JFP **31**, e20, 56 pages, 2021.

doi:10.1017/S0956796821000101

Blame and coercion: Together again for the first time

JEREMY G. SIEK
Department of Computer Science, Bloomington, Indiana
(*e-mail:* jsiek@indiana.edu)

PETER THIEMANN
Faculty of Engineering, Freiburg, Germany
(*e-mail:* thiemann@informatik.uni-freiburg.de)

PHILIP WADLER
School of Informatics, Edinburgh
(*e-mail:* wadler@inf.ed.ac.uk)

Abstract

C#, Dart, Pyret, Racket, TypeScript, VB: many recent languages integrate dynamic and static types via gradual typing. We systematically develop four calculi for gradual typing and the relations between them, building on and strengthening previous work. The calculi are as follows: λB, based on the blame calculus of Wadler and Findler (2009); λC, inspired by the coercion calculus of Henglein (1994); λS inspired by the space-efficient calculus of Herman, Tomb, and Flanagan (2006); and λT based on the threesome calculus of Siek and Wadler (2010). While λB and λT are little changed from previous work, λC and λS are new. Together, λB, λC, λS, and λT provide a coherent foundation for design, implementation, and optimization of gradual types. We define translations from λB to λC, from λC to λS, and from λS to λT. Much previous work lacked proofs of correctness or had weak correctness criteria; here we demonstrate the strongest correctness criterion one could hope for, that each of the translations is fully abstract. Each of the calculi reinforces the design of the others: λC has a particularly simple definition, and the subtle definition of blame safety for λB is justified by the simple definition of blame safety for λC. Our calculus λS is implementation-ready: the first space-efficient calculus that is both straightforward to implement and easy to understand. We give two applications: first, using full abstraction from λC to λS to establish an equational theory of coercions; and second, using full abstraction from λB to λS to easily establish the Fundamental Property of Casts, which required a custom bisimulation and six lemmas in earlier work.

1 Introduction

Contracts and blame. Findler & Felleisen (2002) introduced two seminal ideas: *higher order contracts* to monitor adherence to a rich dependent type discipline and *blame* to indicate which of the two parties is at fault if the contract is violated. In particular, at higher order, a contract allocates blame to the environment if it supplies an incorrect argument or to the function if it supplies an incorrect result. Blame characterizes correctness: one cannot

guarantee that a contract interposed between typed and untyped code will not be violated, but one can guarantee that if it is violated then blame is allocated to the untyped code, a result first established by Tobin-Hochstadt & Felleisen (2006).

Findler and Felleisen's innovation led to a bloom of others. Siek & Taha (2006) introduced gradual typing; Flanagan (2006) introduced hybrid typing, later implemented in Sage (Gronski et al., 2006); Ou et al. (2004) integrated simple and dependent types. These systems relied crucially on contracts, and all used a similar translation from a source language to an intermediate language of explicit casts. Alas, they ignored blame. Wadler & Findler (2009) restored blame to this intermediate language and formalized it as as the *blame calculus*. They established *blame safety*, a generalization of the correctness criterion for contracts: given a cast between a less-precise and a more-precise type, blame is always allocated to the less-precisely typed side of the cast—"Well-typed programs can't be blamed".

Coercions. Henglein was the first to discuss coercions in the context of dynamic typing. Coercions are built from injections into the dynamic type and projections from the dynamic type and then lifted to pairs, functions, etc. The primary goal of his coercion calculus is a global optimization that eliminates compositions of an injection followed by its accompanying projection (Henglein, 1992). This optimization is formalized in terms of an equational theory (Henglein, 1994). The coercion calculus neither had a concept of blame nor did it receive much attention until Herman et al. (2007, 2010) proposed it as an intermediate language that provides a theory for space efficient casts.

Space-efficient coercions and casts. A naive implementation of contracts (or the blame calculus or the coercion calculus) suffers space leaks. For example, two mutually recursive procedures should run in constant space when the recursive calls are in tail position; but if one of them is statically typed and the other is dynamically typed, the intervening casts break the tail call property, and the program requires space proportional to the number of calls.

Herman et al. (2007, 2010) proposed a solution to this problem based on the coercion calculus of Henglein (1994). Alas, they also ignored blame. Their calculus represents casts as coercions. When two coercions are applied in sequence, they are composed and normalized. The height of the composition of two coercions is bounded by the heights of the two original coercions; the size of a coercion in normalized form is bounded if its height is bounded, ensuring that computation proceeds in bounded space. However, normalizing 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 when implementing an efficient evaluator.

Siek & Wadler (2009, 2010) proposed an alternative solution using casts. At first, they also ignored blame. They observed that any cast factors into a downcast from the source to a mediating type, followed by an upcast from the mediating type to the target—called a *threesome* because the two casts involve three types. Two successive threesomes collapse to a single threesome, where the mediating type is the greatest lower bound of the two original mediating types. In this way, an arbitrarily long sequence of casts collapses to a pair of casts. The height of the greatest lower bound of any two types is bounded by their

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

heights; and the size of a type is bounded if its height is bounded, again ensuring that computation proceeds in bounded space.

Siek & Wadler (2010) then restored blame by decorating the mediating type with labels that indicate how blame is to be allocated, and showed decorated types are in one-to-one correspondence with normalized coercions. A recursive definition computes the meet of the two decorated types (or equivalently the composition of the two corresponding coercions); it is straightforward to calculate, avoiding the associativity problem of coercions.

However, the notation for decorated types is far from transparent. Siek reports that Tanter attempted to implement Gradualtalk with threesomes, but found it too difficult. Wadler reports that while preparing a lecture on threesomes a few years after the paper was published, he required several hours to puzzle out the meaning of his own notation, $\bot^{mGp}$. Eventually, he could only understand it by relating it to the corresponding coercion—a hint that coercions may be clearer than threesomes once blame is involved.

Hence we have two approaches: Herman et al. (2007, 2010) is easy to understand, but hard to compute; Siek & Wadler (2010) is easy to compute, but hard to understand. Garcia (2013) ameliorated this tension by starting with the former and deriving the latter. However, the derivation necessarily contains all the confusing notation of Siek and Wadler while also introducing additional notations of its own, notably, a collection of ten supercoercions. By design, his derived definition of composition matches Siek and Wadler's original and so is no easier to read.

Much previous work lacked proofs of correctness or had weak correctness criteria. Herman et al. (2007, 2010) give no proof relating their calculus to others for gradual typing. Siek & Wadler (2010) establish that a term in the blame calculus converges if and only if its translation into the threesome calculus converges, but they do so only at the top level (Kleene equivalence: roughly, contextual equivalence without the context).

Our approach. We establish new foundations for gradual typing by considering a sequence of calculi and the relations between them: λB, based on the blame calculus of Wadler & Findler (2009); λC, inspired by the coercion calculus of Henglein (1994); λS, inspired by the space-efficient calculus of Herman et al. (2007, 2010); and λT, based on the threesome calculus without blame of Siek & Wadler (2010). While λB and λT are little changed from previous work, λC and λS are new.

Our calculi are not targeted to the programmer, but rather to the implementor. In fact, the low-level calculi λC and λS are equipped with an equational theory that enables space-efficient implementation as well as compile time optimization of coercions.

The two new calculi are based on ideas so simple it is surprising no one thought of them years ago. For λC, the novel insight is to present a computational calculus as close as possible to the original coercion calculus of Henglein (1994). For λS, the novel insight is to restrict coercions to a canonical form and write out the algorithm that composes two canonical coercions to yield a canonical coercion.

Henglein (1994) explored optimization of coercions, but remarkably neither he nor anyone else has written down the obvious reduction rules for evaluating a lambda calculus with coercions, as we have done here with λC. The result is a pleasingly simple calculus, close to correct by construction.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Our translation from λB to λC resembles many in the literature; it compiles casts into coercions. We show that this translation is a lockstep bisimulation, where a single reduction step in λB corresponds to a single reduction step in λC, giving a close correspondence between the two calculi. There are several subtleties in the design of λB, but essentially none in the design of λC, and that the two run in lockstep suggests that both designs are correct.

A key property of the blame calculus is blame safety—"Well-typed programs can't be blamed". Surprisingly, no previous work considers whether translations preserve blame safety. Here, we show that blame safety is preserved by translations between calculi, and as a pleasant consequence that the subtle definition of blame safety for λB is justified by the straightforward definition of blame safety for λC.

Our reverse translation from λC to λB is novel. We observe that a single coercion must translate into a sequence of casts, because a coercion may contain many blame labels but a cast contains only one. The challenge is to show that translating from λC to λB and back again yields a term contextually equivalent to the original. This, together with the bisimulation, establishes the strongest correctness criterion one could hope for, full abstraction: translation from λB to λC preserves and reflects contextual equivalence.

For λS we isolate a novel grammar corresponding to coercions in canonical form. Canonical forms are unique, and in one-to-one correspondence with normal forms. We present a simple recursive function that takes two coercions in canonical form, s and t, and returns their composition in canonical form, $s \fatsemi t$. Validating the correctness of this definition against Henglein's original rules is straightforward. As with threesomes, it avoids the problems of associativity previously attached to using coercions; but because it is based on coercions, it avoids the problems of decoding the meaning of the decorated types attached to threesomes.

Translation from λC to λS is straightforward, but establishing its correctness is the most challenging result in the paper. The difficulty is that λC breaks compositions into simpler components,

$$M\langle c\,;d\rangle \longrightarrow M\langle c\rangle\langle d\rangle,$$

while λS assembles simpler components into compositions,

$$M\langle s\rangle\langle t\rangle \longrightarrow M\langle s \fatsemi t\rangle.$$

(As explained in Sections 3 and 4, c, d range over coercions and s, t over space-efficient coercions, and $M\langle c\rangle$ and $M\langle s\rangle$ denote application to term M of coercions c and s, respectively.) We introduce a relation between terms of λC and λS and show it is a bisimulation. In this case the bisimulation is not lockstep: one step in λC may correspond to many in λS, and vice versa. Siek & Wadler (2010) establish a bisimulation similar to the one here, but our development is simpler because it uses coercions rather than decorated types, and because it uses λC as an intermediate step. Because the mapping of λS back to λC is simply an inclusion, the bisimulation easily establishes full abstraction of the translation from λC to λS.

The λS calculus and its correctness proof plays an important role as a mathematical model of an efficient implementation of gradual typing. Indeed, Kuhlenschmidt et al. (2019) implement a compiler for a language with first-class functions and mutable

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

references based on λS and present empirical evidence that demonstrates how such a cast representation contributes to both the space efficiency and runtime efficiency for a suite of gradually typed benchmarks.

Lastly, we introduce λT, inspired by the threesomes without blame of Siek & Wadler (2010). Although we no longer require the analogy to types and decorated types to represent casts efficiently, we believe it is useful to clarify that a coercion can be characterized by a triple of types when one ignores blame. Translating λS to λT is straightforward, and it is easy to establish a lockstep bisimulation between the two. Whereas the mapping from λB to λC is an injection, the mapping from λS to λT is a bijection, making it easy to extend the bisimulation to a proof of full abstraction.

In this article we do not study threesomes with blame. The reason is that they are isomorphic to λS, so it would be somewhat redundant to do so, and as discussed above, decorated types and their associated blame tracking semantics is more difficult to understand than coercions with blame tracking.

Example. Figure 1 gives an overview of our results by presenting a running example in each of the four calculi. The example involves two mutually recursive functions, *odd* and *even*, which return true if their argument is odd and even, respectively. In each example, casts or coercions are used so that *odd* has type $\texttt{num} \to \texttt{bool}$, meaning it is statically typed and takes a number to a boolean, while *even* has type $\star \to \star$, meaning it is statically known to be a function, but its argument and result are both of dynamic type. Each example uses notations explained in greater detail in Sections 2, 3, and 5, so the reader may wish to return here after reading the relevant sections. To avoid excessive bracketing, we assume that type casts and coercion applications bind weaker than any other operator except lambda abstraction, the scope of which extends as far to the right as possible.

In the blame calculus, λB, function *odd* accepts a number, which is cast to dynamic type before being passed to *even*, and then the result returned is cast from dynamic type to boolean. If no casts were required, then the definitions of *odd* and *even* would be tail recursive and run in constant space. But as shown in the trace of the computation of *odd* 4, the result casts accumulate, requiring space proportional to the number of calls. In traces, we write $\lceil 3 \rceil$ to embed numeric constants into the dynamic type. (As explained in Section 2, $M : A \Longrightarrow^{p} B$ casts a term M of type A to type B, where p is a label used to allocate blame if the cast fails. That section contains complete type and reduction rules for λB.)

In the coercion calculus, λC, the casts have been replaced by coercions. As before, coercions on the results of functions lose tail recursion, and the trace shows the computation of *odd* 4 requires space proportional to the number of calls. (As explained in Section 3, a coercion of the form $G!$ casts a value from ground type G to dynamic type $\star$, while a coercion of the form $G?^{p}$ casts a value from dynamic type $\star$ to base type G, allocating blame to label p if the cast fails, where G ranges over ground types, which are either base types such as numbers num or booleans bool, or the function type $\star \to \star$. That section contains complete type and reduction rules for λC.)

In the space-efficient coercion calculus, λS, the source program is identical to that for λC, save that each coercion is replaced by its canonical form. Any two adjacent coercions are immediately replaced by their composition in canonical form. The height of the composition of two canonical coercions is bounded by the heights of the two original

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Blame calculus (λB)

$$
\begin{array}{rcl}
\mathit{odd} & = & \lambda x : \mathtt{num}.\ \mathtt{if}\ x == 0\ \mathtt{then}\ \mathtt{false}\ \mathtt{else} \\
 & & \quad \mathit{even}\,(x - 1 : \mathtt{num} \overset{p_1}{\Longrightarrow} \star) : \star \overset{p_2}{\Longrightarrow} \mathtt{bool} \\
\mathit{even} & = & \lambda x : \star.\ \mathtt{if}\ (x : \star \overset{p_5}{\Longrightarrow} \mathtt{num}) == 0\ \mathtt{then}\ \mathtt{true}\ \mathtt{else} \\
 & & \quad \mathit{odd}\,((x : \star \overset{p_3}{\Longrightarrow} \mathtt{num}) - 1) : \mathtt{bool} \overset{p_4}{\Longrightarrow} \star
\end{array}
$$

$$
\begin{array}{rl}
 & \mathit{odd}\,4 \\
\longrightarrow & \mathit{even}\,\lceil 3 \rceil : \star \overset{p_2}{\Longrightarrow} \mathtt{bool} \\
\longrightarrow & \mathit{odd}\,2 : \mathtt{bool} \overset{p_4}{\Longrightarrow} \star \overset{p_2}{\Longrightarrow} \mathtt{bool} \\
\longrightarrow & \mathit{even}\,\lceil 1 \rceil : \star \overset{p_2}{\Longrightarrow} \mathtt{bool} \overset{p_4}{\Longrightarrow} \star \overset{p_2}{\Longrightarrow} \mathtt{bool} \\
\longrightarrow & \mathit{odd}\,0 : \mathtt{bool} \overset{p_4}{\Longrightarrow} \star \overset{p_2}{\Longrightarrow} \mathtt{bool} \overset{p_4}{\Longrightarrow} \star \overset{p_2}{\Longrightarrow} \mathtt{bool} \\
\longrightarrow & \mathtt{false}
\end{array}
$$

Coercion calculus (λC)

$$
\begin{array}{rcl}
\mathit{odd} & = & \lambda x : \mathtt{num}.\ \mathtt{if}\ x == 0\ \mathtt{then}\ \mathtt{false}\ \mathtt{else} \\
 & & \quad \mathit{even}\,(x - 1\langle \mathtt{num!} \rangle)\langle \mathtt{bool?}^{p_2} \rangle \\
\mathit{even} & = & \lambda x : \star.\ \mathtt{if}\ x\langle \mathtt{num?}^{p_5} \rangle == 0\ \mathtt{then}\ \mathtt{true}\ \mathtt{else} \\
 & & \quad \mathit{odd}\,(x\langle \mathtt{num?}^{p_3} \rangle - 1)\langle \mathtt{bool!} \rangle
\end{array}
$$

$$
\begin{array}{rl}
 & \mathit{odd}\,4 \\
\longrightarrow & \mathit{even}\,\lceil 3 \rceil\langle \mathtt{bool?}^{p_2} \rangle \\
\longrightarrow & \mathit{odd}\,2\langle \mathtt{bool!} \rangle\langle \mathtt{bool?}^{p_2} \rangle \\
\longrightarrow & \mathit{even}\,\lceil 1 \rceil\langle \mathtt{bool?}^{p_2} \rangle\langle \mathtt{bool!} \rangle\langle \mathtt{bool?}^{p_2} \rangle \\
\longrightarrow & \mathit{odd}\,0\langle \mathtt{bool!} \rangle\langle \mathtt{bool?}^{p_2} \rangle\langle \mathtt{bool!} \rangle\langle \mathtt{bool?}^{p_2} \rangle \\
\longrightarrow & \mathtt{false}
\end{array}
$$

