Blame, coercion, and threesomes: Together again for the first time

Draft, 19 October 2014

Jeremy Siek
Indiana University
jsiek@indiana.edu

Peter Thiemann
Universität Freiburg
thiemann@informatik.uni-freiburg.de

Philip Wadler
University of Edinburgh
wadler@inf.ed.ac.uk

Abstract

We present four calculi for gradual typing: $\lambda\mathsf{B}$, based on the blame calculus of Wadler and Findler (2009); $\lambda\mathsf{C}$, based on the coercion calculus of Henglein (1994); and $\lambda\mathsf{T}$ and $\lambda\mathsf{W}$, based on the threesome calculi with and without blame of Siek and Wadler (2010). We define translations from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$, from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$, and from $\lambda\mathsf{T}$ to $\lambda\mathsf{W}$. We show each of the translations is fully abstract—far stronger correctness results than have previously appeared.

1. Introduction

C#, Dart, Pyret, Racket, TypeScript, VB: many recent languages integrate dynamic and static types. Blame, coercions, and threesomes provide a foundation for such integration. The *blame calculus* models the interaction of dynamic and static types; it compiles into a *coercion calculus*, which models how to implement casts; it in turn compiles into the *threesome calculus*, which models a space-efficient implementation.

Henglein (1994) introduced a coercion calculus to study compilation of dynamically typed languages into statically typed ones. Findler and Felleisen (2002) introduced the notion of higher-order contracts and blame labels to pinpoint the location of contract failures. Siek and Taha (2006) introduced gradual typing as a way to integrate static and dynamic typing via implicit casts. Wadler and Findler (2009) introduced the blame calculus and the use of progress and preservation to prove blame safety. Herman et al. (2007, 2010) exploited the coercion calculus to provide a version of gradual typing that is space-efficient. Siek et al. (2009) proved that the coercion calculus simulates the blame calculus. Siek and Wadler (2010) introduced threesomes, a variant of the blame calculus that is space-efficient, and connected threesomes with the coercion calculus. Garcia (2013) observes that coercions are easier to understand while threesomes are easier to implement, and ameliorates this tension by showing how to derive threesomes from coercions. Greenberg (2013) relates casts to space-efficient coercions through a sequence of three calculi, CAST, NAIVE, and EFFICIENT.

This paper makes the following contributions.

- We present four calculi for gradual typing:

  The blame calculus, $\lambda\mathsf{B}$, based on Wadler and Findler (2009).

  A novel coercion calculus, $\lambda\mathsf{C}$, inspired by Henglein (1994), Herman et al. (2007, 2010), and Greenberg (2013). Whereas previous work uses a reduction rule that combines smaller coercions into larger ones, we use a rule that decomposes larger coercions into smaller ones.

  Two novel threesome calculi with and without blame, $\lambda\mathsf{T}$ and $\lambda\mathsf{W}$, inspired by Siek and Wadler (2010) and Garcia (2013). Whereas previous work introduces threesomes as triples of types and later relate them to coercions, we introduce threesomes as coercions and later relate them to triples of types. And whereas previous work begins with threesomes without blame and later add blame, we begin with threesomes with blame and later remove blame.

- We present translations from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$, from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$, and from $\lambda\mathsf{T}$ to $\lambda\mathsf{W}$. Each translation is shown to be a bisimulation and fully abstract.

  The bisimulations from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ and from $\lambda\mathsf{T}$ to $\lambda\mathsf{W}$ are lockstep, in that a single step in one calculus corresponds to a single step in the other. The bisimulation from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$ is not lockstep, in that a single step in one calculus corresponds to zero or more in the other. The lockstep bisimulations are novel. The bisimulation from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$ is similar to one relating blame and threesomes in Siek and Wadler (2010), though still the hardest proof in the paper. (The proof is presented in full in the supplementary material.)

  The proofs of full abstraction are novel. Previous work establishes weaker results, such as that a closed term converges iff its translation converges (that is, a contextual equivalence result restricted to the empty context).

- We introduce a novel translation from $\lambda\mathsf{C}$ to $\lambda\mathsf{B}$, needed to show that the translation from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ is fully abstract. Since a coercion may contain many blame labels but a cast has a single blame label, our translation breaks a single coercion into many casts. Justification of the translation requires a proof of several contextual equivalences in $\lambda\mathsf{C}$.

- We present formulations of blame safety for $\lambda\mathsf{B}$, $\lambda\mathsf{C}$, and $\lambda\mathsf{T}$, and a proof that the translations from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ and from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$ preserve and reflect blame safety. As a pleasant consequence, the somewhat subtle definition of blame safety and subtyping for $\lambda\mathsf{B}$ is justified by the straightforward definition of blame safety for $\lambda\mathsf{C}$. (The proof is presented in full in the supplementary material.)

  Our formulation of blame safety for $\lambda\mathsf{B}$ is standard, but the formulation for $\lambda\mathsf{C}$ and $\lambda\mathsf{T}$ is novel. Remarkably, preservation of blame safety has not previously been considered.

- We introduce a novel proof technique. Often, distinct terms in $\lambda\mathsf{B}$ or $\lambda\mathsf{C}$ translate into the same term of $\lambda\mathsf{T}$; this provides an easy way to establish equivalences via full abstraction. We apply the technique to validate the equations needed to justify the translation from $\lambda\mathsf{C}$ to $\lambda\mathsf{B}$, and to provide a simpler proof of the Fundamental Property of Casts than that given in Siek and Wadler (2010).

The paper is organised as follows. Section 2 provides an overview. Sections 3–6 describe the four calculi; each section presents type safety and blame safety for its calculus, and translations to and from calculi of the preceding sections. Section 7 describes and applies our novel proof technique. Section 8 surveys related work.

2. Overview

This section demonstrates key features of the three calculi through a series of examples.

2.1 Blame calculus, $\lambda\mathsf{B}$

The blame calculus supports integration of dynamically and statically typed code. Dynamically typed code has the type $\star$, where

$$\star \cong K + (\star \to \star)$$

meaning every value of dynamic type is either of base type K or function type $\star{\to}\star$. Base types include integers $\mathtt{I}$ and Booleans $\mathtt{B}$.

Let $f : \mathtt{I}{\to}\mathtt{I}$ be the statically-typed increment function, $(\lambda x{:}\mathtt{I}.\, x + 1)$. Here we use dynamically-typed code in a statically-typed context:

$$((\lceil \lambda g.\, g\ \mathtt{3} \rceil : \star \overset{p}{\Longrightarrow} (\mathtt{I}{\to}\mathtt{I}){\to}\mathtt{I})\ f) \longrightarrow^*_{\mathsf{B}} \mathtt{4}$$

We write $\longrightarrow_{\mathsf{B}}$ to indicate the first term reduces to the second. Here $\lceil\cdot\rceil$ embeds a dynamically-typed term in the blame calculus, and $A \overset{p}{\Longrightarrow} B$ is a *cast* from a source type A to a target type B topped by a blame label. We let p, q range over blame labels. Each blame label p has a complement $\overline{p}$. Complement is involutive, $\overline{\overline{p}} = p$. Our notation is chosen for clarity, not compactness: a practical language might infer the source or target type of a cast, or even an entire cast.

Blame labels are used to report the location of a cast failure. Here is our code with integer 3 replaced by Boolean `t`:

$$((\lceil \lambda g.\, g\ \mathtt{t} \rceil : \star \overset{p}{\Longrightarrow} (\mathtt{I}{\to}\mathtt{I}){\to}\mathtt{I})\ f) \longrightarrow^*_{\mathsf{B}} \mathtt{blame}\ p$$

It returns `blame` p, indicating that the cast labeled p failed, and, further, that blame is allocated to the *term contained* in the cast—we call this *positive* blame.

We may also use statically-typed code in a dynamically-typed context:

$$(\mathtt{let}\ g = (f : \mathtt{I}{\to}\mathtt{I} \overset{p}{\Longrightarrow} \star)\ \mathtt{in}\ \lceil g\ \mathtt{3} \rceil) \longrightarrow^*_{\mathsf{B}} \lceil \mathtt{4} \rceil$$

Here is our code with integer 3 replaced by Boolean `t`:

$$(\mathtt{let}\ g = (f : \mathtt{I}{\to}\mathtt{I} \overset{p}{\Longrightarrow} \star)\ \mathtt{in}\ \lceil g\ \mathtt{t} \rceil) \longrightarrow^*_{\mathsf{B}} \mathtt{blame}\ \overline{p}$$

It returns `blame` $\overline{p}$ (note the complement!), indicating that the cast labeled p failed, and, further, that blame is allocated to the *context containing* the cast—we call this *negative* blame.

Dynamically typed programs can themselves be described as terms in the blame calculus. For instance $\lceil \lambda g.\, g\ \mathtt{3} \rceil$ abbreviates

$$(\lambda g{:}\star\,.\,(g : \star \overset{p_1}{\Longrightarrow} \star \to \star)\ (3 : \mathtt{I} \overset{p_2}{\Longrightarrow} \star)) : \star{\to}\star \overset{p_3}{\Longrightarrow} \star$$

A fresh blame label p is introduced for each cast.

Blame safety lets us characterise under what circumstances a cast may produce positive or negative blame. In particular, we show that a cast from dynamic type only allocates positive blame; and that a cast to dynamic type only allocates negative blame. Hence, if a cast fails, it allocates blame to the dynamically typed side—"Well-typed programs can't be blamed".

Details of the blame calculus $\lambda\mathsf{B}$ appear in Section 3.

2.2 Coercion calculus, $\lambda\mathsf{C}$

The blame calculus compiles into a coercion calculus based on Henglein (1994).

For example, the cast

$$\star \overset{p}{\Longrightarrow} (\mathtt{I} \to \mathtt{I}) \to \mathtt{I}$$

compiles to the coercion

$$(\star{\to}\star)\mathtt{?}^p\ ;\ (((\mathtt{I}\mathtt{?}^p \to \mathtt{I}\mathtt{!})\ ;\ (\star{\to}\star)\mathtt{!}) \to \mathtt{I}\mathtt{?}^p)$$

while the cast

$$\mathtt{I} \to \mathtt{I} \overset{p}{\Longrightarrow} \star$$

compiles to the coercion

$$(\mathtt{I}\mathtt{?}^{\overline{p}} \to \mathtt{I}\mathtt{!})\ ;\ (\star{\to}\star)\mathtt{!}.$$

Here coercions `I!` and $(\star{\to}\star)\mathtt{!}$ are injections into the dynamic type; coercions $\mathtt{I}\mathtt{?}^p$ and $(\star{\to}\star)\mathtt{?}^p$ are projections from the dynamic type; arrow ($\to$) coerces functions; and semicolon (;) composes coercions—we discuss the details later. For now, observe that our first cast, which can be allocated positive blame but not negative, compiles to a coercion that contains p but not $\overline{p}$, while our second cast, which can be allocated negative blame but not positive, compiles to a coercion that contains $\overline{p}$ but not p.

Safety for $\lambda\mathsf{B}$ depends on a somewhat subtle definition of positive and negative subtyping, $<:^+$ and $<:^-$. In contrast, safety for $\lambda\mathsf{C}$ has a pleasingly simple definition: a coercion is safe for p if it does not contain label p. Consider a cast from A to B with label p; then $A <:^+ B$ if and only if the cast compiles to a coercion not containing p; and $A <:^- B$ if and only if the cast compiles to a coercion not containing $\overline{p}$. The forward direction of this implication is only to be expected, but the backward direction is a surprise, and a pleasant one: the somewhat subtle definition of $<:^+$ and $<:^-$ in $\lambda\mathsf{B}$ is justified by the correspondence to $\lambda\mathsf{C}$.

