.

Lemma 8.5. *(soundness) If* $\cdot|\cdot \vdash (t_1[\cdot] : \tau) \overset{+}{\rightsquigarrow} e_1 : A^+$, $\cdot|\cdot \vdash (t_2[\cdot] : \tau) \overset{+}{\rightsquigarrow} e_2 : A^+$, *and* $\cdot \vdash e_1 \equiv e_2 : A^+$ *then* $\cdot \vdash t_1 = t_2 : \tau$

Proof. The result follows directly from the previous lemmas.

1. By the round trip lemma, t_1 and t_2 are equal to the back translations of e_1 and e_2, respectively

2. By the soundness of definitional equality lemma, the back translations are definitionally equal

3. By transitivity of definitional equality, $\cdot \vdash t_1 = t_2 : \tau$

□

References

T. Altenkirch and T. Uustalu. Normalization by evaluation for lambda-2. In Y. Kameyama and P. J. Stuckey, editors, *FLOPS*, volume 2998 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2004. ISBN 3-540-21402-X.

T. Altenkirch, P. Dybjer, M. Hofmann, and P. Scott. Normalization by evaluation for typed λ-calculus with coproducts. In *IEEE Symposium on Logic in Computer Science*, pages 203–210. IEEE Press, 2001.

J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2:297–347, 1992.

V. Balat, R. D. Cosmo, and M. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. *SIGPLAN Not.*, 39(1):64–76, 2004. ISSN 0362-1340.

N. Ghani. $\beta\eta$-equality for coproducts. In *International Conference on Typed Lambda Calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 171–185. Springer-Verlag, 1995.

D. R. Licata, N. Zeilberger, and R. Harper. Focusing on binding and computation. In *IEEE Symposium on Logic in Computer Science*, 2008.

S. Lindley. Extension rewriting with sums. In *International Conference on Typed Lambda Calculi and Applications*, volume 4583, pages 255–271. Springer Berlin/Heidelberg, 2007.

K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

N. Zeilberger. Focusing and higher-order abstract syntax. *SIGPLAN Not.*, 43(1):359–369, 2008. ISSN 0362-1340.