Space-efficient coercion calculus (λS)

$$
\begin{array}{rlrl}
 & & & \mathit{odd}\,4 \\
 & & \longrightarrow & \mathit{even}\,\lceil 3 \rceil\langle \mathtt{bool?}^{p_2} ; \mathtt{id}_{\mathtt{bool}} \rangle \\
\longrightarrow & \mathit{odd}\,2\langle \mathtt{id}_{\mathtt{bool}} ; \mathtt{bool!} \rangle\langle \mathtt{bool?}^{p_2} ; \mathtt{id}_{\mathtt{bool}} \rangle & \longrightarrow & \mathit{odd}\,2\langle \mathtt{id}_{\mathtt{bool}} \rangle \\
\longrightarrow & \mathit{even}\,\lceil 1 \rceil\langle \mathtt{bool?}^{p_2} ; \mathtt{id}_{\mathtt{bool}} \rangle\langle \mathtt{id}_{\mathtt{bool}} \rangle & \longrightarrow & \mathit{even}\,\lceil 1 \rceil\langle \mathtt{bool?}^{p_2} ; \mathtt{id}_{\mathtt{bool}} \rangle \\
\longrightarrow & \mathit{odd}\,0\langle \mathtt{id}_{\mathtt{bool}} ; \mathtt{bool!} \rangle\langle \mathtt{bool?}^{p_2} ; \mathtt{id}_{\mathtt{bool}} \rangle & \longrightarrow & \mathit{odd}\,0\langle \mathtt{id}_{\mathtt{bool}} \rangle \\
 & & \longrightarrow & \mathtt{false}
\end{array}
$$

Threesome calculus without blame (λT)

$$
\begin{array}{rlrl}
 & & & \mathit{odd}\,4 \\
 & & \longrightarrow & \mathit{even}\,\lceil 3 \rceil : \star \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} \\
\longrightarrow & \mathit{odd}\,2 : \mathtt{bool} \overset{\mathtt{bool}}{\Longrightarrow} \star \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} & \longrightarrow & \mathit{odd}\,2 : \mathtt{bool} \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} \\
\longrightarrow & \mathit{even}\,\lceil 1 \rceil : \star \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} & \longrightarrow & \mathit{even}\,\lceil 1 \rceil : \star \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} \\
\longrightarrow & \mathit{odd}\,0 : \mathtt{bool} \overset{\mathtt{bool}}{\Longrightarrow} \star \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} & \longrightarrow & \mathit{odd}\,0 : \mathtt{bool} \overset{\mathtt{bool}}{\Longrightarrow} \mathtt{bool} \\
 & & \longrightarrow & \mathtt{false}
\end{array}
$$

Fig. 1. Examples.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

compositions, and the size of a canonical coercion is bounded by its height. Hence, the trace shows the computation of *odd* 4 now requires only constant space. (As explained in Section 4, the canonical forms of $G!$ and $G?^p$ are $\mathtt{id}_G ; G!$ and $G?^p ; \mathtt{id}_G$, respectively, where $\mathtt{id}_G$ is the identity coercion on base type G, and $c ; d$ denotes the composition of coercions c and d. That section contains complete type and reduction rules for λS.)

In the calculus of threesomes without blame, λT, the source program is identical to that for λB, save that each cast has been replaced by a corresponding threesome cast, where the blame label has been replaced by a mediating type. Any two adjacent threesome casts may be immediately replaced by a single threesome cast, where the source is taken from the first cast, the target from the second cast, and the mediating type by the meet of the two mediating types. The trace shows the computation of *odd* 4 requires only constant space. (As explained in Section 5, the threesome cast corresponding to $M : A \Longrightarrow^p B$ is $M : A \Longrightarrow^T B$, where the mediating type T is chosen equal to the meet $A \,\&\, B$. The blame label p is dropped because this calculus does not allocate blame. That section contains complete type and reduction rules for λT.)

Outline. This paper revises Siek et al. (2015a). The example of the preceding section, a few minor corrections detailed in Section 2, the switch from evaluation contexts to frames with labeled reductions, the equational theory, and all material on λT is new.

Sections 2, 3, 4, and 5 systematically consider λB, λC, λS, and λT. For each calculus we introduce its syntax, type rules, and reduction rules; and we establish type safety and blame safety. The type safety results for λB, λC, and λS have been formalized in Agda (Siek, 2020a).

In Sections 3, 4, and 5, for each calculus, we also consider translations to and from the previous calculus, show the translations preserve type and blame safety, and demonstrate a bisimulation and full abstraction.

In Section 6, we observe that full abstraction often makes it easy to establish equivalences in λB or λC, because equivalent terms in those calculi translate into one and the same term in λS. In particular, we exploit full abstraction between λC and λS to establish an equational theory of coercions in λC. We also exploit full abstraction between λB and λS to establish the Fundamental Theorem of Casts, which required a custom bisimulation and six lemmas in earlier work (Siek & Wadler, 2010).

Section 7 compares with previous work, and includes a survey of how gradual typing is used in practice. Section 8 concludes.

We provide proofs of the more difficult results in the Appendix. In particular, Appendix A gives a proof of Lemma 8 (used in blame safety) and Appendix C provides the proof of Proposition 19 (the bisimulation between λC and λS).

2 Blame calculus

Figure 2 defines the blame calculus, λB. This section reprises results from Wadler & Findler (2009), Siek & Wadler (2010), and Ahmed et al. (2011). Wadler (2015) provides additional motivation and examples.

Blame calculus is based on simply typed lambda calculus, standard constructs of which are shown in gray. Let A, B, C range over types. A type is either a base type ι, a function

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Syntax

$$A, B, C ::= \iota \mid A \to B \mid \star$$
$$G, H ::= \iota \mid \star \to \star$$
$$L, M, N ::= k \mid op(\vec{M}) \mid x \mid \lambda x{:}A.\, N \mid L\, M \mid M : A \overset{p}{\Longrightarrow} B \mid \mathtt{blame}\, p$$
$$V, W ::= k \mid \lambda x{:}A.\, N \mid V : A \to B \overset{p}{\Longrightarrow} A' \to B' \mid V : G \overset{p}{\Longrightarrow} \star$$
$$\mathcal{E} ::= op(\vec{V}, \Box, \vec{M}) \mid \Box\, M \mid V\, \Box \mid \Box : A \overset{p}{\Longrightarrow} B$$

Compatible $\boxed{A \sim B}$

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

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

$$\frac{}{\Gamma \vdash k : \iota} \qquad \frac{\Gamma \vdash \vec{M} : \vec{\iota}}{\Gamma \vdash op(\vec{M}) : \iota}$$
$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x : A.N : A \to B} \qquad \frac{\Gamma \vdash L : A \to B \quad \Gamma \vdash M : A}{\Gamma \vdash L\, M : B}$$
$$\frac{\Gamma \vdash M : A \quad A \sim B}{\Gamma \vdash (M : A \overset{p}{\Longrightarrow} B) : B} \qquad \frac{}{\Gamma \vdash \mathtt{blame}\, p : A}$$

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

$$op(\vec{V}) \longrightarrow [\![op]\!](\vec{V})$$
$$(\lambda x{:}A.\, N)\, V \longrightarrow N[x{:=}V]$$
$$V : \iota \overset{p}{\Longrightarrow} \iota \longrightarrow V$$
$$(V : A \to B \overset{p}{\Longrightarrow} A' \to B')\, W \longrightarrow (V\, (W : A' \overset{\bar{p}}{\Longrightarrow} A)) : B \overset{p}{\Longrightarrow} B'$$
$$V : \star \overset{p}{\Longrightarrow} \star \longrightarrow V$$
$$V : A \overset{p}{\Longrightarrow} \star \longrightarrow V : A \overset{p}{\Longrightarrow} G \overset{p}{\Longrightarrow} \star \qquad \text{if } \mathrm{ug}(A, G)$$
$$V : \star \overset{p}{\Longrightarrow} A \longrightarrow V : \star \overset{p}{\Longrightarrow} G \overset{p}{\Longrightarrow} A \qquad \text{if } \mathrm{ug}(A, G)$$
$$V : G \overset{p}{\Longrightarrow} \star \overset{q}{\Longrightarrow} G \longrightarrow V$$
$$V : G \overset{p}{\Longrightarrow} \star \overset{q}{\Longrightarrow} H \longrightarrow \mathtt{blame}\, q \qquad \text{if } G \neq H$$
$$\frac{M \longrightarrow M'}{\mathcal{E}[M] \longrightarrow \mathcal{E}[M']} \qquad \frac{}{\mathcal{E}[\mathtt{blame}\, p] \longrightarrow \mathtt{blame}\, p}$$

Fig. 2. Blame calculus (λB).

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

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

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

Types A and B are compatible, written $A \sim B$, if either is the dynamic type, if they are both the same base type, or they are both function types with compatible domains and

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

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. We rule out incompatible casts from the beginning because they always fail at runtime. Write $\mathsf{ug}(A, G)$ to indicate that A has unique ground G distinct from A, that is when $A \neq \star$, $A \neq G$, and $A \sim G$.

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 simply typed lambda calculus, plus casts and blame. Each operator op on base types is specified by a total meaning function $[\![op]\!]$ that preserves types: if $op : \vec{\iota} \to \iota$ and $\vec{k} : \vec{\iota}$, then $[\![op]\!](\vec{k}) = k$ with $k : \iota$.

Typing, reduction, and safety judgments are written with subscripts indicating to which calculus they belong, except we omit subscripts in figures to avoid clutter. We write $\Gamma \vdash_{\mathsf{B}} M : A$ to indicate that in type environment Γ term M has type A. Type rules for simply typed lambda calculus are standard and omitted. The type rule for casts is straightforward:

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

If term M has type A and types A and B are compatible then a cast of M from A to B is a term of type B. The cast is decorated with a blame label p. We abbreviate a pair of casts

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

and similarly for sequences of more than two casts. A term $\mathtt{blame}\, p$ has any type.

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 subterm of the form $\mathtt{blame}\, p$, then $A = A'$.

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.

Let V, W range over values. A value is a constant, a lambda abstraction, a cast of a value from function type to function type, or a cast of a value from ground type to dynamic type. Let $\mathcal{E}$ range over single-level evaluation contexts (Myers, 2013), which we call *frames*. They include casts in the obvious way. It is more common to use recursive evaluation contexts (Felleisen, 1987) rather than single-level frames; Section 4 explains why we prefer frames. We write $M \longrightarrow_{\mathsf{B}} N$ to indicate that term M steps to term N. For any reduction relation $\longrightarrow$, we write its reflexive and transitive closure as $\longrightarrow^*$.

The first two reduction rules are standard (and not repeated in subsequent figures). A cast from a base type to itself leaves the value unchanged. 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,

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Embedding dynamically typed λ-calculus $\boxed{\lceil M \rceil}$

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

Fig. 3. Embedding.

corresponding to the fact that function types are contravariant in the domain and covariant in the range (Findler & Felleisen, 2002; Wadler & Findler, 2009). A cast from type $\star$ to itself leaves the value unchanged. Assume $\mathsf{ug}(A, G)$. Then a cast from A to $\star$ factors into a cast from A to G followed by a cast from G to $\star$, and a cast from $\star$ to A factors into a cast from $\star$ to G followed by a cast from G to A. A cast from a ground type G to type $\star$ and back to the same ground type G leaves the value unchanged. 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 & 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.)

Two rules have side conditions $\mathsf{ug}(A, G)$. The condition implies that $G = \star \to \star$, so we could rewrite the rules replacing G by $\star \to \star$. We use the given form because it is more compact, and it adapts if we permit other ground types, such as product $G = \star \times \star$.

The following lemma will prove useful later.

Lemma 2 (Failure)**.** *If $A \neq \star$, $A \sim G$, and $G \neq H$, then*

$$V : A \xRightarrow{p_1} G \xRightarrow{p_2} \star \xRightarrow{p_3} H \xRightarrow{p_4} \star \xRightarrow{p_5} B \longrightarrow^* \mathtt{blame}\, p_3$$

Embedding $\lceil M \rceil$ in Figure 3 takes terms of dynamically typed lambda calculus into the blame calculus. The embedding introduces a fresh label p for each cast. As the subsequent calculi C, S, and T are connected to B by fully abstract translations, the embedding for those calculi is obtained by composing $\lceil \cdot \rceil$ with the respective translation.

The reduction rules are deterministic.

Proposition 3 (Determinism)**.** *If $M \longrightarrow_{\mathsf{B}} N$ and $M \longrightarrow_{\mathsf{B}} N'$ then $N = N'$.*

Type safety is established via preservation and progress.

Proposition 4 (Type safety, Wadler & Findler (2009))**.**

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 term N such that $M \longrightarrow_{\mathsf{B}} N$, or*
   - *(b) there exists a value V such that $M = V$, or*
   - *(c) there exists a label p such that $M = \mathtt{blame}\, p$.*

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

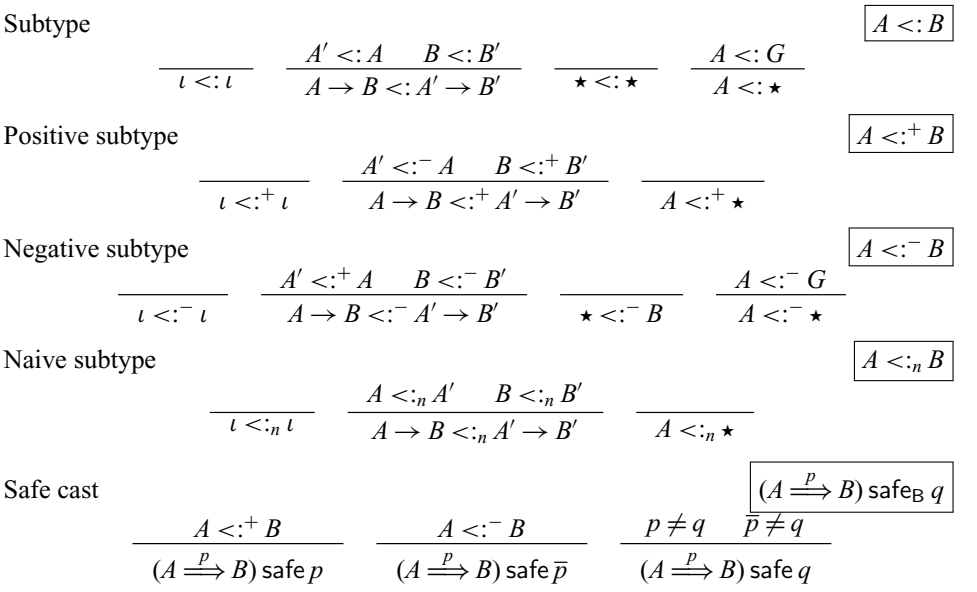


Fig. 4. Subtyping and blame safety.

The same will hold, mutatis mutandis, for λC, λS, and λT.

Type safety does not rule out blame as a result. How to guarantee blame cannot arise in certain circumstances is the subject of the next section.

2.1 Blame safety

Figure 4 presents four different subtyping relations and defines safety for blame calculus. These subtyping relations describe the potential of a cast in a well-typed program to allocate blame. They do not restrict or enhance the typing relation as the type system contains no subsumption rule.

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.

The first three subtyping relations are characterized by *contravariance*. A cast from a base type to itself never yields blame. 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 contravariant in the function domain and covariant in the function range. A cast from ground type to dynamic type never yields blame. A cast to dynamic type never yields positive blame, while a cast from dynamic type never yields negative blame.

Naive subtyping is characterized 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.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

All four relations imply compatibility: if $A <: B$ then $A \sim B$, and similarly for $<:^+$, $<:^-$, and $<:_n$. All four relations are reflexive, and both $<:$ and $<:_n$ are transitive and anti-symmetric.

As a counterexample to transitivity for $<:^-$, observe that $\iota <:^- \star$ and $\star <:^- \star \to \star$ both hold, but $\iota <:^- \star \to \star$ does not hold (it relates incompatible types). Contravariance then gives rise to a counterexample for $<:^+$, since $(\star \to \star) \to A <:^+ \star \to A$ and $\star \to A <:^+ \iota \to A$ both hold for any A, but $(\star \to \star) \to A <:^+ \iota \to A$ does not hold.

We must report a few errors in our previous work. Siek et al. (2015a) omits the rule $\star <: \star$ in its definition of subtype. Wadler & Findler (2009) and Siek et al. (2015a) incorrectly claim that $<:^+$ and $<:^-$ are transitive. Wadler & Findler (2009) incorrectly claims that $<:^-$ does not imply compatibility.