The key to space efficiency is to consolidate coercions. Hence, Herman et al. (2007, 2010) and Greenberg (2013) use the reduction rule

$$M\langle c\rangle\langle d\rangle \longrightarrow M\langle c\,;\,d\rangle \qquad \text{(COMPOSE)}$$

where $M\langle c\rangle$ applies to term M the coercion c, and $c\,;\,d$ composes coercion c followed by coercion d. Henglein (1994) only states (COMPOSE) as an equation and does not orient it as a reduction at all. In contrast, our calculus uses the reduction rule

$$M\langle c\,;\,d\rangle \longrightarrow M\langle c\rangle\langle d\rangle \qquad \text{(DECOMPOSE)}$$

Remarkably, our coercion calculus is the first to orient the reduction in this direction. The benefit is a tight connection between $\lambda\mathsf{B}$ and $\lambda\mathsf{C}$: the translation between the two is a lockstep bisimulation, and many proofs are simplified.

Details of the coercion calculus $\lambda\mathsf{C}$ appear in Section 4.

2.3 Threesome calculus, $\lambda\mathsf{T}$ and $\lambda\mathsf{W}$

A naive implementation of the blame or coercion calculus suffers space leaks. For instance, two mutually recursive procedures where the recursive calls are in tail position should run in constant space; but if one of them is statically typed and the other is dynamically typed, the mediating casts break the tail call property, and the program requires space proportional to the number of calls.

This problem was observed by Herman et al. (2007, 2010), who propose a solution based on the coercion calculus. If two coercions are applied in sequence then they are composed and normalised. Normalising a composition of two coercions yields a coercion whose height is the maximum of the heights of the two original coercions, ensuring that computation can proceed in bounded space. However, normalising coercions requires that sequences of compositions are treated as equal up to associativity. While this is not a difficult problem in symbol manipulation, it does pose a challenge in the context of implementing an efficient evaluator.

Siek and Wadler (2010) proposed a variant of this solution, observing that any sequence of casts

$$A \Longrightarrow C_1 \Longrightarrow \cdots \Longrightarrow C_n \Longrightarrow B$$

always compresses to an equivalent sequence of two casts

$$A \Longrightarrow C \Longrightarrow B$$

called a *threesome*—here we ignore blame labels, so these are threesomes without blame. The mediating type C is computed as a greatest lower bound of the types $A, C_1, \ldots, C_n, B$. Threesomes, like coercions, enable any sequence of casts to be represented in bounded space. Computing the greatest lower bound of two types yields a type whose height is the maximum of the heights of the two original types, again ensuring that computation can proceed in bounded space.

Blame is restored by decorating the mediating type with labels that indicate how blame is to be allocated. Siek and Wadler (2010) demonstrate that the decorated types are in one-to-one correspondence with normalised coercions. However, the notation for decorated types is far from transparent. The simplest way to validate that composition of threesomes is correctly defined is to convert the threesomes to coercions, normalise the composition of the coercions, and convert the result back to a threesome. In contrast, the correctness of our composition operator can be read off directly from the reduction rules introduced by Henglein (1994).

Siek and Wadler (2010) introduce threesomes as a triple of types, and later give the correspondence with coercions. They begin with threesomes without blame, and later add labeled types to represent blame. In contrast, here we introduce threesomes as coercions, and later give the correspondence with triples of types. We begin with threesomes with blame, and later simplify them to eliminate blame.

Details of the threesome calculi $\lambda\mathsf{T}$ and $\lambda\mathsf{W}$ appear in Sections 5 and 6.

3. Blame calculus

Figure 1 defines the blame calculus, $\lambda\mathsf{B}$. Similar results appear in Wadler and Findler (2009), Siek and Wadler (2010), and Ahmed et al. (2011).

Let A, B, C range over types. A type is either a base type K, a function type $A \to B$, or the dynamic type $\star$. Let G, H range over ground types. A ground type is either a base type K or the function type $\star \to \star$. The dynamic type satisfies the domain equation

$$\star \cong K + (\star \to \star)$$

so each value of dynamic type belongs to one ground type.

Types A and B are compatible if either is the dynamic type, if they are both the same base type, or they are both function types with compatible domains and ranges. Every type is either the dynamic type or compatible with a unique ground type. Two ground types are compatible if and only if they are equal.

Lemma 1 (Grounding)**.**

1. *If $A \neq \star$, there is a unique G such that $A \sim G$.*
2. *$G \sim H$ iff $G = H$.*

Incompatibility is the source of all blame: casting a type into the dynamic type and then casting out at an incompatible type allocates blame to the second cast.

Let p, q range over blame labels. To indicate on which side of a cast blame lays, each blame label p has a complement $\overline{p}$. Complement is involutive, $\overline{\overline{p}} = p$.

Let L, M, N range over terms. Terms are those of the simply-typed lambda calculus, plus blame and casts. Type judgments take the form $\Gamma \vdash_{\mathsf{B}} M : A$, where Γ is a type environment pairing variables with types. Each operator op on base types is specified by a total meaning function $[\![op]\!]$ that preserves types: if $op : \vec{K} \to K$ and $\vec{k} : \vec{K}$, then $[\![op]\!](\vec{k}) = k$ with $k : K$.

The typing rules for constants, operators, variables, abstraction, and application are standard (and not repeated in subsequent figures). Typing, reduction, and safety judgments are written with subscripts indicating to which calculus they belong, except we omit subscripts in figures to avoid clutter.

The typing rule for casts is straightforward:

$$\frac{\Gamma \vdash_{\mathsf{B}} M \qquad A \sim B}{\Gamma \vdash_{\mathsf{B}} (M : A \stackrel{p}{\Longrightarrow} B) : B}$$

If term M has type A and types A and B are compatible then the cast of M from type A to type B is a term of type B. The cast is decorated with a blame label p. If a cast from A to B decorated with p allocates blame to p we say it has *positive* blame, meaning the fault lies with the *term contained* in the cast; and if it allocates blame to $\overline{p}$ we say it has *negative* blame, meaning the fault lies with the *context containing* the cast.

We abbreviate a pair of casts

$$(M : A \stackrel{p}{\Longrightarrow} B) : B \stackrel{q}{\Longrightarrow} C \quad \text{as} \quad M : A \stackrel{p}{\Longrightarrow} B \stackrel{q}{\Longrightarrow} C.$$

Let V, W range over values. A value is a constant, a lambda term, a cast between values of function type, or a cast of a value from ground type to dynamic type. Let $\mathcal{E}$ range over evaluation contexts, which are standard, and include casts in the obvious way.

We write $M \longrightarrow_{\mathsf{B}} N$ to indicate that term M steps to term N. We write $\longrightarrow_{\mathsf{B}}^*$ for the reflexive and transitive closure of $\longrightarrow_B$.

Rules (BETA), (DELTA) are standard (and not repeated in subsequent figures). Rule (BASE): a cast from a base type to itself leaves the value unchanged. Rule (WRAP): a cast of a function applied to a value reduces to a term that casts on the domain, applies the function, and casts on the range; to allocate blame correctly, the blame label on the cast of the domain is complemented, corresponding to the fact that function types are contravariant in the domain and covariant in the range (Findler and Felleisen 2002; Wadler and Findler 2009). Rule (STAR): a cast from type $\star$ to itself leaves the value unchanged. Rule (INJECT): If A is not the dynamic type or a ground type, a cast from type A to type $\star$ factors into a cast from A to G followed by a cast from G to $\star$, where G is the unique ground type that is compatible with A. Rule (PROJECT): If A is not the dynamic type or a ground type, a cast from type $\star$ to type A factors into a cast from $\star$ to G followed by a cast from G to A, where G is the unique ground type that is compatible with A. Rule (COLLAPSE): A cast from a ground type G to type $\star$ and back to the same ground type G leaves the value unchanged. Rule (CONFLICT): a cast from a ground type G to type $\star$ and back to an incompatible ground type H allocates blame to the label of the outer cast. (Why the outer cast? This choice traces back to Findler and Felleisen (2002), and reflects the idea that we always hold an injection from ground type to dynamic type blameless, but may allocate blame to a projection from dynamic type to ground type.)

We write rule (INJECT) as follows.

$$\mathcal{E}[V : A \stackrel{p}{\Longrightarrow} \star] \longrightarrow \mathcal{E}[V : A \stackrel{p}{\Longrightarrow} G \stackrel{p}{\Longrightarrow} \star] \qquad \text{if } A \neq \star, A \neq G, A \sim G$$

A base type never satisfies the condition (there is no A such that $A \neq \star, A \neq K, A \sim K$), so in our case G must be the function type, and the rule is equivalent to the following.

$$\mathcal{E}[V : A \to B \stackrel{p}{\Longrightarrow} \star] \longrightarrow \mathcal{E}[V : A \to B \stackrel{p}{\Longrightarrow} \star \to \star \stackrel{p}{\Longrightarrow} \star] \qquad \text{if } A \to B \neq \star \to \star$$

The first rule is more compact, and adapts if we permit other ground types, such as product $\star \times \star$. Similarly for rule (PROJECT).

Syntax

Variables	x, y			Blame labels	p, q		
Constants	k	$::=$	$\mathtt{0} \mid \mathtt{1} \mid \cdots \mid \mathtt{t} \mid \mathtt{f} \mid \cdots$	Types	A, B, C	$::=$	$K \mid A \to B \mid \star$
Operators	op	$::=$	$+ \mid \leq \mid \cdots$	Ground types	G, H	$::=$	$K \mid \star \to \star$
Base types	K	$::=$	$\mathtt{I} \mid \mathtt{B}$	Type environment	Γ	$::=$	$\cdot \mid \Gamma,\, x : A$

Terms	L, M, N	$::=$	$k \mid op(\vec{M}) \mid x \mid \lambda x{:}A.\, N \mid L\ M \mid M : A \overset{p}{\Longrightarrow} B \mid \mathtt{blame}\ p$
Values	V, W	$::=$	$k \mid \lambda x{:}A.\, N \mid V : A \to B \overset{p}{\Longrightarrow} A' \to B' \mid V : G \overset{p}{\Longrightarrow} \star$
Evaluation context	$\mathcal{E}$	$::=$	$\Box \mid \mathcal{E}[op(\vec{V}, \Box, \vec{M})] \mid \mathcal{E}[\Box\ M] \mid \mathcal{E}[V\ \Box] \mid \mid \mathcal{E}[\Box : A \overset{p}{\Longrightarrow} B]$

Compatible $\boxed{A \sim B}$

$$\frac{}{K \sim K} \qquad \frac{A \sim A' \quad B \sim B'}{A \to B \sim A' \to B'} \qquad \frac{}{A \sim \star} \qquad \frac{}{\star \sim B}$$

Term typing $\boxed{\Gamma \vdash_{\mathsf{B}} M : A}$

$$\frac{k : K}{\Gamma \vdash k : K} \qquad \frac{\Gamma \vdash \vec{M} : \vec{K} \quad op : \vec{K} \to K}{\Gamma \vdash op(\vec{M}) : K} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x{:}A.\, N : A \to B} \qquad \frac{\Gamma \vdash L : A \to B \quad \Gamma \vdash M : A}{\Gamma \vdash L\ M : B}$$

$$\frac{\Gamma \vdash M : A \quad A \sim B}{\Gamma \vdash (M : A \overset{p}{\Longrightarrow} B) : B} \qquad \frac{}{\Gamma \vdash \mathtt{blame}\ p : A}$$

Reduction $\boxed{M \longrightarrow_{\mathsf{B}} N}$

$$\mathcal{E}[op(\vec{V})] \longrightarrow \mathcal{E}[[\![op]\!](\vec{V})] \qquad \text{(DELTA)}$$

$$\mathcal{E}[(\lambda x{:}A.\, N)\ V] \longrightarrow \mathcal{E}[N[x{:=}V]] \qquad \text{(BETA)}$$

$$\mathcal{E}[K : K \overset{p}{\Longrightarrow} K] \longrightarrow \mathcal{E}[V] \qquad \text{(BASE)}$$

$$\mathcal{E}[(V : A \to B \overset{p}{\Longrightarrow} A' \to B')\ W] \longrightarrow \mathcal{E}[(V\ (W : A' \overset{\overline{p}}{\Longrightarrow} A)) : B \overset{p}{\Longrightarrow} B'] \qquad \text{(WRAP)}$$

$$\mathcal{E}[V : \star \overset{p}{\Longrightarrow} \star] \longrightarrow \mathcal{E}[V] \qquad \text{(STAR)}$$

$$\mathcal{E}[V : A \overset{p}{\Longrightarrow} \star] \longrightarrow \mathcal{E}[V : A \overset{p}{\Longrightarrow} G \overset{p}{\Longrightarrow} \star] \quad \text{if } A \neq \star, A \neq G, A \sim G \qquad \text{(INJECT)}$$

$$\mathcal{E}[V : \star \overset{p}{\Longrightarrow} A] \longrightarrow \mathcal{E}[V : \star \overset{p}{\Longrightarrow} G \overset{p}{\Longrightarrow} A] \quad \text{if } A \neq \star, A \neq G, A \sim G \qquad \text{(PROJECT)}$$

$$\mathcal{E}[V : G \overset{p}{\Longrightarrow} \star \overset{q}{\Longrightarrow} G] \longrightarrow \mathcal{E}[V] \qquad \text{(COLLAPSE)}$$

$$\mathcal{E}[V : G \overset{p}{\Longrightarrow} \star \overset{q}{\Longrightarrow} H] \longrightarrow \mathtt{blame}\ q \quad \text{if } G \neq H \qquad \text{(CONFLICT)}$$

Figure 1. Blame calculus ($\lambda\mathsf{B}$)

Every well-typed term not containing blame has a unique type: if $\Gamma \vdash M : A$ and $\Gamma \vdash M : A'$ and M does not contain a term of the form $\mathtt{blame}\ p$, then $A = A'$.