The 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 5 (Tangram, Wadler & Findler (2009))**.**

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 label q,

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

if evaluation of the cast can never allocates blame to 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: $M\ \mathsf{safe}_{\mathsf{B}}\ q$ if every cast in M is safe for q. Blame safety is established via a variant of preservation and progress.

Proposition 6 (Blame safety, Wadler & Findler (2009))**.**

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

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

2.2 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. A context is an expression with a single hole in any position. Write $M\uparrow_{\mathsf{B}}$ if M diverges; defined coinductively by $M\uparrow_{\mathsf{B}}$ if $M \longrightarrow_{\mathsf{B}} N$ and $N\uparrow_{\mathsf{B}}$.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Definition 7 (Contextual equivalence)**.** *Two terms are contextually equivalent,* $M \overset{ctx}{=}_{\mathsf{B}} N$*, if for any context* $\mathcal{C}$*, either*

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

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

3 Coercion calculus

Figure 5 defines the coercion calculus, $\lambda\mathsf{C}$. Our coercions correspond to those of Henglein (1994), except that a coercion from dynamic type to ground type is decorated with a blame label, as done by Siek & Wadler (2010), and we add a coercion $\bot^{GpH}$, similar to `Fail` in Herman et al. (2007, 2010). Our type rules and definition of height are well known; our reduction rules and all results in this section are updated versions from Siek et al. (2015a).

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. Our type rules follow Henglein (1994). The identity coercion at type A is written $\mathtt{id}_A$. 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. If the fail coercion is used at type $\bot^{GpH} : A \to B$, then G is compatible to A but H need not be related to B! Even the case $A = B$ is possible. For a completely formal treatment, the fail coercion would have to be adorned with the source and target types as in the translation of coercions from $\lambda\mathsf{C}$ to $\lambda\mathsf{B}$ in Figure 6.

Terms of the calculus are as before, except that we replace casts by application of a coercion, $M\langle c\rangle$. The typing rule 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 $B = B'$. Conversely, distinct coercions may have the same type: for example, $\mathtt{id}_\star$ and $G\mathtt{?}^p\,;G\mathtt{!}$ both have type $\star \Longrightarrow \star$.

Values and evaluation contexts are as in the blame calculus, with casts replaced by corresponding coercions. We write $M \longrightarrow_{\mathsf{C}} N$ to indicate that term M steps to term N. The identity coercion leaves a value unchanged. A coercion of a function applied to a value

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Syntax

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

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

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

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

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

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

$$V\langle \mathtt{id}_A\rangle \longrightarrow V$$
$$(V\langle c \to d\rangle)\, W \longrightarrow (V\, (W\langle c\rangle))\langle d\rangle$$
$$V\langle G!\rangle\langle G?^p\rangle \longrightarrow V$$
$$V\langle G!\rangle\langle H?^p\rangle \longrightarrow \mathtt{blame}\, p \qquad \text{if } G \neq H$$
$$V\langle c\,;d\rangle \longrightarrow V\langle c\rangle\langle d\rangle$$
$$V\langle \bot^{GpH}\rangle \longrightarrow \mathtt{blame}\, p$$
$$\frac{M \longrightarrow M'}{\mathcal{E}[M] \longrightarrow \mathcal{E}[M']} \qquad \frac{}{\mathcal{E}[\mathtt{blame}\, p] \longrightarrow \mathtt{blame}\, p}$$

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

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

Height $\boxed{||c||}$

$$||\mathtt{id}_A|| = 1 \qquad ||G?^p|| = 1 \qquad ||c \to d|| = \max(||c||, ||d||) + 1$$
$$||\bot^{GpH}|| = 1 \qquad ||G!|| = 1 \qquad ||c\,;d|| = \max(||c||, ||d||)$$

Fig. 5. Coercion calculus (λC).

reduces to a term that coerces on the domain, applies the function, and coerces on the range. If an injection meets a matching projection, the coercion leaves the value unchanged. 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.) Application of a composed coercion applies each of the coercions in turn.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

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

$$|\iota \xRightarrow{p} \iota|^{\mathsf{BC}} = \mathtt{id}_\iota$$
$$|A \to B \xRightarrow{p} A' \to B'|^{\mathsf{BC}} = |A' \xRightarrow{\bar{p}} A|^{\mathsf{BC}} \to |B \xRightarrow{p} B'|^{\mathsf{BC}}$$
$$|\star \xRightarrow{p} \star|^{\mathsf{BC}} = \mathtt{id}_\star$$
$$|G \xRightarrow{p} \star|^{\mathsf{BC}} = G\mathtt{!}$$
$$|\star \xRightarrow{p} G|^{\mathsf{BC}} = G\mathtt{?}^p$$
$$|A \xRightarrow{p} \star|^{\mathsf{BC}} = |A \xRightarrow{p} G|^{\mathsf{BC}} ; G\mathtt{!} \qquad \text{if } \mathrm{ug}(A, G)$$
$$|\star \xRightarrow{p} A|^{\mathsf{BC}} = G\mathtt{?}^p ; |G \xRightarrow{p} A|^{\mathsf{BC}} \qquad \text{if } \mathrm{ug}(A, G)$$

Coercion to blame (λC to λB) $\boxed{|c|^{\mathsf{CB}} = Z}$

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

where if

$$Z = A_1 \xRightarrow{p_1} A_2 \cdots A_m \xRightarrow{p_m} A_{m+1}$$
$$Z' = A_{m+1} \xRightarrow{p_{m+1}} A_{m+2} \cdots A_{m+n} \xRightarrow{p_{m+n}} A_{m+n+1}$$

then

$$Z \to B = A_1 \to B \xRightarrow{p_1} A_2 \to B \cdots A_m \to B \xRightarrow{p_m} A_{m+1} \to B$$
$$B \to Z = B \to A_1 \xRightarrow{p_1} B \to A_2 \cdots B \to A_m \xRightarrow{p_m} B \to A_{m+1}$$
$$\overline{Z} = A_{m+1} \xRightarrow{\overline{p_m}} A_m \cdots A_2 \xRightarrow{\overline{p_1}} A_1$$
$$Z +\!\!+ Z' = A_1 \xRightarrow{p_1} A_2 \cdots A_{m+n} \xRightarrow{p_{m+n}} A_{m+n+1}$$

Fig. 6. Relating λB to λC.

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.

The height of a coercion is as defined by Herman et al. (2007, 2010), and will be used in Section 4.

Determinism, type safety, blame safety, and contextual equivalence for λC are as in λB. Propositions 3, 4, and 6 and Definition 7 apply mutatis mutandis.

3.1 Relating λB *to* λC

The relation between λB and λC is shown in Figure 6. In this section, we let M, N range over terms of λB and M', N' range over terms of λC.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

We write

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

to indicate that the cast on the left translates to the coercion on the right. The translation is designed to ensure there is a lockstep bisimulation between λB and λC. The translation extends to terms in the obvious way, replacing each cast by the corresponding coercion as in

$$|M : A \stackrel{p}{\Longrightarrow} B|^{\mathsf{BC}} = |M|^{\mathsf{BC}}\langle |A \stackrel{p}{\Longrightarrow} B|^{\mathsf{BC}}\rangle$$

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 nonempty sequences of casts. As defined in Figure 6, 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 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 $\bot^{GpH}_{A \Longrightarrow B}$ to indicate that $\bot^{GpH}$ is used as a cast from A to B. This is an informal notation, with the extra information easily recovered by type inference. We choose not to use $\bot^{GpH}_{A \Longrightarrow B}$ as a formal notation throughout, since it would complicate the definition of $\fatsemi$ in Section 4. 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.

We start with some static properties of the translations. The 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 8 (Positive and negative subtyping)**.**

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

The full proof is in Appendix A.

It follows immediately that translation from λB to λC and back preserves type and blame safety.

Proposition 9 (Preservation, λB to λC)**.**

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

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Proposition 10 (Preservation, λC to λB)**.**

1. $\Gamma \vdash_{\mathsf{C}} M' : A$ *iff* $\Gamma \vdash_{\mathsf{B}} |M'|^{\mathsf{CB}} : A$.
2. $M'\ \mathsf{safe}_{\mathsf{C}}\ q$ *iff* $|M'|^{\mathsf{CB}}\ \mathsf{safe}_{\mathsf{B}}\ q$.

(In Siek et al. (2015a), the identity cast is translated as an empty sequence of casts, whereas here we permit only nonempty sequences, and so translate $\mathtt{id}_A$ as $A \Longrightarrow^{\bullet} A$. A consequence of the change is that there Proposition 10 could only use "implies", whereas here we use "iff".)

Thus, the subtle definition of blame safety for B is justified by the simple definition of blame safety for C. The translations from B to C and back are themselves somewhere between the subtlety of the former and the simplicity of the latter.

Turning to operational properties, we observe several contextual equivalences for λC.

Lemma 11 (Equivalences)**.** *The following hold in* λC.

1. $M\langle \mathtt{id}_A \rangle \overset{ctx}{=}_{\mathsf{C}} M$.
2. $M\langle c\,;d \rangle \overset{ctx}{=}_{\mathsf{C}} M\langle c \rangle\langle d \rangle$
3. $M\langle c \to d \rangle \overset{ctx}{=}_{\mathsf{C}} M\langle (c \to \mathtt{id}_B)\,;(\mathtt{id}_{A'} \to d) \rangle$, *where* $c : A' \Longrightarrow A$ *and* $d : B \Longrightarrow B'$.

Parts 1 and 2 follow from the corresponding reduction rules, $M\langle \mathtt{id}_A \rangle \longrightarrow_{\mathsf{C}} M$ and $M\langle c; d \rangle \longrightarrow_{\mathsf{C}} M\langle c \rangle\langle d \rangle$, and part 3 follows from the equational theory established in Section 6.1 and we explain the proof there.

Translating from λC to λB and back again is the identity, up to contextual equivalence.

Lemma 12 (Coercions to blame)**.** *If* M *is a term of* λC *then* $||M|^{\mathsf{CB}}|^{\mathsf{BC}} \overset{ctx}{=}_{\mathsf{C}} M$.

Proof By induction on M, using case analysis on the clauses in the definition of $|\cdot|^{\mathsf{CB}}$. In particular, the clause for $\mathtt{id}$ is justified by part 1 of Lemma 11, 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. The clause for $\bot^{GpH}$ is justified by Lemma 2. □

The translation from λB to λC is a bisimulation. The bisimulation is lockstep: a single step in λB corresponds to a single step in λC, and vice versa.

Proposition 13 (Bisimulation, λB to λC)**.**
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* $M = \mathtt{blame}\,p$.

A proof of Proposition 13 has been mechanized in Agda by Siek (2020b), in the file `EquivLamBLamC.agda`.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

The translation from λB to λC is fully abstract.

Proposition 14 (Fully abstract, λB to λC)**.** *If M and N are terms of* λB *then* $M \overset{ctx}{=}_{\mathsf{B}} N$ *iff* $|M|^{\mathsf{BC}} \overset{ctx}{=}_{\mathsf{C}} |N|^{\mathsf{BC}}$.

Proof For the backward direction, assume that $|M|^{\mathsf{BC}} \overset{ctx}{=}_{\mathsf{C}} |N|^{\mathsf{BC}}$ and let $\mathcal{C}$ be a context of λB.

$$
\begin{aligned}
&\exists V.\, \mathcal{C}[M] \longrightarrow_{\mathsf{B}}^{*} V \\
&\quad \text{iff (bisimulation, Proposition 13, and compositionality)} \\
&\exists V.\, |\mathcal{C}|^{\mathsf{BC}}[|M|^{\mathsf{BC}}] \longrightarrow_{\mathsf{C}}^{*} |V|^{\mathsf{BC}} \\
&\quad \text{iff (assumption)} \\
&\exists W'.\, |\mathcal{C}|^{\mathsf{BC}}[|N|^{\mathsf{BC}}] \longrightarrow_{\mathsf{C}}^{*} W' \\
&\quad \text{iff (bisimulation, Proposition 13, and compositionality)} \\
&\exists W.\, \mathcal{C}[N] \longrightarrow_{\mathsf{B}}^{*} W
\end{aligned}
$$

For the forward direction, assume that $M \overset{ctx}{=}_{\mathsf{B}} N$ and let $\mathcal{C}'$ be a context of λC.

$$
\begin{aligned}
&\exists V'.\, \mathcal{C}'[|M|^{\mathsf{BC}}] \longrightarrow_{\mathsf{C}}^{*} V' \\
&\quad \text{iff (Lemma 12 on context } \mathcal{C}') \\
&\exists V'.\, ||\mathcal{C}'|^{\mathsf{CB}}|^{\mathsf{BC}}[|M|^{\mathsf{BC}}] \longrightarrow_{\mathsf{C}}^{*} V' \\
&\quad \text{iff (compositionality)} \\
&\exists V'.\, ||\mathcal{C}'|^{\mathsf{CB}}[M]|^{\mathsf{BC}} \longrightarrow_{\mathsf{C}}^{*} V' \\
&\quad \text{iff (bisimulation, Proposition 13)} \\
&\exists V.\, |\mathcal{C}'|^{\mathsf{CB}}[M] \longrightarrow_{\mathsf{B}}^{*} V \\
&\quad \text{iff (assumption)} \\
&\exists W.\, |\mathcal{C}'|^{\mathsf{CB}}[N] \longrightarrow_{\mathsf{B}}^{*} W \\
&\quad \text{iff (bisimulation, Proposition 13)} \\
&\exists W'.\, ||\mathcal{C}'|^{\mathsf{CB}}[N]|^{\mathsf{BC}} \longrightarrow_{\mathsf{C}}^{*} W' \\
&\quad \text{iff (compositionality)} \\
&\exists W'.\, ||\mathcal{C}'|^{\mathsf{CB}}|^{\mathsf{BC}}[|N|^{\mathsf{BC}}] \longrightarrow_{\mathsf{C}}^{*} W' \\
&\quad \text{iff (Lemma 12 on context } \mathcal{C}') \\
&\exists W'.\, \mathcal{C}'[|N|^{\mathsf{BC}}] \longrightarrow_{\mathsf{C}}^{*} W'
\end{aligned}
$$

□

4 Space-efficient coercion calculus

Figure 7 defines the space-efficient coercion calculus, λS. Space-efficient coercions correspond to coercions in a canonical form. All the results in this section are updated versions of the results of Siek et al. (2015a).

Blame labels and types are as in λB and λC. There is one space-efficient coercion for each equivalence class of coercions with respect to the equational theory of Section 6.1. Space-efficient coercions follow a specific, three-part grammar, chosen to facilitate the

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Syntax

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

Composition $\boxed{s \fatsemi t = r}$

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

Reduction $\boxed{\longrightarrow_{\mathsf{S}} = \longrightarrow^{\mathcal{E}}_{\mathsf{S}} \cup \longrightarrow^{\mathcal{F}}_{\mathsf{S}}}$ $\boxed{M \longrightarrow^{\mathcal{E}}_{\mathsf{S}} N}$ $\boxed{M \longrightarrow^{\mathcal{F}}_{\mathsf{S}} N}$

$$
\begin{aligned}
op(\vec{V}) &\longrightarrow^{\mathcal{E}} [\![op]\!](\vec{V}) & U\langle \mathtt{id}_\iota \rangle &\longrightarrow^{\mathcal{F}} U \\
(\lambda x{:}A.\, N)\, V &\longrightarrow^{\mathcal{E}} N[x{:=}V] & M\langle s \rangle\langle t \rangle &\longrightarrow^{\mathcal{F}} M\langle s \fatsemi t \rangle \\
(U\langle s \to t \rangle)\, V &\longrightarrow^{\mathcal{E}} (U\,(V\langle s \rangle))\langle t \rangle & U\langle \bot^{GpH} \rangle &\longrightarrow^{\mathcal{F}} \mathtt{blame}\, p
\end{aligned}
$$

$$
\frac{M \longrightarrow M'}{\mathcal{F}[M] \longrightarrow^{\mathcal{E}} \mathcal{F}[M']} \qquad \frac{M \longrightarrow^{\mathcal{E}} M'}{M\langle s \rangle \longrightarrow^{\mathcal{F}} M'\langle s \rangle}
$$

$$
\frac{}{\mathcal{F}[\mathtt{blame}\, p] \longrightarrow^{\mathcal{E}} \mathtt{blame}\, p} \qquad \frac{}{(\mathtt{blame}\, p)\langle s \rangle \longrightarrow^{\mathcal{F}} \mathtt{blame}\, p}
$$

Fig. 7. Space-efficient coercion calculus ($\lambda\mathsf{S}$).

definition of a recursive composition operator, which takes two canonical coercions and computes the canonical coercion corresponding to their composition.

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

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

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

Lemma 15 (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 λC, except that we restrict coercions to space-efficient coercions. The key idea of the dynamics, as in Herman et al. (2007, 2010) and Siek & Wadler (2010), is to combine and normalize adjacent coercions, which ensures space efficiency. Ensuring that adjacent coercions are combined requires we adjust the notion of value and of reduction. Let U range over uncoerced values, that is, values that do not contain a top-level coercion (constants and lambda abstractions). Let V, W range over values, which we constrain to have at most one top-level coercion. Let $\mathcal{E}$ range over evaluation frames, as before, and let $\mathcal{F}$ range over all evaluation frames except for coercions.

If space-efficient coercions s and t are the canonical form of coercions c and d, then $s \fatsemi t$ is the canonical form of $c ; d$. A straightforward induction shows that composition is well defined. The key is to observe that the composition $(i \fatsemi t)$ yields an intermediate coercion for any t and that the composition of two ground coercions $(g \fatsemi h)$ yields a ground coercion. We establish the termination of composition by observing that the sum of the sizes of the arguments gets smaller at each recursive call. The relation between these definitions and the equational theory of Henglein (1994) is discussed in Section 6.1. That discussion will make use of the associativity property of composition.

Lemma 16 (Composition is Associative)**.** *For any* $r : A \Longrightarrow B$, $s : B \Longrightarrow C$, *and* $t : C \Longrightarrow D$, $(r \fatsemi s) \fatsemi t = r \fatsemi (s \fatsemi t)$.

The full proof is in Appendix B.

Height is preserved by composition.

Proposition 17 (Height)**.** $||s \fatsemi t|| \leq \max(||s||, ||t||)$.

A space-efficient coercion contains at most two compositions at its top-level (check the grammar), so a space-efficient coercion bounded in height is also bounded in size.

The reduction rules are designed to ensure that reduction is deterministic and that each reduction has a unique derivation. If a term contains two coercions in succession, then those coercions are composed into one before other reductions occur underneath them. For example, in Figure 1 a space leak in λC is avoided in λS by combining two or more coercions in tail position prior to performing the underlying recursive function application. In contrast, any single coercion evaluates the term under the coercion before the coercion is performed; this order of reduction is necessary to maintaining the correspondence between λS and λC.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

We have three reduction relations,

$$M \longrightarrow_{\mathsf{S}} N \qquad M \longrightarrow_{\mathsf{S}}^{\mathcal{E}} N \qquad M \longrightarrow_{\mathsf{S}}^{\mathcal{F}} N.$$

In the last two of these, superscripts $\mathcal{E}$ and $\mathcal{F}$ are part of the name of the reduction relation, not metavariables ranging over frames. The middle relation is so named because its reductions may occur nested directly inside any frame $\mathcal{E}$, while the last relation is so named because its reduction may only occur nested directly inside a frame $\mathcal{F}$ that does not contain a coercion. The first reduction is simply the union of the other two. In Figure 7 we omit the subscript S on the reduction relations.

There are four congruence rules. The first states that any reduction may be nested in an $\mathcal{F}$ frame; the resulting reduction may take place in any frame, hence it is labeled with $\mathcal{E}$. The second states that an $\mathcal{E}$ reduction may be nested underneath a coercion; the resulting reduction can only take place in an $\mathcal{F}$ frame (else it would reduce under two nested coercions), hence it is labeled with $\mathcal{F}$. The second rule only mentions coercions, and not arbitrary $\mathcal{E}$ frames, in order not to overlap with the preceding rule; this guarantees that each reduction has a unique derivation. The final two congruence rules deal with removing a frame around a blame term, and are justified similarly to the first two rules.

If the rule with left-hand side $M\langle s\rangle\langle t\rangle$ were labeled $\mathcal{E}$ instead of $\mathcal{F}$, then it would not enforce that the outermost string of two casts is the one that is reduced. Similarly, if the rules with left-hand sides $U\langle \mathtt{id}_\iota\rangle$ or $U\langle \bot^{GpH}\rangle$ were labeled with $\mathcal{E}$ in place of $\mathcal{F}$ or had M in place of U, then they would overlap with the rule with left-hand side $M\langle s\rangle\langle t\rangle$.

As an example consider reducing a well-typed term of the form $M\langle \mathtt{id}_\iota\rangle\langle \mathtt{id}_\iota\rangle\langle \mathtt{id}_\iota\rangle$ using relation $\longrightarrow_{\mathsf{S}} = \longrightarrow_{\mathsf{S}}^{\mathcal{E}} \cup \longrightarrow_{\mathsf{S}}^{\mathcal{F}}$.

$$\begin{aligned} M\langle \mathtt{id}_\iota\rangle\langle \mathtt{id}_\iota\rangle\langle \mathtt{id}_\iota\rangle &\longrightarrow_{\mathsf{S}}^{\mathcal{F}} M\langle \mathtt{id}_\iota\rangle\langle \mathtt{id}_\iota \fatsemi \mathtt{id}_\iota\rangle = M\langle \mathtt{id}_\iota\rangle\langle \mathtt{id}_\iota\rangle \\ &\longrightarrow_{\mathsf{S}}^{\mathcal{F}} M\langle \mathtt{id}_\iota \fatsemi \mathtt{id}_\iota\rangle = M\langle \mathtt{id}_\iota\rangle \\ &\longrightarrow_{\mathsf{S}}^{*} U\langle \mathtt{id}_\iota\rangle \\ &\longrightarrow_{\mathsf{S}}^{\mathcal{F}} U \end{aligned}$$

Two initial reduction steps using $\longrightarrow_{\mathsf{S}}^{\mathcal{F}}$ simplify the nested coercion applications to a single one. Next, reduction with $\longrightarrow_{\mathsf{S}}^{\mathcal{E}}$ applies repeatedly to evaluate the term M under the coercion. Finally, reduction $\longrightarrow_{\mathsf{S}}^{\mathcal{F}}$ reduces the identity coercion.

Whereas we use single-level frames, prior work uses recursive evaluation contexts. Herman et al. (2010) use outside-in contexts that provide convenient access to the outermost frame that eases the proof of progress by streamlining the decomposition lemma. Siek et al. (2015a) use inside-out contexts that provide convenient access to the innermost frame, making it easier to constrain the reductions to occur in the correct frame. Here, we obtain the best of both worlds by using frames and by labeling our reduction rules to constrain their immediately enclosing frame. Concurrently, Siek (2020a) mechanizes $\lambda\mathsf{S}$ in Agda and shows that using frames streamlines the proofs of progress and preservation by enabling induction on the term and typing derivation, respectively, as is standard for structural operational semantics (Pierce, 2002; Wadler et al., 2020).

Siek et al. (2015a) also had a slightly different definition of evaluation contexts, which only permitted reduction under coercions in a particular syntactic form (denoted by the

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Coercions to space-efficient (λC to λS) $\boxed{|c|^{\mathsf{CS}} = s}$

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

Bisimulation between λC and λS $\boxed{M \approx_{\mathsf{CS}} M'}$

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

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

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

$$\frac{M \approx M' \qquad \vdash M : A \qquad |\mathtt{id}_A|^{\mathsf{CS}} = s}{M \approx M'\langle s \rangle} \tag{i}$$

$$\frac{M \approx M'\langle s \rangle \qquad |c|^{\mathsf{CS}} = t}{M\langle c \rangle \approx M'\langle s \fatsemi t \rangle} \tag{ii}$$

$$\frac{M \approx L'\langle r \rangle\, M'\langle s \rangle \qquad |d|^{\mathsf{CS}} = t}{M\langle d \rangle \approx (L'\langle r \fatsemi (s \to t) \rangle)\, M'} \tag{iii}$$

Fig. 8. Relating λC to λS.

metavariable f) that did not permit identity coercions. That definition was in error. For example, the following program is stuck.

$$(1 + 2)\langle \mathtt{id}_{\mathtt{num}} \rangle \not\longrightarrow_{\mathsf{S}}$$

Here we fix the problem by permitting reductions underneath arbitrary coercions.

Determinism, type safety, blame safety, and contextual equivalence for λS are as in λB. Propositions 3, 4, and 6 and Definition 7 apply mutatis mutandis.

4.1 Relating λC *to* λS

The translation from λC to λS is shown in Figure 8. In this section, we let M, N range over terms of λC and let M', N' range over terms of λS.

We write

$$|c|^{\mathsf{CS}} = s$$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

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

The inverse translation

$$|s|^{\mathsf{SC}} = c$$

is trivial, since each space-efficient coercion is a coercion.

Translating λC to λS preserves type and blame safety.

Proposition 18 (Preservation, λC to λS)**.**

1. $\Gamma \vdash_{\mathsf{C}} M : A$ *if and only if* $\Gamma \vdash_{\mathsf{S}} |M|^{\mathsf{CS}} : A$.
2. $M\ \mathsf{safe}_{\mathsf{C}}\ q$ *if and only if* $|M|^{\mathsf{CS}}\ \mathsf{safe}_{\mathsf{S}}\ q$.

The same holds trivially for the reverse translation which is the identity.

The dynamics of λC and λS differ in that the former breaks up compositions, while the latter combines them. In Figure 8, we define a bisimulation $\approx$ that relates λC to λS. Rules in gray make the relation a congruence; the rules (i) and (ii) relate a sequence of zero or more coercion applications to a single space-efficient coercion application. The rule (iii) handles the case of applying a function that is wrapped in multiple coercion applications in λC but only wrapped in a single coercion application in λS. The rule relates the two sides during the intermediate reduction steps of λC by mimicking the forward steps on the λS side. For example, consider the sequence of reductions in λC.

$$(V\langle c_1 \to d_1\rangle\langle c_2 \to d_2\rangle)\ W \tag{a}$$
$$\longrightarrow_{\mathsf{C}} (V\langle c_1 \to d_1\rangle\ W\langle c_2\rangle)\langle d_2\rangle \tag{b}$$
$$\longrightarrow_{\mathsf{C}} (V\ (W\langle c_2\rangle\langle c_1\rangle))\langle d_1\rangle\langle d_2\rangle \tag{c}$$

If $V \approx V'$ (where V and V' both have type $A \to B$), $W \approx W'$, $|c_i|^{\mathsf{CS}} = s_i$, and $|d_i|^{\mathsf{CS}} = t_i$, these two reductions relate to a single reduction in λS.

$$(V'\langle (s_1 \to t_1) \fatsemi (s_2 \to t_2)\rangle)\ W' \tag{d}$$
$$\longrightarrow_{\mathsf{S}} (V'\ (W'\langle s_2 \fatsemi s_1\rangle))\langle t_1 \fatsemi t_2\rangle \tag{e}$$

Here (a) $\approx$ (d) via (i) once and (ii) twice. Rule (iii) comes into play to establish (b) $\approx$ (d) (derivation shown below). Here (i) and (ii) are used in both the domain and range, and (iii) is used once to mimic a partial forward step on the λS side. Note that $s_2 = |\mathtt{id}_A|^{\mathsf{CS}} \fatsemi s_2$, which enables the use of (ii) in the range (similar reasoning enables (ii) in the domain).

$$\dfrac{\dfrac{\dfrac{\dfrac{V \approx V'}{V \approx V'\langle |\mathtt{id}_{A\to B}|^{\mathsf{CS}}\rangle}\,(\mathrm{i})}{V\langle c_1 \to d_1\rangle \approx V'\langle s_1 \to t_1\rangle}\,(\mathrm{ii}) \qquad \dfrac{\dfrac{W \approx W'}{W \approx W'\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle}\,(\mathrm{i})}{W\langle c_2\rangle \approx W'\langle s_2\rangle}\,(\mathrm{ii})}{V\langle c_1 \to d_1\rangle\ W\langle c_2\rangle \approx V'\langle s_1 \to t_1\rangle\ W'\langle s_2\rangle} \qquad |d_2|^{\mathsf{CS}} = t_2}{(\mathrm{b}) \approx (\mathrm{d})}\,(\mathrm{iii})$$

We have (c) $\approx$ (e) by two uses of rule (ii) for the d's and t's, two uses of rule (ii) for the c's and s's, and finally one use of rule (i) to relate W and $W'\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle$.

The relation $\approx$ is a bisimulation. It is not lockstep: a single step in λC corresponds to zero or more steps in λS, and vice versa.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Proposition 19 (Bisimulation, λC to λS)**.**
Assume $\vdash_{\mathsf{C}} M : A$ *and* $\vdash_{\mathsf{S}} M' : A$ *and* $M \approx M'$.

1. *If* $M \longrightarrow_{\mathsf{C}} N$ *then* $M' \longrightarrow_{\mathsf{S}}^{*} N'$ *and* $N \approx N'$ *for some* N'.
2. *If* $M' \longrightarrow_{\mathsf{S}} N'$ *then* $M \longrightarrow_{\mathsf{C}}^{*} N$ *and* $N \approx N'$ *for some* N.
3. *If* $M = V$ *then* $M' \longrightarrow_{\mathsf{S}}^{*} V'$ *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* $M' \longrightarrow_{\mathsf{S}}^{*} \mathtt{blame}\, p$.
6. *If* $M' = \mathtt{blame}\, p$ *then* $M = \mathtt{blame}\, p$.

The full proof is in Appendix C. A variant of this bisimulation has also been mechanized in Agda by Lu (2020).

Terms relate to their translations by ≈.

Proposition 20. $M \approx |M|^{\mathsf{CS}}$.

The translation from λC to λS is fully abstract.

Proposition 21 (Fully abstract, λC to λS)**.** *If* M *and* N *are terms of* λC *then* $M \overset{ctx}{=}_{\mathsf{C}} N$ *iff* $|M|^{\mathsf{CS}} \overset{ctx}{=}_{\mathsf{S}} |N|^{\mathsf{CS}}$.

5 Threesomes without blame

Siek & Wadler (2009, 2010) use a different development than the one given here. They first introduce threesomes as a pair of casts,

$$A \overset{T}{\Longrightarrow} B = A \Longrightarrow T \Longrightarrow B$$

from a source type A through a mediating type T to a target type B, where the three types explain the name. This form does not account for blame, which they restore by decorating the mediating cast with blame labels. In contrast, here λC and λS are directly inspired by coercions. We now tie the knot, showing how canonical coercions in λS relate to threesomes when blame is ignored.

We ignore blame here to simplify the presentation. A variant of T with blame would be isomorphic to S. An analogous statement is proved by Siek & Wadler (2010) in Theorem 5.

To account for the case where the source and target type are incompatible, threesomes require introducing the empty type ⊥, which is the least type in the naive ordering. Siek & Wadler (2010) permits every type to include ⊥, which requires extending consistency in an *ad hoc* way. To avoid this issue we follow Siek & Wadler (2009) and only permit ⊥ in the mediating type.

We let R, S, T range over pointed types, which consist of the usual type constructors together with ⊥. Every ordinary type (not using ⊥) is a pointed type, but not conversely. A threesome coercion is written as

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

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

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Pointed 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 types S and T is written $S \,\&\, T$ and defined in Figure 9. It is the greatest lower bound with regard to naive subtyping.

Lemma 22 (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 $R <:_n S \,\&\, T$.*

The reduction rules for threesomes without blame ($\lambda\mathsf{T}$) are in close correspondence to those for space-efficient coercions, save that composition of coercions ($s \fatsemi t$) is replaced by meet of pointed types ($S \,\&\, T$). The β, δ, and congruence rules for $\lambda\mathsf{T}$ are the same as those for $\lambda\mathsf{S}$, so we omit them from Figure 9.

Determinism, type safety, and contextual equivalence for $\lambda\mathsf{T}$ are as in $\lambda\mathsf{B}$. Propositions 3 and 4 and Definition 7 apply mutatis mutandis. Blame safety is not relevant for $\lambda\mathsf{T}$, because there are no blame labels.

5.1 Translation from space-efficient coercions to threesomes

Ignoring blame labels, a space-efficient coercion is determined by its source, target, and mediating types. The mediating type of a space-efficient coercion t is a pointed type $||t||$ as defined in Figure 10.

Lemma 23 (Mediating type)**.** *If $t : A \Longrightarrow B$ then $||t|| <:_n A$ and $||t|| <:_n B$.*

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

Lemma 24 (Composition and meet)**.** *If s and t are space-efficient coercions, then*

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

The above results suggest a simple translation. If t is a space-efficient coercion, $t : A \Longrightarrow B$, define

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

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

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

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

Proposition 25 (Bisimulation, threesomes without blame)**.**
Assume $\vdash_{\mathsf{S}} M : A$ and $\vdash_{\mathsf{T}} M' : A$ and $|M|^{\mathsf{ST}} = M'$.