Embedding $\lceil M \rceil$ takes terms of dynamically-typed lambda calculus into the blame calculus. The embedding introduces a fresh label p for each cast.

$$\begin{aligned}
\lceil k \rceil &= k : K \overset{p}{\Longrightarrow} \star && \text{if } k : K \\
\lceil op(\vec{M}) \rceil &= op(\lceil \vec{M} \rceil : \vec{\star} \overset{\vec{p}}{\Longrightarrow} \vec{K}) : K \overset{p}{\Longrightarrow} \star && \text{if } op : \vec{K} \to K \\
\lceil x \rceil &= x \\
\lceil \lambda x.\, M \rceil &= (\lambda x{:}\star\,.\, \lceil M \rceil) : \star \to \star \overset{p}{\Longrightarrow} \star \\
\lceil M\ N \rceil &= (\lceil M \rceil : \star \overset{p}{\Longrightarrow} \star \to \star)\ \lceil N \rceil
\end{aligned}$$

Type safety is established via preservation and progress.

Proposition 2 (Type safety for blame calculus)**.**

1. *If* $\vdash_{\mathsf{B}} M : A$ *and* $M \longrightarrow_{\mathsf{B}} N$ *then* $\vdash_{\mathsf{B}} N : A$.
2. *If* $\vdash_{\mathsf{B}} M : A$ *then either*
   (a) *there exists a value* V *such that* $M = V$, *or*
   (b) *there exists a label* p *such that* $M = \mathtt{blame}\ p$, *or*
   (c) *there exists a term* N *such that* $M \longrightarrow_{\mathsf{B}} N$.

The same result will apply, mutatis mutandis, for $\lambda\mathsf{C}$, $\lambda\mathsf{T}$, and $\lambda\mathsf{W}$.

Preservation and progress for the blame calculus do not rule out blame as a result. How to guarantee that blame cannot arise in certain circumstances is the subject of the next section.

3.1 Blame safety

Figure 2 presents four different subtyping relations and defines safety for the blame calculus.

Why do we need *four* different subtyping relations? Each has a different purpose. Relation $A <: B$ characterizes when a cast $A \Longrightarrow B$ *never* yields blame; relations $A <:^{+} B$ and $A <:^{-} B$ characterize when a cast $A \Longrightarrow B$ cannot yield *positive* or *negative* blame, respectively; and relation $A <:_n B$ characterizes when type A is more *precise* than type B. All four relations are reflexive and transitive, and subtyping, positive subtyping, and naive subtyping are antisymmetric.

The first three subtyping rules are characterised by *contravariance*. A cast from a base type to itself never yields blame (as per (BASE)). A cast from a function type to a function type never yields positive blame if the cast of the arguments never yields negative blame and if the cast of the results never yields positive blame; and ditto with positive and negative reversed; as with casts, each rule is

Subtype $\boxed{A <: B}$

$$\frac{}{K <: K} \qquad \frac{A' <: A \quad B <: B'}{A \to B <: A' \to B'} \qquad \frac{A <: G}{A <: \star}$$

Positive subtype $\boxed{A <:^+ B}$

$$\frac{}{K <:^+ K} \qquad \frac{A' <:^- A \quad B <:^+ B'}{A \to B <:^+ A' \to B'} \qquad \frac{}{A <:^+ \star}$$

Negative subtype $\boxed{A <:^- B}$

$$\frac{}{K <:^- K} \qquad \frac{A' <:^+ A \quad B <:^- B'}{A \to B <:^- A' \to B'} \qquad \frac{}{\star <:^- B} \qquad \frac{A <:^- G}{A <:^- \star}$$

Naive subtype $\boxed{A <:_n B}$

$$\frac{}{K <:_n K} \qquad \frac{A <:_n A' \quad B <:_n B'}{A \to B <:_n A' \to B'} \qquad \frac{}{A <:_n \star}$$

Safe cast $\boxed{(A \overset{p}{\Longrightarrow} B)\ \mathsf{safe}_{\mathsf{B}}\ q}$

$$\frac{A <:^+ B}{(A \overset{p}{\Longrightarrow} B)\ \mathsf{safe}\ p} \qquad \frac{A <:^- B}{(A \overset{\overline{p}}{\Longrightarrow} B)\ \mathsf{safe}\ p}$$

$$\frac{p \neq q \quad \overline{p} \neq q}{(A \overset{q}{\Longrightarrow} B)\ \mathsf{safe}\ p}$$

Figure 2. Subtyping and blame safety

contravariant in the function domain and covariant in the function range (as per (WRAP)). A cast from ground type to dynamic type never yields blame (as per (COLLAPSE) and (CONFLICT)). A cast to the dynamic type never yields positive blame, while a cast from the dynamic type never yields negative blame.

Naive subtyping is characterised by *covariance*. A base type is as precise as itself, precision of function types is covariant in *both* the domain and range of functions, and the dynamic type is the least precise type.

These four relations are closely connected: ordinary subtyping decomposes into positive and negative subtyping, which can be reassembled to yield naive subtyping, almost like a tangram.

Lemma 3 (Tangram)**.**

1. *$A <: B$ iff $A <:^+ B$ and $A <:^- B$.*
2. *$A <:_n B$ iff $A <:^+ B$ and $B <:^- A$.*

A cast from A to B decorated with p is *safe* for blame q, written

$$(A \overset{p}{\Longrightarrow} B)\ \mathsf{safe}_{\mathsf{B}}\ q$$

if evaluation of the cast can never yield blame q. The three rules reflect that if $A <:^+ B$ the cast never allocates positive blame, if $A <:^- B$ the cast never allocates negative blame, and a cast with label p never allocates blame other than to p or $\overline{p}$. Safety extends to terms in the obvious way: we write $M\ \mathsf{safe}_{\mathsf{B}}\ q$ if every cast in M is safe for q. Blame safety is established via a variant of preservation and progress.

Proposition 4 (Blame safety for the blame calculus)**.**

1. *If $M\ \mathsf{safe}_{\mathsf{B}}\ p$ and $M \longrightarrow_{\mathsf{B}} N$ then $N\ \mathsf{safe}_{\mathsf{B}}\ p$.*
2. *If $M\ \mathsf{safe}_{\mathsf{B}}\ p$ then $M \not\longrightarrow_{\mathsf{B}} \mathtt{blame}\ p$.*

The same result will apply, mutatis mutandis, for $\lambda\mathsf{C}$ and $\lambda\mathsf{T}$.

3.2 Failing cast

The following lemma about failing casts is useful when considering the relation between casts and coercions.

Lemma 5 (Failing cast)**.** *If $A \neq \star$, $A \sim G$, and $G \neq H$, then*

$$V : A \overset{p_1}{\Longrightarrow} G \overset{p_2}{\Longrightarrow} \star \overset{p_3}{\Longrightarrow} H \overset{p_4}{\Longrightarrow} B \longrightarrow^* \mathtt{blame}\ p_3$$

3.3 Contextual equivalence

Contextual equivalence is defined as usual. Evaluating a term may have three outcomes: converge, allocate blame to p, or diverge. Two terms are contextually equivalent if they have the same outcome in any context.

Let $\mathcal{C}$ range over contexts, which are expressions with holes in any position. Write $M{\uparrow_{\mathsf{B}}}$ if M diverges; coinductively, $M{\uparrow_{\mathsf{B}}}$ if $M \longrightarrow_{\mathsf{B}} N$ and $N{\uparrow_{\mathsf{B}}}$.

Definition 6 (Contextual equivalence)**.** *Two terms are contextually equivalent, written $M \overset{\mathsf{ctx}}{=}_{\mathsf{B}} N$, if for any context $\mathcal{C}$, either*

1. *both converge to a value, $\mathcal{C}[M] \longrightarrow_{\mathsf{B}} V$ and $\mathcal{C}[N] \longrightarrow_{\mathsf{B}} W$ for some values V and W.*
2. *both allocate blame to the same label, $\mathcal{C}[M] \longrightarrow_{\mathsf{B}} \mathtt{blame}\, p$ and $\mathcal{C}[N] \longrightarrow_B \mathtt{blame}\ p$ for some label p, or*
3. *both diverge, $\mathcal{C}[M]{\uparrow_{\mathsf{B}}}$ and $\mathcal{C}[N]{\uparrow_{\mathsf{B}}}$.*

The same definition will apply, mutatis mutandis, for $\lambda\mathsf{C}$, $\lambda\mathsf{T}$, and $\lambda\mathsf{W}$.

4. Coercion calculus

Figure 3 defines the coercion calculus, $\lambda\mathsf{C}$. The coercions of this calculus correspond to those of Henglein (1994), except that the coercion from dynamic type to ground type is decorated with a blame label, as in Siek and Wadler (2010). Blame labels and types are as in $\lambda\mathsf{B}$.

Let c, d range over coercions. We write $c : A \Longrightarrow B$ to indicate that c coerces values of type A to type B. The identity coercion at type A is written $\mathtt{id}_A$. The injection from ground type G to dynamic type is written $G\mathtt{!}$, and projection from dynamic type to ground type G is written $G\mathtt{?}^p$. The latter is decorated with a label p, to which blame is allocated if the projection fails. A function coercion $c \to d$ coerces a function $A \to B$ to a function $A' \to B'$, where c coerces A' to A, and d coerces B to B'. This construct is contravariant in the domain coercion c and covariant in the range coercion d. The composition $c\,;d$ coerces A to C, where c coerces A to B, and d coerces B to C. The fail coercion $\bot^{GpH}$ represents the result of a failed coercion from ground type G to ground type H, and is introduced because it is essential to the space-efficient representation described in the following section.

Terms of the calculus are as before, except that we replace casts by application of a coercion, written $M\langle c\rangle$. The typing rule for coercion application is straightforward:

$$\frac{\Gamma \vdash_{\mathsf{C}} M : A \qquad c : A \Longrightarrow B}{\Gamma \vdash_{\mathsf{C}} M\langle c\rangle : B}$$

If term M has type A, and c coerces A to B, then application to M of c is a term of type B.

Every well-typed coercion not containing failure has a unique type: if $c : A \Longrightarrow B$ and $c : A' \Longrightarrow B'$ and c does not contain a coercion of the form $\bot^{GpH}$ then $A = A'$ and $B = B'$. The converse is false: for instance $\mathtt{id}_\star$ and $G\mathtt{?}^p\ ;\ G\mathtt{!}$ both have type $\star \Longrightarrow \star$.