1. *If $M \longrightarrow_{\mathsf{S}} N$ then $M' \longrightarrow_{\mathsf{T}} N'$ and $|N|^{\mathsf{ST}} = N'$ for some N'.*

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Syntax

$$R, S, T ::= \iota \mid S \to T \mid \star \mid \bot$$
$$L, M, N ::= x \mid k \mid op(\vec{M}) \mid \lambda x{:}A.\, N \mid L\, M \mid M : A \overset{T}{\Longrightarrow} B \mid \mathtt{blame}$$
$$U ::= k \mid \lambda x{:}A.\, N$$
$$V, W ::= U \mid U : A \to B \overset{S \to T}{\Longrightarrow} A' \to B' \mid U : A \overset{T}{\Longrightarrow} \star$$
$$\mathcal{E} ::= \mathcal{F} \mid \Box : A \overset{T}{\Longrightarrow} B$$
$$\mathcal{F} ::= op(\vec{V}, \Box, \vec{M}) \mid \Box\, M \mid V\, \Box$$

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

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

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

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

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

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

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

$$\iota \,\&\, \iota = \iota \qquad (S \to T) \,\&\, (S' \to T') = (S \,\&\, S') \to (T \,\&\, T')$$
$$\star \,\&\, T = T \qquad S \,\&\, T = \bot \quad \text{if } S \# T$$
$$T \,\&\, \star = T$$

Reduction $\boxed{\longrightarrow_{\mathsf{T}} = \longrightarrow^{\mathcal{E}}_{\mathsf{T}} \cup \longrightarrow^{\mathcal{F}}_{\mathsf{T}}}$ $\boxed{M \longrightarrow^{\mathcal{E}}_{\mathsf{T}} N}$ $\boxed{M \longrightarrow^{\mathcal{F}}_{\mathsf{T}} N}$

$$(U : A \to B \overset{S \to T}{\Longrightarrow} A' \to B')\, V \longrightarrow^{\mathcal{E}} (U\, (V : A' \overset{S}{\Longrightarrow} A)) : B \overset{T}{\Longrightarrow} B'$$
$$U : \iota \overset{\iota}{\Longrightarrow} \iota \longrightarrow^{\mathcal{F}} U$$
$$M : A \overset{S}{\Longrightarrow} B \overset{T}{\Longrightarrow} C \longrightarrow^{\mathcal{F}} M : A \overset{S \& T}{\Longrightarrow} C$$
$$U : A \overset{\bot}{\Longrightarrow} B \longrightarrow^{\mathcal{F}} \mathtt{blame}$$

Fig. 9. Threesomes without blame ($\lambda\mathsf{T}$).

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

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

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Space-efficient coercion to mediating type $\boxed{||t|| = T}$

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

Space-efficient coercion to threesome ($\lambda\mathsf{S}$ to $\lambda\mathsf{T}$) $\boxed{|M|^{\mathsf{ST}} = M'}$

$$|\mathtt{blame}\,p|^{\mathsf{ST}} = \mathtt{blame}$$

$$|M\langle t\rangle|^{\mathsf{ST}} = |M|^{\mathsf{ST}} : A \overset{T}{\Longrightarrow} B \qquad \text{if } t : A \Longrightarrow B \text{ and } ||t|| = T$$

Fig. 10. Relating $\lambda\mathsf{S}$ to $\lambda\mathsf{T}$.

Whereas the translation from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ is an injection, which from $\lambda\mathsf{S}$ to $\lambda\mathsf{T}$ is a bijection. Say that a coercion is *label-free* if the only label appearing in it is $\bullet$, and similarly for terms.

Lemma 26 (Bijection, threesomes without blame)**.** *For each label-free space-efficient coercion t there is exactly one threesome $A \overset{T}{\Longrightarrow} B$ such that $t : A \Longrightarrow B$ and $||t|| = T$, and conversely.*

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

Proposition 27 (Fully abstract, threesomes without blame)**.** *If M and N are label-free terms of $\lambda\mathsf{S}$ then $M \overset{ctx}{=}_{\mathsf{S}} N$ iff $|M|^{\mathsf{ST}} \overset{ctx}{=}_{\mathsf{T}} |N|^{\mathsf{ST}}$.*

The development in this section is straightforward. Lemmas 23 and 24 are established by easy inductions, and Propositions 25 and 27 are straightforward. In contrast, the weaker correctness result of Siek & 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 do not 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.

6 Applications

Full abstraction considerably eases some proofs. In this section, we use it to demonstrate an equational theory of coercions similar to that of Henglein (1994) and the Fundamental Law of Casts from Siek & Wadler (2010).

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

$$(c; d); e \simeq c; (d; e) \tag{E1}$$
$$\mathtt{id}_A; c \simeq c \tag{E2}$$
$$c; \mathtt{id}_B \simeq c \tag{E3}$$
$$(c \to d); (c' \to d') \simeq (c'; c) \to (d; d') \tag{E4}$$
$$\mathtt{id}_{A \to B} \simeq \mathtt{id}_A \to \mathtt{id}_B \tag{E5}$$
$$G!; G?^p \simeq \mathtt{id}_G \tag{E6}$$
$$G!; H?^p \simeq \bot^{GpH} \quad \text{if } G \neq H \tag{E7}$$
$$(c \to d); \bot^{GpH} \simeq \bot^{GpH} \quad \text{if } G = \star \to \star \tag{E8}$$
$$\bot^{GpH}; c \simeq \bot^{GpH} \tag{E9}$$

Fig. 11. Equational theory of coercions.

6.1 Equational theory of coercions

Figure 11 presents an equational theory of coercions. We write $c \simeq d$ to indicate that coercions c and d are equal in the theory, and take $\simeq$ to be the reflexive, symmetric, transitive, and congruence closure of the equations in the figure. All of the equations assume that the coercions are well typed. For instance, in (E2) the phrase $\mathtt{id}_A; c$ implies $c : A \Longrightarrow B$ for some B. In (E7) and (E8), the conditions are for clarification only as they are implied by the equations being well typed.

It is straightforward to establish that this theory is sound.

Proposition 28 (Equational theory)**.** *If $c \simeq d$ by the equational theory of Figure 11, then $M\langle c\rangle \overset{ctx}{\equiv}_{\mathsf{C}} M\langle d\rangle$.*

The proposition follows easily by full abstraction of the translation from λC to λS and properties of $⨟$. For instance, (E1) follows because $⨟$ is associative (Lemma 31), (E4) corresponds to the second line in the definition of $⨟$ (Figure 7), and (E5) corresponds to the third line in the translation $|\cdot|^{\mathsf{CS}}$ (Figure 8).

The equational theory is similar to that given by Henglein (1994). Indeed, (E1)–(E5) of our equational theory are the same as Henglein's core theory (his Figure 1), while (E6) corresponds to his ϕ equation (his Figure 2). All our coercions have immediate analogues in his theory except for $\bot^{GpH}$, but we will see how to translate this last one shortly.

Henglein also has an equation ψ (his Figure 2), which does not hold in our theory; it would correspond in our notation to $G?^p; G! \simeq \mathtt{id}_\star$. Henglein defines for any two types a canonical coercion $c : A \Longrightarrow B$ between those types (his Figure 3). Henglein shows that the canonical coercions are exactly those in $\phi\psi$-normal form, and that any two coercions with the same source and target types must be $\phi\psi$ convertible: if $c : A \Longrightarrow B$ and $d : A \Longrightarrow B$ then $\phi\psi \vdash c \simeq d$. In contrast, for our purposes it is vital that the coercions $\mathtt{id}_G : G \Longrightarrow G$ and $\bot^{GpH} : G \Longrightarrow G$ are distinct.

Recall that $\bot^{GpH} : A \Longrightarrow B$ requires that $A \sim G$ and $G \neq H$ but puts no requirement on B. It corresponds in Henglein's notation to $c; G!; H?^p; d$ where $c : A \Longrightarrow G$ and $d : H \Longrightarrow B$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

are the canonical coercions of the appropriate types. That definition yields our (E7) as a special case, but our (E8) and (E9) do not follow from Henglein's equations without ψ.

Part 3 of Lemma 11 from Section 3.1 is instrumental in establishing full abstraction between $\lambda\mathsf{B}$ and $\lambda\mathsf{C}$. Typically, one might be tempted to prove a result such as Part 3 by introducing a custom bisimulation relation—indeed, that is how we first attempted to demonstrate it. Here it follows directly from the equational theory, which in turn follows by full abstraction of the mapping from $\lambda\mathsf{C}$ to $\lambda\mathsf{S}$.

$$\begin{array}{ll} & (c \to d) \\ \simeq & \quad \text{(E2),(E3)} \\ & (\mathtt{id}\,;c) \to (\mathtt{id}\,;d) \\ \simeq & \quad \text{(E4)} \\ & (c \to \mathtt{id})\,;(\mathtt{id} \to d) \end{array}$$

Instead of introducing a custom bisimulation relation, all of the "heavy lifting" is done by bisimulation $\approx$ from Figure 8 and by Proposition 19. Full abstraction from $\lambda\mathsf{C}$ to $\lambda\mathsf{S}$ does not depend of full abstraction from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$, so there is no circularity.

6.2 Fundamental property of casts

As a second application, we show how to establish the Fundamental Property of Casts, Lemma 2 of Siek & Wadler (2010), which asserts that a single cast is contextually equivalent to a pair of casts. We will do so by mapping two terms of $\lambda\mathsf{B}$ to contextually equivalent terms of $\lambda\mathsf{S}$.

Take $|-|^{\mathsf{BS}}$ to be the composition of $|-|^{\mathsf{BC}}$ and $|-|^{\mathsf{CS}}$. We first establish a simple lemma, which follows immediately by case analysis on A, B, and C. The lemma relies on the meet operation $A \,\&\, B = T$ for naive subtyping, which is always defined and may produce a pointed type if A and B are ordinary types.

Lemma 29. *If* $A \,\&\, B <:_n C$ *then*

$$|A \overset{p}{\Longrightarrow} B|^{\mathsf{BS}} = |A \overset{p}{\Longrightarrow} C|^{\mathsf{BS}} \fatsemi |C \overset{p}{\Longrightarrow} B|^{\mathsf{BS}}$$

The fundamental property follows immediately by full abstraction from $\lambda\mathsf{B}$ to $\lambda\mathsf{C}$ and $\lambda\mathsf{C}$ to $\lambda\mathsf{S}$.

Lemma 30 (Fundamental Property of Casts)**.** *Let* M *be a term of* $\lambda\mathsf{B}$. *If* $A \,\&\, B <:_n C$ *then*

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

Siek & Wadler (2010) establish the same result with more difficulty: they require a custom bisimulation and six lemmas.

(Our statement of the fundamental property uses ordinary types, while Siek & Wadler (2010) uses pointed types throughout. Hence, the property proved here is not identical to the one proved there. This is a minor technical difference, not one of substance.)

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

7 Related work

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

7.1 Relation to Siek & Wadler (2010)

Siek & Wadler (2010) use threesomes of the form

$$\langle T \overset{P}{\Longleftarrow} S \rangle\, s$$

where s is a term, S, T are types, and P is a labeled type that indicates how blame is allocated if the cast fails. Here is the grammar for labeled types:

$$p, q ::= l \mid \epsilon$$
$$P, Q ::= B^p \mid P \to^p Q \mid \star \mid \bot^{lGp}$$

Their l, m range over blame labels (our p, q), their p, q range over optional blame labels, their P, Q range over labeled types, their B ranges over base types (our ι), and their G, H range over ground types (our G, H). The meaning of a labeled type is subtle as it depends on whether each label is present or not. For example, their $\bot^{lG\epsilon}$ corresponds to our $\bot^{GpH}$, while their $\bot^{lGm}$ correspond to our $G?^q ; \bot^{GpH}$ (taking their l, m to correspond to our p, q, respectively). Their paper includes a translation $(\!|-|\!)$ from threesomes to coercions.

If our space-efficient coercions s, t correspond to their labeled types P, Q, then our $s \fatsemi t$ corresponds to their $Q \circ P$ (note the reversal!), defined as follows.

$$\begin{aligned}
B^q \circ B^p &= B^p \\
P \circ \star &= P \\
\star \circ P &= P \\
Q^{Hm} \circ P^{Gp} &= \bot^{mGp} && \text{if } G \neq H \\
Q \circ \bot^{mGp} &= \bot^{mGp} \\
\bot^{mGq} \circ P^{Gp} &= \bot^{mGp} \\
\bot^{mHl} \circ P^{Gp} &= \bot^{lGp} && \text{if } G \neq H \\
(P' \to^q Q') \circ (P \to^p Q) &= (P \circ P') \to (Q' \circ Q)
\end{aligned}$$

Here, P^{Gp} means that labeled type P is compatible with ground type G and that p is the topmost optional blame label in P. The correctness of these equations is not immediate. For instance, in the penultimate line why do P^{Gp} and $\bot^{mHl}$ compose to yield $\bot^{lGp}$? Perhaps the easiest way to validate the equations is to translate to coercions using $(\!|-|\!)$, then check that the left-hand side normalizes to the right-hand side. In contrast, our definition of $\fatsemi$ (Figure 7) is easily justified by the equational theory of Henglein (1994).

7.2 Relation to Greenberg (2013)

Greenberg (2013) considers a sequence of calculi CAST, NAIVE, and EFFICIENT, roughly corresponding to our λB, λC, and λS. Unlike us, he includes refinement types, but

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

omits blame; and he formulates correctness in terms of logical relations rather than full abstraction.

His EFFICIENT resembles our λS, in that it defines a composition operator that serves the same purpose as our ⨟. He writes $c_1 * c_2 \Rightarrow c_3$ to indicate that the composition of c_1 and c_2 is equivalent to c_3. The rules to compute $c_1 * c_2$ compose the rightmost primitive coercion of c_1 with the leftmost primitive coercion of c_2, then recursively compose the result with what is left of c_1 and c_2. For example, here is the rule for composing function coercions.

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

His definition is recursive but proving it total is challenging, requiring four pages. In contrast, for our definition totality is straightforward.

7.3 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, he defines supercoercions and gives their meaning in terms of a translation $\mathcal{N}(-)$ to coercions.

$$\begin{aligned}
\mathcal{N}(\iota_P) &= \iota_P \\
\mathcal{N}(\mathtt{Fail}^l) &= \mathtt{Fail}^l \\
\mathcal{N}(\mathtt{Fail}^{l_1 Gl_2}) &= \mathtt{Fail}^{l_1} \circ G?^{l_2} \\
\mathcal{N}(G!) &= G! \\
\mathcal{N}(G?^l) &= G?^l \\
\mathcal{N}(G?^l\,!) &= G! \circ G?^l \\
\mathcal{N}(\ddot{c}_1 \to \ddot{c}_2) &= \mathcal{N}(\ddot{c}_1) \to \mathcal{N}(\ddot{c}_2) \\
\mathcal{N}(\ddot{c}_1\ !\to \ddot{c}_2) &= (\star \to \star)! \circ (\mathcal{N}(\ddot{c}_1) \to \mathcal{N}(\ddot{c}_2)) \\
\mathcal{N}(\ddot{c}_1 \to ?^l\, \ddot{c}_2) &= (\mathcal{N}(\ddot{c}_1) \to \mathcal{N}(\ddot{c}_2)) \circ (\star \to \star)?^l \\
\mathcal{N}(\ddot{c}_1\,!\to ?^l \ddot{c}_2) &= (\star \to \star)! \circ (\mathcal{N}(\ddot{c}_1) \to \mathcal{N}(\ddot{c}_2)) \circ (\star \to \star)?^l
\end{aligned}$$

His l ranges over blame labels (our p, q), his ι is the identity coercion (our id), his P ranges over atomic types (either a base type or the dynamic type), his $\mathtt{Fail}^l$ is a failure coercions (our $\bot^{GpH}$), and his $\ddot{c}$ ranges over supercoercions. Garcia (2013) derives a recursive composition function for supercoercions but the definition was too large to publish as there are sixty pairs of compatible supercoercions. In contrast, our definition fits in ten lines.

7.4 Systems that use gradual typing

Racket (formerly Scheme) supports dynamic and static typing and higher order contracts with blame (Flatt & PLT, 2014). Racket permits contracts to be written directly. Typed Racket inserts contracts that allocate blame when dynamically typed code fails to conform

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

to the static types declared for it Tobin-Hochstadt & Felleisen (2008). Racket is the origin, via Findler & Felleisen (2002), of the rule for casting functions in λB (the fourth reduction rule in Figure 2). A few years after the conference version of this article (Siek et al., 2015a), collapsible contracts were added to Racket using a merge operator (Feltey et al., 2018) that plays a similar role to the composition operator of Figure 7.

Pyret has limited support for gradual typing (Patterson et al., 2014). Pyret checks that a first-order value (such as integer) conforms to its declaration, but only checks that a higher order value is a function, not that it conforms to its declared parameter and result types. Pyret does not implement any equivalent of the rule for casting functions in λB.

Dart provides support for gradual typing with implicit casts to and from type `dynamic` (Bracha & Bak, 2011; ECMA, 2014). Dart does not provide full static type checking; its type checker aims to warn of likely errors rather than to ensure lack of failures. In checked mode, Dart performs a test at every place that a value can be assigned to a variable and raises an exception if the value's type is not a subtype of the variable's declared type. Dart does not implement any equivalent of the rule for casting functions in λB.

C# type `dynamic` and VB type `Object` play a role similar to our type ⋆, with the compiler introducing first-order casts as needed (Bierman et al., 2010; Feigenbaum, 2008). These languages do not have higher order structural types, only nominal types, so the programmer must manually construct explicit wrappers to accomplish what would amount to a higher order cast. C# and VB do not implement any equivalent of the rule for casting functions in λB.

TypeScript provides `interface` declarations that allow users to specify types for an imported JavaScript module or library (Hejlsberg, 2012). The DefinitelyTyped repository contains over 150 such declarations for a variety of popular JavaScript libraries (Yankov, 2013). TypeScript is not concerned with type soundness, which it does not provide (Bierman et al., 2014), but instead exploits types to provide better prompting in Visual Studio, for instance to to populate a pulldown menu with well-typed methods that might be invoked at a given point. The information supplied by `interface` declarations is taken on faith; failures to conform to the declaration are not reported. Typescript does not implement any equivalent of the rule for casting functions in λB.

Several systems explore how to modify TypeScript to restore various forms of type safety.

Safe TypeScript is a refinement of TypeScript that guarantees type safety by adding run-time type information (RTTI) to values of dynamic type `any` (Rastogi et al., 2015). It introduces the notion of erased types that cannot be coerced to `any`. Erased types are used to communicate with external libraries that are unaware of RTTI. Furthermore, subtyping of function types is restricted to never manipulate RTTI, avoiding the need for wrappers that may change the object identity. Safe Typescript does not implement any equivalent of the rule for casting functions in λB.

StrongScript (Richards et al., 2015) extends TypeScript's optional types with concrete types. A concrete type is a (nominal) class type, which is statically checked and which is protected by compiler-generated casts against its less strictly typed context. The main goals of this work are compatibility with TypeScript and enabling the generation of efficient code for concretely typed parts of a program. Blame tracking is an optional feature that may be

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

disabled to avoid runtime overhead. StrongScript relies upon an equivalent of the rule for casting functions in λB.

Microsoft has funded Wadler and a PhD student, Jack Williams, to build a tool, TypeScript TNG that uses blame calculus to generate wrappers from TypeScript `interface` declarations. The wrappers monitor interactions between a library and a client, and if a failure occurs then blame indicates whether it is the library or the client that has failed to conform to the declared types. TypeScript TNG relies upon an equivalent of the rule for casting functions in λB, but goes beyond our work in supporting union types (Williams et al., 2017, 2018).

7.5 Other relevant work

Abadi et al. (1991) study an early notion of type `Dynamic`. Floyd (1967) and Hoare (1969) introduce reasoning about programs with pre- and post-conditions and Meyer (1988) popularizes checking them at runtime under the name *contracts*. Findler & Felleisen (2002) introduce higher order contracts for functional languages.

Tobin-Hochstadt & Felleisen (2006) formalize the interaction between static and dynamic typing at the granularity of modules and prove a precursor to blame safety. Matthews & 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. Thiemann (2014) takes first steps towards gradual typing for session types.

Hinze et al. (2006) design an embedded DSL for contracts with blame assignment in Haskell. Chitil (2012) develops a lazy version of contracts for Haskell. Greenberg et al. (2010) study dependent contracts and the translation between latent and manifest systems. Blume & McAllester (2006) develop a sound and complete semantics of contracts based on a quotient model and raise a question concerning the meaning of the `any` contract. Findler & Blume (2006) provide an answer by interpreting contracts as pairs of projections. 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 language TS⋆ 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 and Ahmed et al. (2017) prove that it satisfies a notion of parametricity. Siek & Garcia (2012) define a space-efficient abstract machine for the gradually typed lambda calculus based on coercions. Wadler (2015) surveys work on the blame calculus. Siek et al. (2015b) propose the *gradual guarantee* as a new criteria for gradual typing, characterizing how changes in the precision of type annotations may change a program's static and dynamic semantics. Toro et al. (2019) show that parametricity is in tension with the gradual guarantee. They construct a language that satisfies the former but not the later. New et al. (2019) resolve the tension by departing from System F as the statically typed reference language and instead use a new language with explicit primitives for runtime sealing.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Vitousek et al. (2014, 2017) propose an alternative *transient* semantics for casts that sidesteps the efficiency problems regarding higher order casts with pervasive first-order runtime checks. Greenman & Felleisen (2018) report promising performance results for the transient semantics in the context of Typed Racket and Vitousek et al. (2019) do the same for Reticulated Python. There are interesting tradeoffs between the transient semantics and the traditional semantics studied in this article. The traditional semantics provides stronger type soundness guarantees and achieves low overhead in statically typed code (Kuhlenschmidt et al., 2019), whereas the transient semantics is easier to implement and achieves good performance on programs with a mixture of static and dynamic typing (Greenman & Felleisen, 2018; Vitousek et al., 2019).

While the runtime compression of coercions is necessary to provide ironclad space efficiency guarantees, it is beneficial to also optimize coercions at compile time. In the literature, this problem was studied in the context of compiling dynamically typed languages such as Scheme (Henglein, 1992; Henglein & Rehof, 1995) and led to the *soft typing* line of research on type inference (Wright & Cartwright, 1997; Aiken et al., 1994; Flanagan et al., 2002). More recently, Rastogi et al. (2012) adapted constraint-based inference to gradually typed languages. Vitousek et al. (2019) investigate the impact of static optimization on languages that use the transient semantics. Moy et al. (2021) apply symbolic execution to optimize contracts and demonstrate significant performance improvements in Typed Racket.

8 Conclusion

Findler & Felleisen (2002) introduced higher order contracts, setting up a foundation for gradual typing; but they observed a problem with space efficiency. Herman et al. (2007, 2010) restored space efficiency; but required an evaluator to reassociate parentheses. Siek & Wadler (2010) gave a recursive definition of composition that is easy to compute; but the correctness of their definition is not transparent. Here we provide composition that is easy to compute and transparent. At last, we are in a position to implement space-efficient contracts and test them in practice.

When Siek & Wadler (2010) was published, we thought we had discovered a solution that was easy to implement and easy to understand. Only later did we realize that it was not quite so easy as we thought! We believe that the presentation here provides a highly accessible foundation for future work on advanced topics. For us, the lesson is clear: no matter how simple your theory, strive to make it simpler still!

Acknowledgments

The authors would like to thank Jonathan Coates and Ben Sheffield, student at Edinburgh, who formalized some of these results and uncovered some incorrect claims (reported in Section 2.1). Thanks to Shayan Najd, Michael Greenberg, the PLDI referees, and the students of TSPL for comments. Siek acknowledges NSF Grants 1360694, 1518844, and 1763922. Wadler acknowledges EPSRC Programme Grant EP/K034413/1 and a Microsoft Research PhD Scholarship.

Conflicts of interest

The first author is employed at Indiana University, the second author is employed at Universität Freiburg, and the third author is employed at University of Edinburgh.

References

Abadi, M., Cardelli, L., Pierce, B. & Plotkin, G. (1991) Dynamic typing in a statically typed language. *ACM Trans. Prog. Lang. Syst.* **13**(2), 237–268.

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

Ahmed, A., Jamner, D., Siek, J. G. & Wadler, P. (2017) Theorems for free for free: Parametricity, with and without types. In *International Conference on Functional Programming*, ICFP.

Aiken, A., Wimmers, E. L. & Lakshman, T. K. (1994) Soft typing with conditional types. In *POPL '94: Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, NY, USA: ACM Press, pp. 163–173.

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

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

Bierman, G. M., Abadi, M. & Torgersen, M. (2014) Understanding TypeScript. In *European Conference on Object-Oriented Programming (ECOOP)*, pp. 257–281.

Blume, M. & McAllester, D. (2006) Sound and complete models of contracts. *J. Funct. Program.*, 16(4&5):375–414, 2006.

Bracha, G. & Bak, L. (2011) Dart, a new programming language for structured web programming. In *Presentation at GOTO Conference*.

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

Dimoulas, C., Findler, R. B., Flanagan, C. & Felleisen, M. (2011) Correct blame for contracts: no more scapegoating. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 2011, New York, NY, USA: ACM, pp. 215–226.

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

Disney, T., Flanagan, C. & McCarthy, J. (2011) Temporal higher-order contracts. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP 2011, New York, NY, USA: ACM, pp. 176–188.

ECMA. *Dart Programming Language Specification*, 2nd edition, December 2014.

Feigenbaum, L. (2008) *Walkthrough: Dynamic programming in Visual Basic 10.0 and C# 4.0*. Available at: http://blogs.msdn.com/b/vbteam/archive/2008/12/17/walkthrough-dynamic-programming-in-visual-basic-10-0-and-c-4-0-lisa-feigenbaum.aspx

Felleisen, M. (1987) *The Calculi of Lambda-v-CS Conversion: A Syntactic Theory of Control and State in Impereative Higher-Order Programming Languages*. PhD thesis, Indiana University.

Feltey, D., Greenman, B., Scholliers, C., Findler, R. B. & St-Amour, V. (2018) Collapsible contracts: Fixing a pathology of gradual typing. *Proc. ACM Program. Lang.* **2**(OOPSLA), 133:1–133:27.

Findler, R. & Blume, M. (2006) Contracts as pairs of projections. In *Functional and Logic Programming (FLOPS)*.

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

Flanagan, C. (2006) Hybrid type checking. In *Principles of Programming Languages (POPL)*.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Flanagan, C., Leino, K. R. M., Lillibridge, M., Nelson, G., Saxe, J. B. & Stata, R. (2002) Extended static checking for Java. In *PLDI 2002: Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, New York, NY, USA: ACM Press, pp. 234–245.

Flatt, M. & PLT (2014) *The Racket Reference 6.0*. Technical report, PLT. Available at: `http://docs.racket-lang.org/reference/index.html`

Floyd, R. W. (1967) Assigning meanings to programs. In *Symposium in Applied Mathematics*, vol. 19, pp. 19–32.

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

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

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

Greenman, B. & Felleisen, M. (2018) A spectrum of type soundness and performance. *Proc. ACM Program. Lang.* **2**(ICFP), 71:1–71:32.

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

Hejlsberg, A. (2012) Introducing TypeScript. *Microsoft Channel 9 Blog*.

Henglein, F. (1992) Global tagging optimization by type inference. In *LFP 1992: Proceedings of the 1992 ACM Conference on LISP and Functional Programming*, New York, NY, USA: ACM Press, pp. 205–215.

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

Henglein, F. & Rehof, J. (1995) Safe polymorphic type inference for a dynamically typed language: translating Scheme to ML. In *Functional Programming Languages and Computer Architecture (FPCA)*.

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

Herman, D., Tomb, A. & Flanagan, C. (2010) Space-efficient gradual typing. *Higher-Order Symb. Comput.* **23**, 167–189.

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

Hoare, C. A. R. (1969) An axiomatic basis for computer programming. *Commun. ACM* **12**(10), 576–580.

Kuhlenschmidt, A., Almahallawi, D. & Siek, J. G. (2019) Toward efficient gradual typing for structural types via coercions. In *Conference on Programming Language Design and Implementation*, PLDI. ACM.

Lu, K.-C. (2020) *Equivalence of Cast Representations in Gradual Typing.* Master's thesis, Indiana University, April 2020.

Matthews, J. & Findler, R. B. (2007) Operational semantics for multi-language programs. In *Principles of Programming Languages (POPL)*, pp. 3–10.

Meyer, B. (1988) *Object-Oriented Software Construction*. Prentice Hall.

Moy, C., Nguyundefinedn, P. C., Tobin-Hochstadt, S. & Van Horn, D. (2021) Corpse reviver: Sound and efficient gradual typing via contract verification. *Proc. ACM Program. Lang.* **5**(POPL).

Myers, A. (2013) Evaluation contexts, semantics by translation. *CS 6110 Lecture 8*.

New, M. S., Jamner, D. & Ahmed, A. (2019) Graduality and parametricity: Together again for the first time. *Proc. ACM Program. Lang.* **4**(POPL).

Ou, X., Tan, G., Mandelbaum, Y. & Walker, D. (2004) Dynamic typing with dependent types. In *IFIP International Conference on Theoretical Computer Science*, pp. 437–450.

Patterson, D., Politz, J. G. & Krishnamurthi, S. (2014) *Pyret Language Reference*. PLT, Brown University, 5.3.6 edition, 2014.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

http://www.pyret.org/docs/.
Pierce, B. (2002) *Types and Programming Languages*. MIT Press.
Rastogi, A., Chaudhuri, A. & Hosmer, B. (2012) The ins and outs of gradual type inference. In *Symposium on Principles of Programming Languages*, POPL, pp. 481–494.
Rastogi, A., Swamy, N., Fournet, C., Bierman, G. M. & Vekris, P. (2015) Safe & efficient gradual typing for TypeScript. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15–17, 2015*, Rajamani, S.K. & Walker, D. (eds), pp. 167–180. ACM.
Richards, G., Nardelli, F. Z. & Vitek, J. (2015) Concrete types for TypeScript. In *European Conference on Object-Oriented Programming*, ECOOP 2015. Springer-Verlag.
Siek, J. (2020a) *Toward a Mechanized Compendium of Gradual Typing*. Available at: https://arxiv.org/abs/2001.11560
Siek, J., Thiemann, P. & Wadler, P. (2015a) Blame and coercion: Together again for the first time. In *Programming Language Design and Implementation (PLDI)*, pp. 425–435.
Siek, J. G. (2020b) Gradual typing in Agda repository. Available at: https://github.com/jsiek/gradual-typing-in-agda
Siek, J. G. & Garcia, R. (2012) Interpretations of the gradually-typed lambda calculus. In *Scheme and Functional Programming Workshop*.
Siek, J. G. & Taha, W. (2006) Gradual typing for functional languages. In *Scheme and Functional Programming Workshop (Scheme)*, pp. 81–92.
Siek, J. G. & Wadler, P. (2006) Threesomes, with and without blame. In *Workshop on Script-to-Program Evolution (STOP)*, pp. 34–46.
Siek, J. G. & Wadler, P. (2010) Threesomes, with and without blame. In *Principles of Programming Languages (POPL)*, pp. 365–376.
Siek, J. G., Garcia, R. & Taha, W. (2009) Exploring the design space of higher-order casts. In *European Symposium on Programming*, *ESOP*, pp. 17–31.
Siek, J. G, Vitousek, M. M., Cimini, M. & Boyland, J. T. (2015b) Refined criteria for gradual typing. In *Summit on Advances in Programming Languages (SNAPL)*.
Strickland, T. S., Tobin-Hochstadt, S., Findler, R. B. & Flatt, M. (2012) Chaperones and impersonators: run-time support for reasonable interposition. In *Conference on Object Oriented Programming Systems Languages and Applications*, *OOPSLA 2012*.
Swamy, N., Fournet, C., Rastogi, A., Bhargavan, K., Chen, J., Strub, P.-Y. & Bierman, G. (2014) Gradual typing embedded securely in Javascript. In *ACM Conference on Principles of Programming Languages (POPL)*.
Thiemann, P. (2014) Session types with gradual typing. In *Trustworthy Global Computing – 9th International Symposium, TGC 2014, Rome, Italy, September 5–6, 2014. Revised Selected Papers*, Maffei, M. & Tuosto, E. (eds), vol. 8902. Springer, pp. 144–158.
Tobin-Hochstadt, S. & Felleisen, M. (2006) Interlanguage migration: From scripts to programs. In *Dynamic Languages Symposium (DLS)*, pp. 964–974.
Tobin-Hochstadt, S. & Felleisen, M. (2008) The design and implementation of typed scheme. In *Principles of Programming Languages (POPL)*, pp. 395–406. doi: 10.1145/1328438.1328486.
Toro, M., Labrada, E. & Tanter, E. (2019) Gradual parametricity, revisited. *Proc. ACM Program. Lang.* **3**(POPL), 17:1–17:30.
Vitousek, M., Swords, C. & Siek, J. G. (2017) Big types in little runtime. In *Symposium on Principles of Programming Languages*, *POPL*.
Vitousek, M. M., Siek, J. G., Kent, A. & Baker, J. (2014) Design and evaluation of gradual typing for Python. In *Dynamic Languages Symposium*.
Vitousek, M. M., Siek, J. G. & Chaudhuri, A. (2019) Optimizing and evaluating transient gradual typing. In *Proceedings of the 15th ACM SIGPLAN International Symposium on Dynamic Languages*, *DLS 2019*, New York, NY, USA: Association for Computing Machinery, pp. 28–41.
Wadler, P. (2015) A complement to blame. In *Summit on Advances in Programming Languages (SNAPL)*.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

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

Wadler, P., Kokke, W. & Siek, J. G. (2020) *Programming Language Foundations in Agda*.