Values and evaluation contexts are as in the blame calculus, except that casts are replaced by corresponding coercions. We write $M \longrightarrow_{\mathsf{C}} N$ to indicate that term M steps to term N. Rule (ID): the identity coercion leaves a value unchanged. Rule (WRAP): a

Syntax

$$c, d ::= \mathtt{id}_A \mid G! \mid G?^p \mid c \to d \mid c\,;d \mid \bot^{GpH}$$
$$L, M, N ::= k \mid op(\vec{M}) \mid x \mid \lambda x{:}A.\, N \mid L\ M \mid M\langle c\rangle \mid \mathtt{blame}\ p$$
$$V, W ::= k \mid \lambda x{:}A.\, N \mid V\langle c \to d\rangle \mid V\langle G!\rangle$$
$$\mathcal{E} ::= \Box \mid \mathcal{E}[op(\vec{V}, \Box, \vec{M})] \mid \mathcal{E}[\Box\ M] \mid \mathcal{E}[V\ \Box] \mid \mathcal{E}[\Box\langle c\rangle]$$

Coercion typing $\boxed{c : A \Longrightarrow B}$

$$\frac{}{\mathtt{id}_A : A \Longrightarrow A}$$
$$\frac{}{G! : G \Longrightarrow \star} \qquad \frac{}{G?^p : \star \Longrightarrow G}$$
$$\frac{c : A' \Longrightarrow A \quad d : B \Longrightarrow B'}{(c \to d) : A \to B \Longrightarrow A' \to B'}$$
$$\frac{c : A \Longrightarrow B \quad d : B \Longrightarrow C}{(c\,;d) : A \Longrightarrow C}$$
$$\frac{A \neq \star \quad A \sim G \quad G \neq H}{\bot^{GpH} : A \Longrightarrow B}$$

Term typing $\boxed{\Gamma \vdash_{\mathsf{C}} M : A}$

$$\frac{\Gamma \vdash M : A \quad c : A \Longrightarrow B}{\Gamma \vdash M\langle c\rangle : B} \qquad \frac{}{\Gamma \vdash \mathtt{blame}\ p : A}$$

Reduction $\boxed{M \longrightarrow_{\mathsf{C}} N}$

$$\mathcal{E}[V\langle \mathtt{id}_A\rangle] \longrightarrow \mathcal{E}[V] \qquad \text{(ID)}$$
$$\mathcal{E}[(V\langle c \to d\rangle)\ W] \longrightarrow \mathcal{E}[(V\ (W\langle c\rangle))\langle d\rangle] \qquad \text{(WRAP)}$$
$$\mathcal{E}[V\langle G!\rangle\langle G?^p\rangle] \longrightarrow \mathcal{E}[V] \qquad \text{(COLLAPSE)}$$
$$\mathcal{E}[V\langle G!\rangle\langle H?^p\rangle] \longrightarrow \mathtt{blame}\ p \quad \text{if } G \neq H \qquad \text{(CONFLICT)}$$
$$\mathcal{E}[V\langle c\,;d\rangle] \longrightarrow \mathcal{E}[V\langle c\rangle\langle d\rangle] \qquad \text{(DECOMPOSE)}$$
$$\mathcal{E}[V\langle \bot^{GpH}\rangle] \longrightarrow \mathtt{blame}\ p \qquad \text{(FAIL)}$$

Safe coercion $\boxed{c\ \mathsf{safe}_{\mathsf{C}}\ q}$

$$\frac{}{\mathtt{id}_A\ \mathsf{safe}\ q} \qquad \frac{}{G!\ \mathsf{safe}\ q} \qquad \frac{p \neq q}{G?^p\ \mathsf{safe}\ q}$$
$$\frac{c\ \mathsf{safe}\ q \quad d\ \mathsf{safe}\ q}{c \to d\ \mathsf{safe}\ q} \qquad \frac{c\ \mathsf{safe}\ q \quad d\ \mathsf{safe}\ q}{c\,;d\ \mathsf{safe}\ q} \qquad \frac{p \neq q}{\bot^{GpH}\ \mathsf{safe}\ q}$$

Figure 3. Coercion calculus ($\lambda\mathsf{C}$)

coercion of a function applied to a value reduces to a term that coerces on the domain, applies the function, and coerces on the range. Rule (COLLAPSE): if an injection meets a matching projection, the coercion leaves the value unchanged. Rule (CONFLICT): if an injection meets an incompatible projection, the coercion fails and allocates blame to the label in the projection. (Here it is clear why blame falls on the outer coercion: the inner coercion is an injection and has no blame label, while the outer is a projection with a blame label.) Rule (DECOMPOSE): application of a composed coercion applies each of the coercions in turn.

A coercion c is *safe* for blame label q, written $c\ \mathsf{safe}_{\mathsf{C}}\ q$, if application of the coercion never allocates blame to q. The definition is pleasingly simple: a coercion is safe for q if it does not mention label q.

Type and blame safety and contextual equivalence for $\lambda\mathsf{C}$ are as in $\lambda\mathsf{B}$. Propositions 2 and 4 and Definition 6 apply mutatis mutandis.

4.1 Translation from blame to coercions

The relation between $\lambda\mathsf{B}$ and $\lambda\mathsf{C}$ is presented in Figure 4. In this section, we let M, N range over terms of $\lambda\mathsf{B}$ and M', N' range over terms of $\lambda\mathsf{C}$.

We write

$$|A \xRightarrow{p} B|^{\mathsf{BC}} = c$$

to indicate that the cast on the left translates to the coercion on the right. The translation is carefully designed to ensure there is a lockstep bisimulation between $\lambda\mathsf{B}$ and $\lambda\mathsf{C}$. The translation extends to terms in the obvious way, replacing each cast by the corresponding coercion.

We write

$$|c|^{\mathsf{CB}} = Z$$

to indicate that the coercion on the left translates to the sequence of casts on the right. Here Z ranges over sequences of casts. As defined in Figure 4, we write $Z \to B$ (respectively $B \to Z$) to replace in Z each source or target type A by $A \to B$ (respectively $B \to A$), we write $\overline{Z}$ to reverse the sequence Z and complement all the blame labels, and we write $Z +\!\!+ Z'$ to concatenate the two sequences Z and Z', where the last type of one sequence must match the first of the other. In the clause for $c \to d$, the right-hand side can be taken as either

$$\overline{(|c|^{\mathsf{CB}} \to B)} +\!\!+ (A' \to |d|^{\mathsf{CB}}) \text{ or } (A \to |d|^{\mathsf{CB}}) +\!\!+ \overline{(|c|^{\mathsf{CB}} \to B')},$$

equivalently. We write $\bullet$ as a blame label in casts where the label is irrelevant because the cast cannot allocate blame. The translation extends to terms in the obvious way, replacing each coercion by the corresponding sequence of casts.

As a first step justifying this definition, we observe several contextual equivalences for $\lambda\mathsf{C}$.

Lemma 7 (Equivalences)**.** *The following hold in* $\lambda\mathsf{C}$*.*

1. $M\langle \mathtt{id}\rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M$
2. $M\langle c\,;d\rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle c\rangle\langle d\rangle$
3. $M\langle c \to d\rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle (c \to \mathtt{id})\,;(\mathtt{id} \to d)\rangle$
4. $M\langle c \to d\rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle (\mathtt{id} \to c)\,;(d \to \mathtt{id})\rangle$

The proof of this lemma is deferred to Section 7.1, where we apply a new technique that makes the proof straightforward.

Translating from $\lambda\mathsf{C}$ to $\lambda\mathsf{B}$ and back again is the identity, up to contextual equivalence.

Lemma 8 (Coercions to blame)**.** *If* M' *is a term of* $\lambda\mathsf{C}$ *then* $||M'|^{\mathsf{CB}}|^{\mathsf{BC}} \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M'$*.*

Proof. By induction on M', using case analysis of the clauses in the definition of $|\cdot|^{\mathsf{CB}}$. In particular, the clause for $\mathtt{id}$ is justified by part 1 of Lemma 7, the clause for $c\,;d$ is justified by part 2 of the same lemma, the clause for $c \to d$ is justified by part 3 of the same lemma, and the alternative definition for the clause is justified by part 4. The clause for $\bot^{GpH}$ is justified by Lemma 5. □

4.2 Preservation of blame safety

The somewhat subtle definition of positive and negative subtyping is justified by the correspondence to the coercion calculus. It is not too surprising that the definition is sound (safety in B implies safety in C), but it is surprising that the definition is also complete (safety in C implies safety in B).

Lemma 9 (Positive and negative subtyping)**.**

1. $A <:^{+} B$ *iff* $|A \xRightarrow{p} B|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ p$*.*
2. $A <:^{-} B$ *iff* $|A \xRightarrow{p} B|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$*.*

Blame to coercion ($\lambda\mathsf{B}$ to $\lambda\mathsf{C}$) $\boxed{|A \xRightarrow{p} B|^{\mathsf{BC}} = c}$

$$
\begin{aligned}
|K \xRightarrow{p} K|^{\mathsf{BC}} &= \mathtt{id}_K \\
|A \to B \xRightarrow{p} A' \to B'|^{\mathsf{BC}} & \\
&= |A' \xRightarrow{\overline{p}} A|^{\mathsf{BC}} \to |B \xRightarrow{p} B'|^{\mathsf{BC}} \\
|\star \xRightarrow{p} \star|^{\mathsf{BC}} &= \mathtt{id}_\star \\
|G \xRightarrow{p} \star|^{\mathsf{BC}} &= G\mathtt{!} \\
|A \xRightarrow{p} \star|^{\mathsf{BC}} &= |A \xRightarrow{p} G|^{\mathsf{BC}} \,;\, G\mathtt{!} \qquad \text{if } A \neq \star, A \neq G, A \sim G \\
|\star \xRightarrow{p} G|^{\mathsf{BC}} &= G\mathtt{?}^p \\
|\star \xRightarrow{p} A|^{\mathsf{BC}} &= G\mathtt{?}^p \,;\, |G \xRightarrow{p} A|^{\mathsf{BC}} \qquad \text{if } A \neq \star, A \neq G, A \sim G
\end{aligned}
$$

Coercion to blame ($\lambda\mathsf{C}$ to $\lambda\mathsf{B}$) $\boxed{|c|^{\mathsf{CB}} = Z}$

$$
\begin{aligned}
|\mathtt{id}_A|^{\mathsf{CB}} &= [\,] \\
|G\mathtt{!}|^{\mathsf{CB}} &= [G \xRightarrow{\bullet} \star] \\
|G\mathtt{?}^p|^{\mathsf{CB}} &= [\star \xRightarrow{p} G] \\
|c \to d|^{\mathsf{CB}} &= \overline{(|c|^{\mathsf{CB}} \to B)} +\!\!+ (A' \to |d|^{\mathsf{CB}}) \\
&\qquad \text{where } c \to d : A \to B \Longrightarrow A' \to B' \\
|c \,;\, d|^{\mathsf{CB}} &= |c|^{\mathsf{CB}} +\!\!+ |d|^{\mathsf{CB}} \\
|\bot^{GpH}_{A \Longrightarrow B}|^{\mathsf{CB}} &= [A \xRightarrow{\bullet} G, G \xRightarrow{\bullet} \star, \star \xRightarrow{p} H, H \xRightarrow{\bullet} B]
\end{aligned}
$$

where if

$$
\begin{aligned}
Z &= [A_1 \xRightarrow{p_1} A_2, \cdots, A_m \xRightarrow{p_m} A_{m+1}] \\
Z' &= [A_{m+1} \xRightarrow{p_{m+1}} A_{m+2}, \cdots, A_{m+n} \xRightarrow{p_{m+n}} A_{m+n+1}]
\end{aligned}
$$

then

$$
\begin{aligned}
Z \to B &= [A_1 \to B \xRightarrow{p_1} A_2 \to B, \cdots, A_m \to B \xRightarrow{p_m} A_{m+1} \to B] \\
B \to Z &= [B \to A_1 \xRightarrow{p_1} B \to A_2, \cdots, B \to A_m \xRightarrow{p_m} B \to A_{m+1}] \\
\overline{Z} &= [A_{m+1} \xRightarrow{\overline{p_m}} A_m, \cdots, A_2 \xRightarrow{\overline{p_1}} A_1] \\
Z +\!\!+ Z' &= [A_1 \xRightarrow{p_1} A_2, \cdots, A_{m+n} \xRightarrow{p_{m+n}} A_{m+n+1}]
\end{aligned}
$$

Figure 4. Relating blame and coercions ($\lambda\mathsf{B}$ and $\lambda\mathsf{C}$)

(The full proof is in the supplementary material.)