Williams, J., Morris, J. G., Wadler, P. & Zalewski, J. (2017) Mixed messages: Measuring conformance and non-interference in TypeScript. In *European Conference on Object-Oriented Programming (ECOOP)*, Müller, P. (ed), vol. 74. *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 28:1–28:29. Available at: http://drops.dagstuhl.de/opus/volltexte/2017/7264

Williams, J., Morris, J. G. & Wadler, P. (2018) The root cause of blame: Contracts for intersection and union types. *Proc. ACM Program. Lang.* **2**(OOPSLA). doi: 10.1145/3276504.

Wright, A. K. & Cartwright, R. (1997) A practical soft typing system for Scheme. *ACM Trans. Prog. Lang. Syst.* **19**(1). Available at: http://portal.acm.org/citation.cfm?id=239912.239917

Yankov, B. (2013) *Definitely Typed Repository*. Available at: https://github.com/borisyankov/DefinitelyTyped.

A Positive and negative subtyping

Lemma 8 (Positive and negative subtyping)**.**

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

Proof $A <:^+ B$ implies $|A \overset{p}{\Longrightarrow} B|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ p$ and $A <:^- B$ implies $|A \overset{p}{\Longrightarrow} B|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$ is proved by mutual induction on the definition of $|A \overset{p}{\Longrightarrow} B|^{\mathsf{BC}}$.

Cases for positive subtyping:

Case $|\iota \overset{p}{\Longrightarrow} \iota|^{\mathsf{BC}} = \mathtt{id}_\iota$ satisfies $\iota <:^+ \iota$ and $\mathtt{id}_\iota\ \mathsf{safe}_{\mathsf{C}}\ p$.

Case $|A \to B \overset{p}{\Longrightarrow} A' \to B'|^{\mathsf{BC}} = |A' \overset{\overline{p}}{\Longrightarrow} A|^{\mathsf{BC}} \to |B \overset{p}{\Longrightarrow} B'|^{\mathsf{BC}}$. From the assumption $A \to B <:^+ A' \to B'$, we obtain $A' <:^- A$ and $B <:^+ B'$. By induction, we get that $|A' \overset{\overline{p}}{\Longrightarrow} A|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ \overline{\overline{p}}$ and $|B \overset{p}{\Longrightarrow} B'|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ p$, which proves the claim.

Case $|\star \overset{p}{\Longrightarrow} \star|^{\mathsf{BC}} = \mathtt{id}_\star$ satisfies $\star <:^+ \star$ and $\mathtt{id}_\star\ \mathsf{safe}_{\mathsf{C}}\ p$.

Case $|G \overset{p}{\Longrightarrow} \star|^{\mathsf{BC}} = G!$. Immediate because $G <:^+ \star$.

Case $|A \overset{p}{\Longrightarrow} \star|^{\mathsf{BC}} = |A \overset{p}{\Longrightarrow} G|^{\mathsf{BC}}\,; G!$ where $A \neq \star$, $A \neq G$, and $A \sim G$. Hence, it must be that $G = \star \to \star$ and $A = A' \to B'$ so that $|A \overset{p}{\Longrightarrow} G|^{\mathsf{BC}} = |A' \to B' \overset{p}{\Longrightarrow} \star \to \star|^{\mathsf{BC}} = |\star \overset{\overline{p}}{\Longrightarrow} A'|^{\mathsf{BC}} \to |B' \overset{p}{\Longrightarrow} \star|^{\mathsf{BC}}$. Since $\star <:^- A'$ and $B' <:^+ \star$, the result holds by induction.

Case $|\star \overset{p}{\Longrightarrow} G|^{\mathsf{BC}}$. Not applicable because $\star \not<:^+ G$.

Case $|\star \overset{p}{\Longrightarrow} A|^{\mathsf{BC}}$ where $A \neq \star$, $A \neq G$, and $A \sim G$. Not applicable because $\star \not<:^+ A$.

Cases for negative subtyping:

Case $|\iota \overset{p}{\Longrightarrow} \iota|^{\mathsf{BC}} = \mathtt{id}_\iota$ satisfies $\iota <:^- \iota$ and $\mathtt{id}_\iota\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$.

Case $|A \to B \overset{p}{\Longrightarrow} A' \to B'|^{\mathsf{BC}} = |A' \overset{\overline{p}}{\Longrightarrow} A|^{\mathsf{BC}} \to |B \overset{p}{\Longrightarrow} B'|^{\mathsf{BC}}$. From the assumption $A \to B <:^- A' \to B'$, we obtain $A' <:^+ A$ and $B <:^- B'$. By induction, we get that $|A' \overset{\overline{p}}{\Longrightarrow} A|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$ and $|B \overset{p}{\Longrightarrow} B'|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$, which proves the claim.

Case $|\star \overset{p}{\Longrightarrow} \star|^{\mathsf{BC}} = \mathtt{id}_\star$ satisfies $\star <:^- \star$ and $\mathtt{id}_\star\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$.

Case $|G \overset{p}{\Longrightarrow} \star|^{\mathsf{BC}} = G!$. Immediate because $G <:^- \star$.

Case $|A \overset{p}{\Longrightarrow} \star|^{\mathsf{BC}} = |A \overset{p}{\Longrightarrow} G|^{\mathsf{BC}}\,; G!$. If $A <:^- \star$, then it must be that $A <:^- G$. Hence, the claim holds by induction.

Case $|\star \overset{p}{\Longrightarrow} G|^{\mathsf{BC}} = G?^p$ is safe for $\overline{p}$ and $\star <:^- G$ holds.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Case $|\star \xRightarrow{p} B|^{\mathsf{BC}} = G?^{p} \mathbin{;} |G \xRightarrow{p} B|^{\mathsf{BC}}$ (where $B \neq \star$, $B \neq G$, and $G \sim B$). $\star <:^{-} B$ is satisfied regardless of B. Hence, it must be that $G = \star \to \star$ so that $B = A' \to B'$ and we need to examine $|\star \to \star \xRightarrow{p} A' \to B'|^{\mathsf{BC}} = |A' \xRightarrow{\overline{p}} \star|^{\mathsf{BC}} \to |\star \xRightarrow{p} B'|^{\mathsf{BC}}$. As $A' <:^{+} \star$ and $\star <:^{-} B'$ we can argue by induction that $|A' \xRightarrow{\overline{p}} \star|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$ and $|\star \xRightarrow{p} B'|^{\mathsf{BC}}\ \mathsf{safe}_{\mathsf{C}}\ \overline{p}$.

The reverse implication is proved by similar mutual induction on the definition of the translation. □

B Associativity of composition

Lemma 31 (Composition is Associative)**.** *For any* $r : A \Longrightarrow B$, $s : B \Longrightarrow C$, *and* $t : C \Longrightarrow D$, $(r \fatsemi s) \fatsemi t = r \fatsemi (s \fatsemi t)$.

Proof We prove the following five variants of associativity simultaneously by induction on the sum of the sizes of the coercions.

1. For any $s_1 : A \Longrightarrow B$, $s_2 : B \Longrightarrow C$, and $s_3 : C \Longrightarrow D$, $(s_1 \fatsemi s_2) \fatsemi s_3 = s_1 \fatsemi (s_2 \fatsemi s_3)$.
2. For any $i_1 : A \Longrightarrow B$, $s_2 : B \Longrightarrow C$, and $s_3 : C \Longrightarrow D$, $(i_1 \fatsemi s_2) \fatsemi s_3 = i_1 \fatsemi (s_2 \fatsemi s_3)$
3. For any $g_1 : A \Longrightarrow B$, $i_2 : B \Longrightarrow C$, and $s_3 : C \Longrightarrow D$, $(g_1 \fatsemi i_2) \fatsemi s_3 = g_1 \fatsemi (i_2 \fatsemi s_3)$
4. For any $g_1 : A \Longrightarrow B$, $g_2 : B \Longrightarrow C$, and $i_3 : C \Longrightarrow D$, $(g_1 \fatsemi g_2) \fatsemi i_3 = g_1 \fatsemi (g_2 \fatsemi i_3)$
5. For any $g_1 : A \Longrightarrow B$, $g_2 : B \Longrightarrow C$, and $g_3 : C \Longrightarrow D$, $(g_1 \fatsemi g_2) \fatsemi g_3 = g_1 \fatsemi (g_2 \fatsemi g_3)$

We prove each of the five parts as follows.

1. Proceed by cases on s_1. The case $s_1 = \mathtt{id}_\star$ is trivial and the cases for $s_1 = (G?^{p} \mathbin{;} i_1)$ and $s_1 = i_1$ are by part 2.
2. Proceed by cases on i_1 and s_2, using part 3 and inversion on the typing derivations.
3. Proceed by cases on i_2 and s_3, using part 4 and inversion on the typing derivations.
4. Proceed by cases on g_1, g_2, and g_3, using part 1 of the induction hypothesis and inversion on the typing derivations. □

C Bisimulation between λC and λS

Here we give the full proof of Proposition 19.

Lemma 32 (Compose Identity)**.** $s \fatsemi |\mathtt{id}_A|^{\mathsf{CS}} = s$ *and* $|\mathtt{id}_A|^{\mathsf{CS}} \fatsemi s = s$

Proof The proof is a straightforward induction on s and A. □

Lemma 33. *Suppose* $M \longrightarrow_{\mathsf{S}}^{*} V_1$.

1. *If* $V_1\langle s \rangle \longrightarrow_{\mathsf{S}}^{*} V_2$, *then* $M\langle s \rangle \longrightarrow_{\mathsf{S}}^{*} V_2$.
2. *If* $V_1\langle s \rangle \longrightarrow_{\mathsf{S}}^{*} \mathtt{blame}\, p$, *then* $M\langle s \rangle \longrightarrow_{\mathsf{S}}^{*} \mathtt{blame}\, p$.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Lemma 34. *Suppose* $M\langle s\rangle \longrightarrow^* V_1$.

1. *If* $V_1\langle t\rangle \longrightarrow_{\mathsf{S}}^* V_2$, *then* $M\langle s \fatsemi t\rangle \longrightarrow_{\mathsf{S}}^* V_2$.
2. *If* $V_1\langle t\rangle \longrightarrow_{\mathsf{S}}^* \mathtt{blame}\, p$, *then* $M\langle s \fatsemi t\rangle \longrightarrow_{\mathsf{S}}^* \mathtt{blame}\, p$.

Proof Direct by Lemma 33, instantiating the M of Lemma 33 with $M\langle s\rangle$ and the s with t. □

Lemma 35 (Substitution Preserves Bisimulation)**.** *If* $M \approx M'$ *and* $N \approx N'$, *then* $M[x := N] \approx M'[x := N']$.

Proof The proof is a straightforward induction on $M \approx M'$.

Case $\dfrac{}{k \approx k}$

We conclude that $k \approx k$.

Case $\dfrac{\vec{M} \approx \vec{M}'}{op(\vec{M}) \approx op(\vec{M}')}$

By the induction hypothesis, we have $\vec{M}[x := N] \approx \vec{M}'[x := N']$. We conclude that $op(\vec{M}[x := N]) \approx op(\vec{M}'[x := N'])$.

Case $\dfrac{}{y \approx y}$

Suppose $y = x$. Then we conclude that $N \approx N'$.

Suppose $y \neq x$. Then we conclude $y \approx y$.

Case $\dfrac{M \approx M'}{\lambda y{:}A.\, M \approx \lambda y{:}A.\, M'}$

By the induction hypothesis, we have $M[x := N] \approx M'[x := N']$. We conclude that $\lambda y{:}A.\, M[x := N] \approx \lambda y{:}A.\, M'[x := N']$.

Case $\dfrac{L \approx L' \qquad M \approx M'}{L\, M \approx L'\, M'}$

By the induction hypothesis, we have $L[x := N] \approx L'[x := N']$ and $M[x := N] \approx M'[x := N']$. We conclude that $L[x := N]\, M[x := N] \approx L'[x := N]\, M'[x := N]$.

Case $\dfrac{}{\mathtt{blame}\, p \approx \mathtt{blame}\, p}$

We conclude that $\mathtt{blame}\, p \approx \mathtt{blame}\, p$.

Case $\dfrac{M \approx M' \qquad |\mathtt{id}_A|^{\mathsf{CS}} = s}{M \approx M'\langle s\rangle}$

By the induction hypothesis, we have $M[x := N] \approx M'[x := N']$. We conclude that $M[x := N] \approx M'[x := N]\langle s\rangle$.

Case $\dfrac{M \approx M'\langle s\rangle \qquad |c|^{\mathsf{CS}} = t}{M\langle c\rangle \approx M'\langle s \fatsemi t\rangle}$

By the induction hypothesis, we have $M[x := N] \approx M'[x := N']\langle s\rangle$. We conclude that $M[x := N] \approx M'[x := N]\langle s \fatsemi t\rangle$.

Case $\dfrac{M \approx L'\langle r\rangle\, M'\langle s\rangle \qquad |d|^{\mathsf{CS}} = t}{M\langle d\rangle \approx (L'\langle r \fatsemi (s \to t)\rangle)\, M'}$

By the induction hypothesis, we have

$$M[x := N] \approx L'[x := N']\langle r\rangle\, M'[x := N']\langle s\rangle.$$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

We conclude that

$$M[x := N]\langle d\rangle \approx L'[x := N]\langle r \fatsemi (s \to t)\rangle\, M'[x := N'].$$

□

Lemma 36 (Values don't bisimulate application)**.** *If $V \approx M'$, then M' is either a value or a cast, but not an application.*

Proof The proof is by induction on the derivation of $V \approx M'$. Consider the two rules where applications appear on the right.

- In the congruence rule for application, the left-hand side is an application, but V is a value.
- In rule (iii), the induction hypothesis tells us that $L'\langle r\rangle\, M'\langle s\rangle$ cannot be an application, but it is. □

Lemma 37. *If $M \approx M'\langle s\rangle\langle t\rangle$, then $M \approx M'\langle s \fatsemi t\rangle$.*

Proof The proof is by induction on $M \approx M'\langle s\rangle\langle t\rangle$. There are just two cases in which the right-hand side is a cast.

Case $\dfrac{M \approx M'\langle s\rangle \qquad |\mathtt{id}_A|^{\mathsf{CS}} = t}{M \approx M'\langle s\rangle\langle t\rangle}$

We need to show that $M \approx M'\langle s \fatsemi t\rangle$, but by Lemma 32, noting that $|\mathtt{id}_A|^{\mathsf{CS}} = t$, we have $s \fatsemi t = s$. So we conclude using $M \approx M'\langle s\rangle$.

Case $\dfrac{M \approx M'\langle s\rangle\langle t_1\rangle \qquad |c|^{\mathsf{CS}} = t_2}{M\langle c\rangle \approx M'\langle s\rangle\langle t_1 \fatsemi t_2\rangle}$

By the induction hypothesis, we have $M \approx M'\langle s \fatsemi t_1\rangle$. Then by rule (ii), we conclude that $M\langle c\rangle \approx M'\langle s \fatsemi t_1 \fatsemi t_2\rangle$. □

Proposition 19 (Bisimulation, $\lambda\mathsf{C}$ to $\lambda\mathsf{S}$)**.**
Assume $\vdash_{\mathsf{C}} M : A$ and $\vdash_{\mathsf{S}} M' : A$ and $M \approx M'$.

1. *If $M \longrightarrow_{\mathsf{C}} N$ then $M' \longrightarrow_{\mathsf{S}}^{*} N'$ and $N \approx N'$ for some N'.*
2. *If $M' \longrightarrow_{\mathsf{S}} N'$ then $M \longrightarrow_{\mathsf{C}}^{*} N$ and $N \approx N'$ for some N.*
3. *If $M = V$ then $M' \longrightarrow_{\mathsf{S}}^{*} V'$ 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 $M' \longrightarrow_{\mathsf{S}}^{*} \mathtt{blame}\, p$.*
6. *If $M' = \mathtt{blame}\, p$ then $M = \mathtt{blame}\, p$.*

Proof We first prove parts 3 and 4 of this proposition, then prove parts 1 and 2 (which depend on parts 3 and 4), and conclude with the proofs of parts 5 and 6.

Part 3. We show that the term M' on the right can become a value V' that corresponds to V. We proceed by induction on V.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Case $V = k$**.** We proceed by cases on $k \approx M'$, but we only have one case to consider.

Subcase $\dfrac{}{k \approx k}$ Take $V' = k$.

Case $V = \lambda x{:}A.\, N$**.** We proceed by induction on $(\lambda x{:}A.\, N) \approx M'$. There are two rules that apply when there is a lambda abstraction on the left-hand side.

Subcase $\dfrac{N \approx N'}{\lambda x{:}A.\, N \approx \lambda x{:}A.\, N'}$

We take $V' = \lambda x{:}A.\, N'$.