It follows immediately that the translation from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ preserves type and blame safety.

Proposition 10 (Preservation, blame to coercions)**.**

1. *If* $\Gamma \vdash_{\mathsf{B}} M : A$ *then* $\Gamma \vdash_{\mathsf{C}} |M|^{\mathsf{BC}} : A$.
2. *If* M $\mathsf{safe}_{\mathsf{B}}$ p *then* $|M|^{\mathsf{BC}}$ $\mathsf{safe}_{\mathsf{C}}$ p.

4.3 Bisimulation

The translation from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ is a bisimulation.

Proposition 11 (Bisimulation, blame to coercions)**.**
Assume $\vdash_{\mathsf{B}} M : A$ *and* $\vdash_{\mathsf{C}} M' : A$ *and* $|M|^{\mathsf{BC}} = M'$.

1. *If* $M \longrightarrow_{\mathsf{B}} N$ *then* $M' \longrightarrow_{\mathsf{C}} N'$ *and* $|N|^{\mathsf{BC}} = N'$ *for some* N'.
2. *If* $M' \longrightarrow_{\mathsf{C}} N'$ *then* $M \longrightarrow_{\mathsf{B}} N$ *and* $|N|^{\mathsf{BC}} = N'$ *for some* N.
3. *If* $M = V$ *then* $M' = V'$ *and* $|V|^{\mathsf{BC}} = V'$ *for some* V'.
4. *If* $M' = V'$ *then* $M = V$ *and* $|V|^{\mathsf{BC}} = V'$ *for some* V.
5. *If* $M = \mathtt{blame}\ p$ *then* $M' = \mathtt{blame}\ p$.
6. *If* $M' = \mathtt{blame}\ p$ *then* $M = \mathtt{blame}\ p$.

The bisimulation is lockstep, in that a single step in $\lambda\mathsf{B}$ corresponds to a single step in $\lambda\mathsf{C}$, and vice versa.

4.4 Fully abstract

The translation from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ is fully abstract.

Proposition 12 (Fully abstract, blame to coercions)**.** *If* M *and* N *are terms of* $\lambda\mathsf{B}$ *then* $M \overset{\mathsf{ctx}}{=}_{\mathsf{B}} N$ *iff* $|M|^{\mathsf{BC}} \overset{\mathsf{ctx}}{=}_{\mathsf{C}} |N|^{\mathsf{BC}}$.

Proof. In the backward direction, the result follows from Proposition 11, clauses 2, 4, and 6, translating each $\lambda\mathsf{B}$ context $\mathcal{C}$ to the $\lambda\mathsf{C}$ context $|\mathcal{C}|^{\mathsf{BC}}$. In the forward direction, the result follows from Proposition 11, clauses 1, 3, and 5, applying Lemma 8 to ensure that each context $\mathcal{C}'$ in $\lambda\mathsf{C}$ corresponds to a context $\mathcal{C}$ in $\lambda\mathsf{B}$. □

5. Threesome calculus

Figure 5 defines the threesome calculus, $\lambda\mathsf{T}$. Threesomes correspond to coercions in a newly identified canonical form, which we call *threesome coercions*. Threesomes are introduced by Siek and Wadler (2010), and their relation to coercions is discussed there and by Garcia (2013). In threesomes without blame a threesome consists of three types: source type, mediating type, and target type. In threesomes with blame, the mediating type is decorated with blame labels, and the correspondence with coercions is specified by a translation. Here threesomes are presented as coercions in canonical form, and the correspondence with threesomes without blame is specified by a translation (given in the next section).

Blame labels and types are as in $\lambda\mathsf{B}$.

Threesome coercions follow a specific, three-part grammar to ensure that they are canonical in the sense that there is one threesome coercion for each equivalence class of coercions with respect to the equational theory of Henglein (1994). Threesome coercions can be seen as an adaptation of Henglein's *canonical coercions* to facilitate the definition of a recursive normalization function.

Let s, t range over threesome coercions, i range over suffix coercions, and g, h range over ground coercions. Threesome coercions are either the identity coercion at dynamic type $\mathtt{id}_\star$, a projection followed by a suffix coercion $(G\mathtt{?}^p \,;\, i)$, or just a suffix coercion i. A suffix coercion is either a ground coercion followed by an injection $(g \,;\, G\mathtt{!})$, just a ground coercion g, or the failure coercion $\bot^{GpH}$. A ground coercion is an identity coercion of base type $\mathtt{id}_K$ or a function coercion $s \to t$.

The source of a suffix coercion is never the dynamic type. The source and target of a ground coercion are never the dynamic type, and both are always compatible with the same unique ground type.

Lemma 13 (Source and Target)**.**

1. *If* $i : A \Longrightarrow B$ *then* $A \neq \star$.
2. *If* $g : A \Longrightarrow B$ *then* $A \neq \star$ *and* $B \neq \star$ *and there exists a unique* G *such that* $A \sim G$ *and* $G \sim B$.

Terms of the calculus are as in $\lambda\mathsf{C}$, except that we restrict coercions to threesome coercions.

The key idea of the dynamics, as in Herman et al. (2007, 2010) and Siek and Wadler (2010), is to combine and normalize adjacent coercions, which ensures space efficiency. Ensuring adjacent coercions are combined requires we adjust the notion of value and evaluation context. Let U range over uncoerced values and V, W range over values, where an uncoerced value contains no top-level coercion and a value at most one. Let $\mathcal{F}$ range over cast-free contexts and $\mathcal{E}$ range over contexts, where the innermost term of a cast-free context is not a cast. Reduction of a term that is a cast must occur in a cast-free context. These adjustments ensure that if a term contains two casts in succession in an evaluation context, that those

Syntax

$$
\begin{aligned}
r, s, t &::= \mathtt{id}_\star \mid (G?^p \,;\, i) \mid i \\
i &::= (g \,;\, G!) \mid g \mid \bot^{GpH} \\
g, h &::= \mathtt{id}_K \mid s \to t \\
L, M, N &::= k \mid op(\vec{M}) \mid x \mid \lambda x{:}A.\, N \mid L\ M \mid M\langle t \rangle \mid \mathtt{blame}\ p \\
U &::= k \mid \lambda x{:}A.\, N \\
V, W &::= U \mid U\langle s \to t \rangle \mid U\langle g \,;\, G! \rangle \\
\mathcal{F} &::= \Box \mid \mathcal{E}[op(\vec{V}, \Box, \vec{M})] \mid \mathcal{E}[\Box\ M] \mid \mathcal{E}[V\ \Box] \\
\mathcal{E} &::= \mathcal{F} \mid \mathcal{F}[\Box\langle t \rangle]
\end{aligned}
$$

Threesome composition $\boxed{s \fatsemi t = r}$

$$
\begin{aligned}
\mathtt{id}_K \fatsemi \mathtt{id}_K &= \mathtt{id}_K \\
(s \to t) \fatsemi (s' \to t') &= (s' \fatsemi s) \to (t \fatsemi t') \\
\mathtt{id}_\star \fatsemi t &= t \\
(g \,;\, G!) \fatsemi \mathtt{id}_\star &= g \,;\, G! \\
(G?^p \,;\, i) \fatsemi t &= G?^p \,;\, (i \fatsemi t) \\
g \fatsemi (h \,;\, H!) &= (g \fatsemi h) \,;\, H! \\
(g \,;\, G!) \fatsemi (G?^p \,;\, i) &= g \fatsemi i \\
(g \,;\, G!) \fatsemi (H?^p \,;\, i) &= \bot^{GpH} \qquad \text{if } G \neq H \\
\bot^{GpH} \fatsemi s &= \bot^{GpH} \\
g \fatsemi \bot^{GpH} &= \bot^{GpH}
\end{aligned}
$$

Reduction $\boxed{M \longrightarrow_{\mathsf{T}} N}$

$$
\begin{aligned}
\mathcal{E}[(U\langle s \to t \rangle)\ V] &\longrightarrow \mathcal{E}[(U\ (V\langle s \rangle))\langle t \rangle] && \text{(WRAP)} \\
\mathcal{F}[U\langle \mathtt{id}_K \rangle] &\longrightarrow \mathcal{F}[U] && \text{(BASE)} \\
\mathcal{F}[U\langle \mathtt{id}_\star \rangle] &\longrightarrow \mathcal{F}[U] && \text{(STAR)} \\
\mathcal{F}[M\langle s \rangle\langle t \rangle] &\longrightarrow \mathcal{F}[M\langle s \fatsemi t \rangle] && \text{(COMPOSE)} \\
\mathcal{F}[U\langle \bot^{GpH} \rangle] &\longrightarrow \mathtt{blame}\ p && \text{(FAIL)}
\end{aligned}
$$

Figure 5. Threesome calculus ($\lambda\mathsf{T}$)

casts are reduced by rule (COMPOSE) before other reductions occur. The other reduction rules are straightforward.

Composition and normalization of threesome coercions is computed by $\cdot \fatsemi \cdot$. If threesomes s and t are equivalent to coercions c and d, then the threesome $s \fatsemi t$ is equivalent to the coercion $c \,;\, d$. The grammar of threesome coercions makes the definition of threesome composition easy to express. In fact, it is a structurally recursive function, so termination is immediate. Further the correctness of each equation in the definition is easily justified by the equational theory of Henglein (1994).

Type and blame safety and contextual equivalence for $\lambda\mathsf{C}$ are as in $\lambda\mathsf{B}$. The definition of blame safety is as in Figure 3, and Propositions 2 and 4 and Definition 6 apply mutatis mutandis.

5.1 Translation from coercions to threesomes

The translation from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$ is presented in Figure 6. In this section, we let M, N range over terms of $\lambda\mathsf{C}$ and let M', N' range over terms of $\lambda\mathsf{T}$.

We write

$$|c|^{\mathsf{CT}} = t$$

to indicate that the coercion on the left translates to the threesome coercion on the right. The translation extends to terms in the obvious way, replacing each coercions by the corresponding threesome coercion.

The translation from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$ preserves type and blame safety.

Proposition 14 (Preservation, coercions to threesomes)**.**

1. *If* $\Gamma \vdash_{\mathsf{C}} M : A$ *then* $\Gamma \vdash_{\mathsf{T}} |M|^{\mathsf{CT}} : A$.
2. *If* M $\mathsf{safe}_{\mathsf{C}}\ p$ *then* $|M|^{\mathsf{CT}}$ $\mathsf{safe}_{\mathsf{T}}\ p$.

The dynamics of the threesome calculus differ significantly from that of the other two calculi because $\lambda\mathsf{T}$ compresses sequences of casts into a single threesome. We define a bisimulation $\approx$ that relates the coercion calculus and the threesome calculus. The rules relate a sequence of zero or more coercion applications to a single threesome application. For example, consider the sequence of (WRAP) reductions in $\lambda\mathsf{C}$.

$$
\begin{aligned}
& (V\langle c_1 \to d_1 \rangle\langle c_2 \to d_2 \rangle\langle c_3 \to d_3 \rangle)\ W && (1) \\
\longrightarrow_{\mathsf{C}}\ & ((V\langle c_1 \to d_1 \rangle\langle c_2 \to d_2 \rangle)\ (W\langle c_3 \rangle))\langle d_3 \rangle && (2) \\
\longrightarrow_{\mathsf{C}}\ & ((V\langle c_1 \to d_1 \rangle)\ (W\langle c_3 \rangle\langle c_2 \rangle))\langle d_2 \rangle\langle d_3 \rangle && (3) \\
\longrightarrow_{\mathsf{C}}\ & (V\ (W\langle c_3 \rangle\langle c_2 \rangle\langle c_1 \rangle))\langle d_1 \rangle\langle d_2 \rangle\langle d_3 \rangle && (4)
\end{aligned}
$$

If $V \approx V'$, $W \approx W'$, $|c_i|^{\mathsf{CT}} = s_i$, and $|d_i|^{\mathsf{CT}} = t_i$, this relates to a single (WRAP) reduction in $\lambda\mathsf{T}$.

$$
\begin{aligned}
& (V\langle (s_3 \fatsemi s_2 \fatsemi s_1) \to (t_1 \fatsemi t_2 \fatsemi t_3) \rangle)\ W && (1') \\
\longrightarrow_{\mathsf{T}}\ & (V\ (W\langle s_3 \fatsemi s_2 \fatsemi s_1 \rangle)\langle t_1 \fatsemi t_2 \fatsemi t_3 \rangle && (2')
\end{aligned}
$$

Here (1) $\approx$ (1') via one use of rule (ID) and three uses of rule (MANY); and (2) $\approx$ (1') via one use of rule (ID), two uses of rule (MANY), and one use of rule (MANYAPP); and (3) $\approx$ (1') via one use of rule (ID), one use of rule (MANY), and two uses of rule (MANYAPP); and (4) $\approx$ (2') via one use of rule (ID) and three uses of rule (MANY) again, for both the domain and the range.

Terms translated from coercions to threesomes are related by $\approx$.

Proposition 15. $M \approx |M|^{\mathsf{CT}}$.

The $\approx$ relation is a bisimulation.

Proposition 16 (Bisimulation, coercions to threesomes)**.**
Assume $\vdash_{\mathsf{C}} M : A$ *and* $\vdash_{\mathsf{T}} M' : A$ *and* $|M|^{\mathsf{CT}} = M'$.

1. *If* $M \longrightarrow_{\mathsf{C}} N$ *then* $M' \longrightarrow_{\mathsf{T}}^* N'$ *and* $N \approx N'$ *for some* N'.
2. *If* $M' \longrightarrow_{\mathsf{T}} N'$ *then* $M \longrightarrow_{\mathsf{C}}^* N$ *and* $N \approx N'$ *for some* N.
3. *If* $M = V$ *then* $M' \longrightarrow_{\mathsf{T}}^* V'$ *and* $V \approx V'$ *for some* V'.
4. *If* $M' = V'$ *then* $M \longrightarrow_{\mathsf{C}}^* V$ *and* $V \approx V'$ *for some* V.
5. *If* $M = \mathtt{blame}\ p$ *then* $M' = \mathtt{blame}\ p$.
6. *If* $M' = \mathtt{blame}\ p$ *then* $M = \mathtt{blame}\ p$.

The bisimulation is not lockstep: a single step in $\lambda\mathsf{C}$ corresponds to zero or more steps in $\lambda\mathsf{T}$, and vice versa.

(The full proof is in the supplementary material.)

5.2 Fully abstract

The translation from $\lambda\mathsf{C}$ to $\lambda\mathsf{T}$ is fully abstract.

Proposition 17 (Fully abstract, coercions to threesomes)**.** *If* M *and* N *are terms of* $\lambda\mathsf{C}$ *then* $M \overset{\mathsf{ctx}}{=}_{\mathsf{C}} N$ *iff* $|M|^{\mathsf{CT}} \overset{\mathsf{ctx}}{=}_{\mathsf{T}} |N|^{\mathsf{CT}}$.

Proof. In the backward direction, the result follows immediately from Proposition 16, clauses 2, 4, and 6, translating each $\lambda\mathsf{C}$ context $\mathcal{C}$ to the $\lambda\mathsf{T}$ context $|\mathcal{C}|^{\mathsf{CT}}$. In the forward direction, the result follows from Proposition 16, clauses 1, 3, and 5, utilising the fact that each $\lambda\mathsf{T}$ context $\mathcal{C}'$ is also a $\lambda\mathsf{C}$ context. □

Coercions to threesomes ($\lambda\mathsf{C}$ to $\lambda\mathsf{T}$) $\boxed{|c|^{\mathsf{CT}} = t}$

$$
\begin{aligned}
|\mathtt{id}_\star|^{\mathsf{CT}} &= \mathtt{id}_\star \\
|\mathtt{id}_K|^{\mathsf{CT}} &= \mathtt{id}_K \\
|\mathtt{id}_{A\to B}|^{\mathsf{CT}} &= |\mathtt{id}_A|^{\mathsf{CT}} \to |\mathtt{id}_B|^{\mathsf{CT}} \\
|G?^p|^{\mathsf{CT}} &= G?^p \mathbin{;} |\mathtt{id}_G|^{\mathsf{CT}} \\
|G!|^{\mathsf{CT}} &= |\mathtt{id}_G|^{\mathsf{CT}} \mathbin{;} G! \\
|c \to d|^{\mathsf{CT}} &= |c|^{\mathsf{CT}} \to |d|^{\mathsf{CT}} \\
|c \mathbin{;} d|^{\mathsf{CT}} &= |c|^{\mathsf{CT}} \fatsemi |d|^{\mathsf{CT}} \\
|\bot^{GpH}|^{\mathsf{CT}} &= \bot^{GpH}
\end{aligned}
$$

Bisimulation between $\lambda\mathsf{C}$ and $\lambda\mathsf{T}$ $\boxed{M \approx_{\mathsf{CT}} M'}$

$$\frac{}{k \approx k} \qquad \frac{\vec{M} \approx \vec{M'}}{op(\vec{M}) \approx op(\vec{M'})} \qquad \frac{}{x \approx x}$$

$$\frac{M \approx M'}{\lambda x{:}A.\, M \approx \lambda x{:}A.\, M'} \qquad \frac{L \approx L' \quad M \approx M'}{L\ M \approx L'\ M'}$$

$$\frac{}{\mathtt{blame}\ p \approx \mathtt{blame}\ p}$$

$$\frac{M \approx M' \quad \vdash M : A \quad |\mathtt{id}_A|^{\mathsf{CT}} = s}{M \approx M'\langle s \rangle} \tag{ID}$$

$$\frac{M \approx M'\langle s \rangle \quad |c|^{\mathsf{CT}} = t}{M\langle c \rangle \approx M'\langle s \fatsemi t \rangle} \tag{MANY}$$

$$\frac{M \approx (L'\langle r \rangle)\ (M'\langle s \rangle) \quad |d|^{\mathsf{CT}} = t}{M\langle d \rangle \approx (L'\langle r \fatsemi (s \to t) \rangle)\ M'} \tag{MANYAPP}$$

Figure 6. Relating coercions to threesomes ($\lambda\mathsf{C}$ and $\lambda\mathsf{T}$)

6. Threesomes without blame

Siek and Wadler (2010) use a different development than the one given here. They first introduce threesomes as a pair of casts, from a source type through a mediating type to a target type—the three types explaining for the name. This form does not account for blame, which they restore by decorating the mediating cast with blame labels. In contrast, here coercions directly inspired threesome coercions. We now recover the three types, at the expense of no longer accounting for blame.

To account for the case where the source and target type are incompatible, threesomes require introducing a new type $\bot$, which is the lowest type in the naive ordering. Siek and Wadler (2010) permits every type to include $\bot$, where here we offer a more refined development where $\bot$ appears only in the mediating type.

We let R, S, T range over threesome types, which consist of the usual type constructors together with $\bot$. Every ordinary type is a threesome type, but not conversely (because of $\bot$). A threesome coercions is written

$$M : A \overset{T}{\Longrightarrow} B$$

where M is a term, A and B are ordinary types, and T is a threesome type that is bounded above by A and B.

Threesome types S and T are shallowly incompatible, written $S \# T$, if they are different base types, if one is a base type and the other is a function, or if one is the empty type.

The meet of two type S and T is written $S \,\&\, T$ and defined in Figure 7. It is the greatest lower bound with regard to naive subtyping.

Lemma 18 (Meet is greatest lower bound)**.**

1. *$S \,\&\, T <:_n S$ and $S \,\&\, T <:_n T$, and*
2. *$R <:_n S$ and $R <:_n T$ iff $R <:_n S \,\&\, T$.*

Proof. Induction on the structure of S and T. □

The reduction rules for threesomes without blame are identical to those for threesomes, save that composition of threesome coercions ($s \fatsemi t$) is replaced by meet of threesome types ($S \,\&\, T$).

Type safety and contextual equivalence for $\lambda\mathsf{W}$ are as in $\lambda\mathsf{B}$. Proposition 2 and Definition 6 apply mutatis mutandis. Blame safety is not relevant for $\lambda\mathsf{W}$, because there are no blame labels.

6.1 Translation from threesomes with blame to those without

Ignoring blame labels, a threesome coercion is determined by its source, target, and mediating types. The mediating type of a threesome t is written $||t||$, and defined in Figure 7. Write $t^\bullet$ for the result of replacing each blame label in t by $\bullet$, where $\bullet$ is a special blame label satisfying $\overline{\bullet} = \bullet$. Write $|\cdot|^{\mathsf{BT}} = ||\cdot|^{\mathsf{BC}}|^{\mathsf{CT}}$ to translate from $\lambda\mathsf{B}$ to $\lambda\mathsf{T}$ via $\lambda\mathsf{C}$.

Lemma 19 (Mediating type)**.** *If $t : A \Longrightarrow B$ and $||t|| = T$ then $T <:_n A$ and $T <:_n B$ and*

$$t^\bullet = |A \overset{\bullet}{\Longrightarrow} T|^{\mathsf{BT}} \fatsemi |T \overset{\bullet}{\Longrightarrow} B|^{\mathsf{BT}}.$$

Proof. Induction on t. □

The correspondence between composition of threesome coercions and meet of threesome types is straightforward.

Lemma 20 (Composition and meet)**.** *If s and t are threesome coercions, then*

$$||s \fatsemi t|| = ||s|| \,\&\, ||t||$$

Proof. Induction on s and t. □

The above results suggest a simple translation. If t is a threesome coercion, $t : A \Longrightarrow B$, and $||t|| = T$, define

$$|t|^{\mathsf{TW}} = A \overset{T}{\Longrightarrow} B.$$

The translation extends to terms in the obvious way, replacing each threesome coercion by the corresponding threesome cast, and replacing $\mathtt{blame}\ p$ by $\mathtt{wrong}$.

Preservation of type safety for the translation of $\lambda\mathsf{T}$ to $\lambda\mathsf{W}$ is straightforward, and omitted. Since blame safety is not relevant for $\lambda\mathsf{W}$, neither is preservation of blame safety.

6.2 Bisimulation

The translation from $\lambda\mathsf{T}$ to $\lambda\mathsf{W}$ is a bisimulation.

Proposition 21 (Bisimulation, threesomes without blame)**.**
Assume $\vdash_{\mathsf{T}} M : A$ and $\vdash_{\mathsf{W}} M' : A$ and $|M|^{\mathsf{TW}} = M'$.

1. *If $M \longrightarrow_{\mathsf{T}} N$ then $M' \longrightarrow_{\mathsf{W}} N'$ and $|N|^{\mathsf{TW}} = N'$ for some N'.*
2. *If $M' \longrightarrow_{\mathsf{W}} N'$ then $M \longrightarrow_{\mathsf{T}} N$ and $|N|^{\mathsf{TW}} = N'$ for some N.*
3. *If $M = V$ then $M' = V'$ and $|V|^{\mathsf{TW}} = V'$ for some V'.*
4. *If $M' = V'$ then $M = V$ and $|V|^{\mathsf{TW}} = V'$ for some V.*
5. *If $M = \mathtt{blame}\ p$ then $M' = \mathtt{wrong}$.*
6. *If $M' = \mathtt{wrong}$ then $M = \mathtt{blame}\ p$ for some p.*

The bisimulation is lockstep, in that a single step in $\lambda\mathsf{T}$ corresponds to a single step in $\lambda\mathsf{W}$, and vice versa.

Proof. Follows immediately from Lemmas 19 and 20. □

Syntax

$$R, S, T ::= K \mid S \to T \mid \star \mid \bot$$

$$L, M, N ::= x|k|op(\vec{M})|\lambda x{:}A.\, N|L\ M \mid M : A \overset{T}{\Longrightarrow} B \mid \texttt{wrong}$$

$$V, W ::= U \mid U : A \to B \overset{S \to T}{\Longrightarrow} A' \to B' \mid U : A \overset{T}{\Longrightarrow} \star$$

$$\mathcal{E} ::= \mathcal{F} \mid \mathcal{F}[\Box : A \overset{T}{\Longrightarrow} B]$$

Naive subtype $\boxed{S <:_n T}$

$$\frac{}{K <:_n K} \qquad \frac{}{T <:_n \star} \qquad \frac{S <:_n S' \quad T <:_n T'}{S \to S' <:_n T \to T'} \qquad \frac{}{\bot <:_n T}$$

Term typing $\boxed{\Gamma \vdash_{\mathsf{W}} M : A}$

$$\frac{\Gamma \vdash M : A \quad T <:_n A \quad T <:_n B}{\Gamma \vdash M : A \overset{T}{\Longrightarrow} B} \qquad \frac{}{\Gamma \vdash \texttt{wrong} : A}$$

Shallow incompatibility $\boxed{S \# T}$

$$\frac{K \neq K'}{K \# K'} \qquad \frac{}{K \# S \to T} \qquad \frac{}{S \to T \# K} \qquad \frac{}{\bot \# T} \qquad \frac{}{T \# \bot}$$

Meet $\boxed{S \,\&\, T = R}$

$$\begin{aligned} K \,\&\, K &= K \\ \star \,\&\, T &= T \\ T \,\&\, \star &= T \\ (S \to T) \,\&\, (S' \to T') &= (S \,\&\, S') \to (T \,\&\, T') \\ S \,\&\, T &= \bot \qquad \text{if } S \# T \end{aligned}$$

Reduction $\boxed{M \longrightarrow_{\mathsf{W}} N}$

$$\mathcal{E}[(U : A \to B \overset{S \to T}{\Longrightarrow} A' \to B')\ V] \longrightarrow \mathcal{E}[(U\ (V : A' \overset{S}{\Longrightarrow} A)) : B \overset{T}{\Longrightarrow} B'] \quad (\textsc{Wrap})$$

$$\mathcal{F}[U : K \overset{K}{\Longrightarrow} K] \longrightarrow \mathcal{F}[U] \quad (\textsc{Base})$$

$$\mathcal{F}[U : \star \overset{\star}{\Longrightarrow} \star] \longrightarrow \mathcal{F}[U] \quad (\textsc{Star})$$

$$\mathcal{F}[M : A \overset{S}{\Longrightarrow} B \overset{T}{\Longrightarrow} C] \longrightarrow \mathcal{F}[M : A \overset{S \& T}{\Longrightarrow} C] \quad (\textsc{Compose})$$

$$\mathcal{F}[U : A \overset{\bot}{\Longrightarrow} B] \longrightarrow \texttt{wrong} \quad (\textsc{Fail})$$

Figure 7. Threesomes without blame ($\lambda\mathsf{W}$)

6.3 Fully abstract

The translation from $\lambda\mathsf{T}$ to $\lambda\mathsf{W}$ is fully abstract.

Proposition 22 (Fully abstract, threesomes without blame)**.** *If M and N are terms of $\lambda\mathsf{T}$ then $M^\bullet \overset{\mathsf{ctx}}{=}_{\mathsf{T}} N^\bullet$ iff $|M|^{\mathsf{TW}} \overset{\mathsf{ctx}}{=}_{\mathsf{W}} |N|^{\mathsf{TW}}$.*

Proof. In the backward direction, the result follows from Proposition 21, clauses 2, 4, and 6, translating each $\lambda\mathsf{T}$ context $\mathcal{C}$ to the $\lambda\mathsf{W}$ context $|\mathcal{C}|^{\mathsf{TW}}$. In the forward direction, the result follows from Proposition 21, clauses 1, 3, and 5, utilising the fact that $|\cdot|^{\mathsf{TW}}$ is a bijection to ensure that each context $\mathcal{C}'$ in $\lambda\mathsf{W}$ corresponds to a context $\mathcal{C}$ in $\lambda\mathsf{T}$. □

The development in this section is straightforward. Lemmas 19 and 20 are established by easy inductions, and Propositions 21 and 22 are straightforward. In contrast, the weaker correctness result of Siek and Wadler (2010) depends on the Fundamental Property of Casts. Establishing the Fundamental Property required a new bisimulation relation and three lemmas, and then establishing the weak correctness result requires a corollary and three further lemmas. The proof techniques we use here are simpler and yield stronger results.

Although we don't require it here, the Fundamental Property of Casts has independent interest, and we show in the next section that it follows easily from the results we have already established.

7. Applications

The fact that the translations reflect contextual equivalences gives a powerful proof technique. In this section, we demonstrate the power of this technique by demonstrating two useful results, Lemma 7 from Section 4.1, which justifies the translation $|\cdot|^{\mathsf{CB}}$, and the Fundamental Law of Casts from Siek and Wadler (2010).

7.1 Lemma 7

Lemma 7 from Section 4.1 is used to justify the design of $|\cdot|^{\mathsf{CB}}$, the mapping from $\lambda\mathsf{C}$ back to $\lambda\mathsf{B}$. We repeat the lemma here, with some additional clauses.

Lemma 23 (Equivalences)**.** *The following hold in $\lambda\mathsf{C}$.*

1. $M\langle \texttt{id} \rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M$
2. $M\langle c\,;\, d \rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle c \rangle\langle d \rangle$
3. $M\langle c\,;\, \texttt{id} \rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle c \rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle \texttt{id}\,;\, c \rangle$
4. $M\langle (c \to d)\,;\, (c' \to d') \rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle (c'\,;\, c) \to (d\,;\, d') \rangle$
5. $M\langle c \to d \rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle (c \to \texttt{id})\,;\, (\texttt{id} \to d) \rangle$
6. $M\langle c \to d \rangle \overset{\mathsf{ctx}}{=}_{\mathsf{C}} M\langle (\texttt{id} \to c)\,;\, (d \to \texttt{id}) \rangle$

Proof. Part 1 follows from the definition of contextual equivalence: either $M \longrightarrow_{\mathsf{C}}^* V$ and $M\langle \texttt{id} \rangle \longrightarrow_{\mathsf{C}}^* V$, or $M \longrightarrow \texttt{blame}\ p$ and $M\langle \texttt{id} \rangle \longrightarrow_{\mathsf{C}} \texttt{blame}\ p$, or $M{\uparrow_{\mathsf{C}}}$ and $M\langle \texttt{id} \rangle{\uparrow_{\mathsf{C}}}$. Part 2 follows similarly, and part 3 follows from parts 1 and 2. Part 4 is more interesting. Let $|c|^{\mathsf{CT}} = s$, $|d|^{\mathsf{CT}} = t$, $|c'|^{\mathsf{CT}} = s'$ and $|d'|^{\mathsf{CT}} = t'$. Applying $|\cdot|^{\mathsf{CT}}$ to each side of the equation gives

$$(s \to t) \fatsemi (s' \to t') \overset{\mathsf{ctx}}{=}_{\mathsf{T}} (s' \fatsemi s) \to (t \fatsemi t')$$

which holds immediately from the definition of $\fatsemi$. Then part 4 follows because $|\cdot|^{\mathsf{CT}}$ reflects contextual equivalence, the backward part of Proposition 17. Part 5 follows from

$$c \to d \overset{\mathsf{ctx}}{=}_{\mathsf{C}} (\texttt{id}\,;\, c) \to (\texttt{id}\,;\, d) \overset{\mathsf{ctx}}{=}_{\mathsf{C}} (c \to \texttt{id})\,;\, (\texttt{id} \to d)$$

which follows from part 3 and part 4, respectively. Part 6 is shown similarly to part 5. □

Typically, one might be tempted to prove a result such as Lemma 7 by introducing a custom bisimulation relation for the purpose—indeed, that is the first way we attempted to demonstrate it. Eventually we realised that this was not required, and that we could exploit the technique of showing terms equivalent in $\lambda\mathsf{C}$ by mapping them into $\lambda\mathsf{T}$ and exploiting the backward half of the full abstraction result, Proposition 17. Instead of introducing a new bisimulation relation, all of the "heavy lifting" is done by the already defined bisimulation $\approx$ from Figure 6 and by Proposition 16.

7.2 Fundamental Property of Casts

As a second application of this technique, we show how to establish the Fundamental Property of Casts, Lemma 2 of (Siek and Wadler 2010), which asserts that a single cast is equivalent to a pair of casts through any type greater than the meet of the source and target.

Threesome to mediating type $\boxed{||t|| = C}$

$$
\begin{aligned}
||\mathtt{id}_K|| &= K \\
||\mathtt{id}_{s \to t}|| &= ||s|| \to ||t|| \\
||\mathtt{id}_\star|| &= \star \\
||g\,;G!|| &= ||g|| \\
||G?^p\,;i|| &= ||i|| \\
||\bot^{GpH}|| &= \bot
\end{aligned}
$$

Threesome to threesome without blame $\boxed{|M|^{\mathsf{TW}} = M'}$

$$
\begin{aligned}
&|\mathtt{blame}\ p|^{\mathsf{TW}} = \mathtt{wrong} \\
&|M\langle t\rangle|^{\mathsf{TW}} = |M|^{\mathsf{TW}} : A \xrightarrow{T} B \quad \text{if } t : A \Longrightarrow B \text{ and } ||t|| = T
\end{aligned}
$$

Figure 8. Relating threesomes without blame ($\lambda\mathsf{T}$ and $\lambda\mathsf{W}$)

(Recall that the meet of two types is their greatest lower bound, as defined in Figure 7.)

Our proof technique is to establish that two terms of $\lambda\mathsf{B}$ map to contextually equivalent terms of $\lambda\mathsf{T}$. We first establish one simple lemma.

Lemma 24. *If $A \,\&\, B <:_n C$ then*

$$|A \xrightarrow{p} B|^{\mathsf{BT}} = |A \xrightarrow{p} C|^{\mathsf{BT}} \fatsemi |C \xrightarrow{p} B|^{\mathsf{BT}}$$

Proof. By induction on A, B, and C. □

Corollary 25 (Fundamental Property of Casts)**.** *Let M be a term of $\lambda\mathsf{B}$. If $A \,\&\, B <:_n C$ then*

$$M : A \xrightarrow{p} B \overset{\mathsf{ctx}}{=_{\mathsf{B}}} M : A \xrightarrow{p} C \xrightarrow{p} B$$

Proof. Immediate from Lemma 24 and Propositions 12 and 17. □

In Siek and Wadler (2010) establishing the same result is significantly more difficult: it requires specifying a new form of bisimulation and proving six lemmas.

8. Related Work

This section provides an in-depth comparison to the work of Siek and Wadler (2010), Greenberg (2013), and Garcia (2013), then summarizes other relevant work.

8.1 Relation to Siek and Wadler (2010)

Siek and Wadler (2010) use threesomes of the form

$$A \xrightarrow{P} B$$

where A and B are types as here, and P is a labeled type indicating how blame is allocated if the cast fails. Here is the grammar for labeled types:

$$
\begin{array}{lcl}
l, m & ::= & p \mid \epsilon \\
P, Q & ::= & K^l \mid P \to^l Q \mid \star \mid \bot^{pG^l}
\end{array}
$$

(We have renamed some variables to make it easier to compare our work and theirs; in the original they use l, m where we use p, q and vice versa, S, T where we use A, B, and B where we use K.) The meaning of a labeled type is subtle as it depends on whether the label is present or not. For example, their $\bot^{pG^\epsilon}$ corresponds to our $\bot^{GpH}$, while their $\bot^{pG^q}$ corresponds to our $G?^q\,;\bot^{GpH}$. Note that our system records that the failure is the result of casting from ground type G to ground type H, whereas theirs only records that the cast was from ground type G. Their paper includes a translation $(\!|\cdot|\!)$ from threesomes to coercions.

Here is their composition function, with notation changed slightly to align with our own.

$$
\begin{aligned}
K^l \fatsemi K^m &= K^l \\
(P \to^l P') \fatsemi (Q \to^m Q') &= (Q \fatsemi P) \to^l (P' \fatsemi Q') \\
\star \fatsemi P &= P \\
P \fatsemi \star &= P \\
P^{G^m} \fatsemi Q^{H^p} &= \bot^{pG^m} \quad \text{if } G \neq H \\
\bot^{pG^m} \fatsemi Q &= \bot^{pG^m} \\
P^{G^l} \fatsemi \bot^{pG^m} &= \bot^{pG^l} \\
P^{G^l} \fatsemi \bot^{pH^q} &= \bot^{qG^l} \quad \text{if } G \neq H
\end{aligned}
$$

Here P^{G^l} means that the labeled type P is compatible with the ground type G and the label at the top of P is l. The correctness of these equations is not immediate. For instance, in the last line why do P^{G^l} and $\bot^{pH^q}$ on the left combine to form $\bot^{qG^l}$ on the right? Perhaps the easiest way to validate the equations is to translate to coercions using $(\!|\cdot|\!)$, then check that the left-hand side normalises to the right-hand side. In contrast, our definition of $\fatsemi$ (Figure 5) is easy to read off as correct, directly from the equations of the coercion calculus.

8.2 Relation to Greenberg (2013)

Greenberg (2013) considers calculi CAST, NAIVE, and EFFICIENT, similar to our $\lambda\mathsf{B}$, $\lambda\mathsf{C}$, and $\lambda\mathsf{T}$; unlike our work, he includes refinement types, but omits blame.

EFFICIENT defines a composition operator that serves the same purpose as our $\fatsemi$. The composition operator $c_1 * c_2 \Rightarrow c_3$ composes the right-most primitive coercion of c_1 with the left-most primitive coercion of c_2, then recursively composes the result with what is left of c_1 and c_2. For example, the following is the rule for composing function coercions.

$$
\frac{c_{21} * c_{11} \Rightarrow c_{31} \quad c_{12} * c_{22} \Rightarrow c_{32} \quad c_1 * (c_{31} \to c_{32})\,;c_2 \Rightarrow c}{c_1\,;(c_{11} \to c_{12}) * (c_{21} \to c_{22})\,;c_2 \Rightarrow c}
$$

The recursive definition is not a structural recursion, so proving that composition is total and that it returns a canonical coercion is challenging, requiring four pages. In contrast, because our definition uses structural recursion, validating that it is total requires simply checking that the definition covers all pairs of threesomes that may be composed according to their type, which is straightforward.

8.3 Relation to Garcia (2013)

Garcia (2013) observes that coercions are easier to understand while threesomes are easier to implement, and shows how to derive threesomes from coercions through a series of correctness-preserving transformations. To accomplish this, Garcia (2013) defines supercoercions and gives their meaning in terms of a translation $\mathcal{N}$ to coercions. Here is his translation, with notation changed slightly to align with our own.

$$
\begin{array}{llll}
|\mathtt{id}_\star|^{\mathsf{SC}} & = \mathtt{id}_\star & |G!|^{\mathsf{SC}} & = G! \\
|\mathtt{id}_K|^{\mathsf{SC}} & = \mathtt{id}_K & |G?^p|^{\mathsf{SC}} & = G?^p \\
|\bot^{pG}|^{\mathsf{SC}} & = \bot^{GpH} & |G?^p!|^{\mathsf{SC}} & = G?^p\,;G! \\
|\bot^{pGq}|^{\mathsf{SC}} & = G?^q\,;\bot^{GpH} &&
\end{array}
$$

$$|\ddot{c}_1 \to \ddot{c}_2|^{\mathsf{SC}} = |\ddot{c}_1|^{\mathsf{SC}} \to |\ddot{c}_2|^{\mathsf{SC}}$$
$$|\ddot{c}_1 \to !\ddot{c}_2|^{\mathsf{SC}} = (|\ddot{c}_1|^{\mathsf{SC}} \to |\ddot{c}_2|^{\mathsf{SC}}) \,;\, (\star \to \star)!$$
$$|\ddot{c}_1 ?^p \to \ddot{c}_2|^{\mathsf{SC}} = (\star \to \star)?^p \,;\, (|\ddot{c}_1|^{\mathsf{SC}} \to |\ddot{c}_2|^{\mathsf{SC}})$$
$$|\ddot{c}_1 ?^p \to !\ddot{c}_2|^{\mathsf{SC}} = (\star \to \star)?^p \,;\, (|\ddot{c}_1|^{\mathsf{SC}} \to |\ddot{c}_2|^{\mathsf{SC}}) \,;\, (\star \to \star)!$$

Garcia (2013) derives a recursive composition function for supercoercions but the definition was too large to publish; it handles all pairs of compatible supercoercions and contains sixty equations. In contrast, our definition fits in ten lines.

8.4 Other Relevant Work

Contemporary languages that integrate dynamic and static types include C# (Bierman et al. 2010), Dart (Bracha and Bak 2011), Pyret (Patterson et al. 2014), Racket (Flatt and PLT 2014), TypeScript (Hejlsberg 2012), and VB (Meijer and Drayton 2004).

Tobin-Hochstadt and Felleisen (2006) formalize the interaction between static and dynamic typing at the granularity of modules and prove a precursor to the blame theorem. Matthews and Findler (2007) define an operational semantics for multi-language programs with static (ML) and dynamic (Scheme) components. Gronski et al. (2006) present Sage, a gradually-typed language with refinement types. Dimoulas et al. (2011, 2012) develop criteria for judging blame tracking strategies. Disney et al. (2011) extend contracts with temporal properties. Strickland et al. (2012) study contracts for mutable objects.

Hinze et al. (2006) design an embedded DSL for contracts in Haskell with multi-label blame assignment. Chitil (2012) develops a lazy version of contract combinators for Haskell. Greenberg et al. (2010) study dependent contracts and the translation between latent and manifest systems. Benton (2008) introduces 'undoable' cast operators, to enable a failed cast to report an error at a more convenient location. Swamy et al. (2014) present a secure embedding of the gradually typed TS* language into JavaScript.

Siek et al. (2009) explore design choices for cast checking and blame tracking in the setting of the coercion calculus. Ahmed et al. (2011) extend the blame calculus to include parametric polymorphism. Siek and Garcia (2012) define a space-efficient abstract machine for the gradually-typed lambda calculus based on coercions.

Acknowledgments Thanks to Shayan Najd and Michael Greenberg. Siek is supported by NSF Grant 1360694. Wadler is supported by EPSRC Programme Grant EP/K034413/1.

References

A. Ahmed, R. B. Findler, J. G. Siek, and P. Wadler. Blame for all. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 201–214, 2011.

N. Benton. Undoing dynamic typing (declarative pearl). In J. Garrigue and M. Hermenegildo, editors, *Functional and Logic Programming*, volume 4989 of *Lecture Notes in Computer Science*, pages 224–238. Springer Berlin Heidelberg, 2008.

G. Bierman, E. Meijer, and M. Torgersen. Adding dynamic types to C#. In *European Conference on Object-Oriented Programming*, ECOOP'10. Springer-Verlag, 2010.

G. Bracha and L. Bak. Dart, a new programming language for structured web programming. Presentation at GOTO conference, Oct. 2011.

O. Chitil. Practical typed lazy contracts. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 67–76, New York, NY, USA, 2012. ACM.

C. Dimoulas, R. B. Findler, C. Flanagan, and M. Felleisen. Correct blame for contracts: no more scapegoating. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '11, pages 215–226, New York, NY, USA, 2011. ACM.

C. Dimoulas, S. Tobin-Hochstadt, and M. Felleisen. Complete monitors for behavioral contracts. In *ESOP*, 2012.

T. Disney, C. Flanagan, and J. McCarthy. Temporal higher-order contracts. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, ICFP '11, pages 176–188, New York, NY, USA, 2011. ACM.

R. B. Findler and M. Felleisen. Contracts for higher-order functions. In *ACM International Conference on Functional Programming (ICFP)*, pages 48–59, Oct. 2002.

M. Flatt and PLT. The Racket reference 6.0. Technical report, PLT Inc., 2014. `http://docs.racket-lang.org/reference/index.html`.

R. Garcia. Calculating threesomes, with blame. In *ACM International Conference on Functional Programming (ICFP)*, pages 417–428, 2013.

M. Greenberg. *Manifest Contracts*. PhD thesis, University of Pennsylvania, Nov. 2013.

M. Greenberg, B. C. Pierce, and S. Weirich. Contracts made manifest. In *Principles of Programming Languages (POPL) 2010*, 2010.

J. Gronski, K. Knowles, A. Tomb, S. N. Freund, and C. Flanagan. Sage: Hybrid checking for flexible specifications. In *Scheme and Functional Programming Workshop (Scheme)*, pages 93–104, Sept. 2006.

A. Hejlsberg. Introducing TypeScript. Microsoft Channel 9 Blog, Oct. 2012.

F. Henglein. Dynamic typing: Syntax and proof theory. *Sci. Comput. Programming*, 22(3):197–230, 1994.

D. Herman, A. Tomb, and C. Flanagan. Space-efficient gradual typing. In *Trends in Functional Programming (TFP)*, Apr. 2007.

D. Herman, A. Tomb, and C. Flanagan. Space-efficient gradual typing. *Higher-Order and Symbolic Computation*, 23:167–189, 2010.

R. Hinze, J. Jeuring, and A. Löh. Typed contracts for functional programming. In M. Hagiya and P. Wadler, editors, *Proceedings of the Eighth International Symposium on Functional and Logic Programming (FLOPS 2006)*, volume 3945 of *Lecture Notes in Computer Science*, pages 208–225. Springer Berlin / Heidelberg, Apr. 2006.

J. Matthews and R. B. Findler. Operational semantics for multi-language programs. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 3–10, Jan. 2007.

E. Meijer and P. Drayton. Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages. In *OOPSLA'04 Workshop on Revival of Dynamic Languages*, 2004.

D. Patterson, J. G. Politz, and S. Krishnamurthi. *Pyret Language Reference*. PLT at Brown University, 5.3.6 edition, 2014. `http://www.pyret.org/docs/`.

J. G. Siek and R. Garcia. Interpretations of the gradually-typed lambda calculus. In *Scheme and Functional Programming Workshop*, 2012.

J. G. Siek and W. Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop (Scheme)*, pages 81–92, Sept. 2006.

J. G. Siek and P. Wadler. Threesomes, with and without blame. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 365–376, 2010.

J. G. Siek, R. Garcia, and W. Taha. Exploring the design space of higher-order casts. In *European Symposium on Programming*, ESOP, pages 17–31, Mar. 2009.

T. S. Strickland, S. Tobin-Hochstadt, R. B. Findler, and M. Flatt. Chaperones and impersonators: run-time support for reasonable interposition. In *Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, 2012.

N. Swamy, C. Fournet, A. Rastogi, K. Bhargavan, J. Chen, P.-Y. Strub, and G. Bierman. Gradual typing embedded securely in javascript. In *ACM Conference on Principles of Programming Languages (POPL)*, Jan. 2014.

S. Tobin-Hochstadt and M. Felleisen. Interlanguage migration: From scripts to programs. In *Dynamic Languages Symposium (DLS)*, pages 964–974, Oct. 2006.

P. Wadler and R. B. Findler. Well-typed programs can't be blamed. In *European Symposium on Programming (ESOP)*, pages 1–16, Mar. 2009.