Subcase $\dfrac{\lambda x{:}A.\, N \approx M_1' \qquad |\mathtt{id}_{A\to B}|^{\mathsf{CS}} = |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}}}{\lambda x{:}A.\, N \approx M_1' \langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}} \rangle}$ (i)

By the inner induction hypothesis we have the following.

$$\begin{array}{ccc} \lambda x{:}A.\, N & \rightsquigarrow & M_1' \\ & \searrow & \downarrow \\ & & V_1' \end{array}$$

Now suppose $V_1' = \lambda x{:}A.\, N'$. Then $V_1' \langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}} \rangle$ is a value. We conclude by Lemma 33.

$$\begin{array}{ccc} \lambda x{:}A.\, N & \rightsquigarrow & M_1' \langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}} \rangle \\ & \searrow & \downarrow \\ & & (\lambda x{:}A.\, N') \langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}} \rangle \end{array}$$

On the other hand, suppose $V_1' = U' \langle s' \to t' \rangle$. Then we have the following reduction (using Lemma 32).

$$U' \langle s' \to t' \rangle \langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}} \rangle \longrightarrow_{\mathsf{S}} U' \langle s' \to t' \rangle$$

Again we conclude by Lemma 33.

$$\begin{array}{ccc} \lambda x{:}A.\, N & \rightsquigarrow & M_1' \langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}} \rangle \\ & \searrow & \downarrow \\ & & U' \langle s' \to t' \rangle \end{array}$$

Case $V = V_1 \langle G! \rangle$**.** We proceed by induction on $V_1 \langle G! \rangle \approx M'$.

Subcase $\dfrac{V_1 \langle G! \rangle \approx M_1'}{V_1 \langle G! \rangle \approx M_1' \langle |\mathtt{id}_\star|^{\mathsf{CS}} \rangle}$ (i)

The inner induction hypothesis gives us

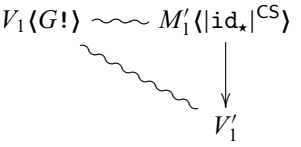
$$\begin{array}{ccc} V_1 \langle G! \rangle & \rightsquigarrow & M_1' \\ & \searrow & \downarrow \\ & & V_1' \end{array}$$

Note that

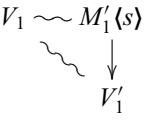
$$V_1' \langle |\mathtt{id}_\star|^{\mathsf{CS}} \rangle \longrightarrow_{\mathsf{S}} V_1'$$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

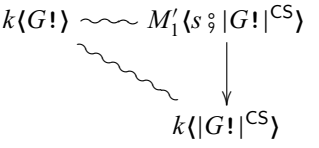
So by Lemma 33 we conclude.



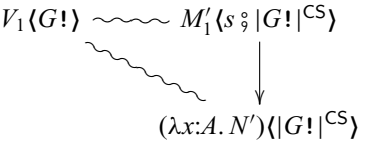
Subcase $\dfrac{V_1 \approx M_1' \langle s \rangle}{V_1 \langle G! \rangle \approx M_1' \langle s \fatsemi |G!|^{\mathsf{CS}} \rangle}$ (ii)
The inner induction hypothesis gives us



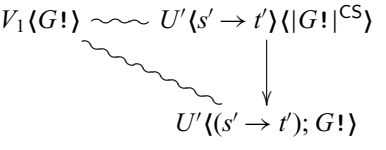
Suppose $V_1' = k$. Then $k\langle |G!|^{\mathsf{CS}} \rangle$ is a value. By Lemma 34 we have



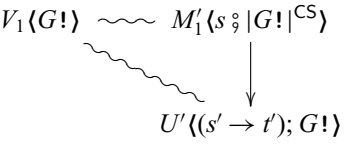
Suppose $V_1' = \lambda x{:}A.\, N'$. Then $(\lambda x{:}A.\, N')\langle |G!|^{\mathsf{CS}} \rangle$ is a value. By Lemma 34 we have



Suppose $V_1' = U'\langle g; H! \rangle$. Then V_1' has type $\star$, but that contradicts it having type G.
Suppose $V_1' = U'\langle s' \to t' \rangle$. We have



By Lemma 34 we conclude



Subcase rule (iii)
For this rule to apply, M' must be an application. But $V_1\langle G! \rangle$ is a value, so Lemma 36 tells us that M' cannot be an application, yielding a contradiction.
Case $V = V_1\langle c \to d \rangle$**.** We proceed by induction on $V_1\langle c \to d \rangle \approx M'$. There are three cases to consider.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Subcase $\dfrac{V_1\langle c \to d\rangle \approx M_1' \qquad \vdash V_1\langle c \to d\rangle : A \to B \qquad |\mathtt{id}_{A\to B}|^{\mathsf{CS}} = t}{V_1\langle c \to d\rangle \approx M_1'\langle t\rangle}$ (i)

We have $M_1' \longrightarrow^* V_1'$ and $V_1\langle c \to d\rangle \approx V_1'$ by the inner induction hypothesis. We proceed by cases on V_1' with the knowledge that it is of function type.

Suppose $V_1' = \lambda x{:}A.\, M_2'$. Then $V_1'\langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}}\rangle$ is a value and we apply Lemma 33 to obtain the following reduction, relating the left to the bottom right by rule (i).

$$\begin{array}{ccc} V_1\langle c \to d\rangle & \rightsquigarrow & M_1'|\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_A|^{\mathsf{CS}} \\ & \searrow & \downarrow \\ & & V_1'\langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}}\rangle \end{array}$$

Suppose $V_1' = U\langle s' \to t'\rangle$.

$$\begin{array}{ccccc} V_1\langle c \to d\rangle & \rightsquigarrow & U\langle s' \to t'\rangle\langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_A|^{\mathsf{CS}}\rangle & & \\ \| & & \downarrow & & \\ & & U\langle (s' \to t') \fatsemi (|\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_A|^{\mathsf{CS}})\rangle & & \\ & & \| \text{Lemma 32} & & \\ V_1\langle c \to d\rangle & \rightsquigarrow & U\langle s' \to t'\rangle & & \end{array}$$

Then we conclude this subcase by Lemma 33.

Subcase $\dfrac{V_1 \approx M_1'\langle s\rangle \qquad |c \to d|^{\mathsf{CS}} = t}{V_1\langle c \to d\rangle \approx M_1'\langle s \fatsemi t\rangle}$ (ii)

We have $M_1'\langle s\rangle \longrightarrow^* V_1'$ and $V_1 \approx V_1'$ by the inner induction hypothesis. We proceed by case analysis on V_1' with the knowledge that it is of function type.

Suppose $V_1' = \lambda x{:}A.\, M_2'$. Then $V_1'\langle |c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}}\rangle$ is a value. So we have $M_1'\langle s \fatsemi t\rangle \longrightarrow_{\mathsf{S}}^* V_1'\langle |c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}}\rangle$ by Lemma 34 and we relate the left to the right by rules (i) and (ii).

Suppose $V_1' = U\langle s' \to t'\rangle$. Then

$$U\langle s' \to t'\rangle\langle |c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}}\rangle \longrightarrow_{\mathsf{S}} U\langle (s' \to t') \fatsemi (|c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}})\rangle$$

and by rule (ii) (recalling $V_1 \approx V_1'$) we have

$$V_1\langle c \to d\rangle \approx U\langle (s' \to t') \fatsemi (|c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}})\rangle$$

We conclude by Lemma 34.

Subcase rule (iii)

(Rule (iii) does not apply because the premise would relate a value to a function application.)

Part 4. We need to prove that

$$\textit{if } M' = V' \textit{ then } M \longrightarrow_{\mathsf{C}}^* V \textit{ and } V \approx V' \textit{ for some } V.$$

We proceed by induction on V'.

Case $V' = k$**.** By inversion on $M \approx k$ we have $M = k$, which is already a value, so we take $V = M$.

Case $V' = \lambda x{:}A.\, N$**.** By inversion on $M \approx \lambda x{:}A.\, N$ we have $M = \lambda x{:}A.\, N'$ and take $V = M$.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Case $V' = U'\langle s \to t\rangle$**.** Inversion of $M \approx U'\langle s \to t\rangle$ gives us two cases.

Subcase $\dfrac{M \approx U' \qquad \vdash M : A \qquad |\mathtt{id}_A|^{\mathsf{CS}} = s \to t}{M \approx U'\langle s \to t\rangle}$(i)

By the induction hypothesis, $M \longrightarrow_{\mathsf{C}}^{*} V$ where $V \approx U'$. Then the left and right sides are related by rule (i).

Subcase. $\dfrac{M_1 \approx U'\langle s'\rangle \qquad |c|^{\mathsf{CS}} = t'}{M_1\langle c\rangle \approx U'\langle s' \fatsemi t'\rangle}$(ii)

We have $M = M_1\langle c\rangle$ and $(s' \fatsemi t') = s \to t$. By the induction hypothesis, $M_1 \longrightarrow_{\mathsf{C}}^{*} V_1$ where $V_1 \approx U'\langle s'\rangle$. We proceed with a nested induction on c.

Suppose $c = \mathtt{id}_A$.

$$\begin{array}{ccc} V_1\langle \mathtt{id}_A\rangle & \rightsquigarrow & U'\langle s' \fatsemi |\mathtt{id}_A|^{\mathsf{CS}}\rangle \\ \downarrow & & \Vert\ \text{Lemma 32} \\ V_1 & \rightsquigarrow & U'\langle s'\rangle \end{array}$$

Suppose $c = G!$. Then $t' = |G!|^{\mathsf{CS}} = |\mathtt{id}_G|^{\mathsf{CS}}; G!$, but that contradicts $(s' \fatsemi t') = s \to t$. Suppose $c = G?^p$. Then $t' = G?^p; |\mathtt{id}_G|^{\mathsf{CS}}$. With $(s' \fatsemi t') = s \to t$, we have $s' = (s \to t)$; $G!$. Then from $V_1 \approx U'\langle (s \to t); G!\rangle$ we have $V_1 = V_2\langle G!\rangle$ with $V_2 \approx U'\langle s \to t\rangle$ for some V_2. So we obtain:

$$\begin{array}{ccc} V_1\langle G?^p\rangle & \rightsquigarrow & U'\langle (s \to t); G! \fatsemi G?^p; |\mathtt{id}_G|^{\mathsf{CS}}\rangle \\ \downarrow & & \\ V_2\langle G!\rangle\langle G?^p\rangle & & \Vert\ \text{Lemma 32} \\ \downarrow & & \\ V_2 & \rightsquigarrow & U'\langle s \to t\rangle \end{array}$$

Next suppose $c = c_1 \to c_2$, then $V_1\langle c_1 \to c_2\rangle$ is already a value. From $V_1 \approx U'\langle s'\rangle$ and $|c|^{\mathsf{CS}} = t'$ we have $V_1\langle c\rangle \approx U'\langle s' \fatsemi t'\rangle$ by rule (ii).

Suppose $c = (c_1; c_2)$. We have $t' = |c_1|^{\mathsf{CS}} \fatsemi |c_2|^{\mathsf{CS}}$. We obtain the following with two uses of the the inner induction hypothesis.

$$\begin{array}{ccc} V_1\langle c_1; c_2\rangle & \rightsquigarrow & U'\langle s' \fatsemi t'\rangle \\ \downarrow & & \Vert \\ V_1\langle c_1\rangle\langle c_2\rangle & \rightsquigarrow & U'\langle s' \fatsemi |c_1|^{\mathsf{CS}} \fatsemi |c_2|^{\mathsf{CS}}\rangle \\ \text{IH}\downarrow & \overset{\text{IH}}{\rightsquigarrow} & \\ V_2\langle c_2\rangle & \overset{\text{IH}}{\rightsquigarrow} & \\ \text{IH}\downarrow & & \\ V_3 & & \end{array}$$

Suppose $c = \bot^{p}_{A \Rightarrow B}$. Then $t' = \bot^{p}_{A \Rightarrow B}$ and $(s' \fatsemi t') = \bot^{p}_{- \Rightarrow B}$, but $(s' \fatsemi t') = s \to t$ so we have a contradiction.

Case $V' = U\langle g; G!\rangle$**.** Considering $M \approx U\langle g; G!\rangle$, only rule (ii) applies.

Subcase $\dfrac{M_1 \approx U\langle s\rangle \qquad |c|^{\mathsf{CS}} = t}{M_1\langle c\rangle \approx U\langle s \fatsemi t\rangle}$(ii)

By the induction hypothesis, we have $M_1 \longrightarrow_{\mathsf{C}}^{*} V_1$ and $V_1 \approx U\langle s\rangle$. We proceed by nested induction on c.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Suppose $c = \mathtt{id}_\star$.

$$\begin{array}{ccc} V_1\langle \mathtt{id}_\star \rangle & \rightsquigarrow & U\langle s \fatsemi |\mathtt{id}_\star|^{\mathsf{CS}} \rangle \\ \downarrow & & \| \text{Lemma 32} \\ V_1 & \rightsquigarrow & U\langle s \rangle \end{array}$$

Suppose $c = H!$. Then we have $V_1\langle H! \rangle \approx U\langle s \fatsemi |H!|^{\mathsf{CS}} \rangle$.
Suppose $c = H?^p$. Then $t = |H?^p|^{\mathsf{CS}} = H?^p; |\mathtt{id}_H|^{\mathsf{CS}}$. But that contradicts $(s \fatsemi t) = (g; G!)$.
Suppose $c = c_1 \to c_2$. Then $t = |c_1 \to c_2|^{\mathsf{CS}} = |c_1|^{\mathsf{CS}} \to |c_2|^{\mathsf{CS}}$. But that contradicts $(s \fatsemi t) = (g; G!)$.
Suppose $c = (c_1; c_2)$. We use the same reasoning as for the corresponding case in $V' = U\langle s \to t \rangle$, that is, we obtain the following with two uses of the the inner induction hypothesis.

$$\begin{array}{ccccc} V_1\langle c_1; c_2 \rangle & & \rightsquigarrow & & U'\langle s' \fatsemi t' \rangle \\ \downarrow & & & & \| \\ V_1\langle c_1 \rangle\langle c_2 \rangle & & \rightsquigarrow & & U'\langle s' \fatsemi |c_1|^{\mathsf{CS}} \fatsemi |c_2|^{\mathsf{CS}} \rangle \\ \text{IH}\downarrow & \overset{\text{IH}}{\rightsquigarrow} & & & \\ V_2\langle c_2 \rangle & & \overset{\text{IH}}{\rightsquigarrow} & & \\ \text{IH}\downarrow & & & & \\ V_3 & & & & \end{array}$$

Suppose $c = \bot^p_{A \Rightarrow B}$. Then $t' = \bot^p_{A \Rightarrow B}$ and $(s' \fatsemi t') = \bot^p_{- \Rightarrow B}$, but $(s' \fatsemi t') = (g; G!)$ so we have a contradiction.

Part 1. We proceed by induction on $M \approx M'$, proving the statement:

If $M \longrightarrow_{\mathsf{C}} N$ *then* $M' \longrightarrow^*_{\mathsf{S}} N'$ *and* $N \approx N'$ *for some* N'.

Case $\dfrac{}{k \approx k}$

The statement is vacuously true because k cannot reduce.

Case $\dfrac{\vec{M} \approx \vec{M'}}{op(\vec{M}) \approx op(\vec{M'})}$

$$\begin{array}{ccc} op(\vec{k}) & \rightsquigarrow & op(\vec{M'}) \\ & \searrow & \downarrow \\ \downarrow & & op(\vec{k}) \\ & & \downarrow \\ \delta(op, \vec{k}) & \rightsquigarrow & \delta(op, \vec{k}) \end{array} \qquad \text{(by Part 3)}$$

Case $\dfrac{}{x \approx x}$

The statement is vacuously true because x cannot reduce.

Case $\dfrac{M \approx M'}{\lambda x{:}A.\, M \approx \lambda x{:}A.\, M'}$

The statement is vacouously true because lambda terms cannot reduce.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Case $\dfrac{M_1 \approx M_1' \qquad M_2 \approx M_2'}{M_1\, M_2 \approx M_1'\, M_2'}$

We proceed by case analysis on $M = M_1\, M_2 \longrightarrow_{\mathsf{C}} N$. So either M_1 reduces, M_2 reduces, or they are both values.

Suppose M_1 reduces, i.e., $M_1 \longrightarrow_{\mathsf{C}} M_3$. From $M_1\, M_2 \approx M'$, we have $M' = M_1'\, M_2'$ and $M_1 \approx M_1'$ and $M_2 \approx M_2'$. By the induction hypothesis, $M_1' \longrightarrow_{\mathsf{S}}^* M_3'$ and $M_3 \approx M_3'$. So $M_1'\, M_2' \longrightarrow_{\mathsf{S}}^* M_3'\, M_2'$ and $M_3\, M_2 \approx M_3'\, M_2'$.

The case for M_2 reducing is essentialy the same as for M_1 reducing.

Suppose M_1 and M_2 are values. Let $V_2 = M_2$. We consider the cases on M_1 with the knowledge that M_1 is of function type, so either

1. $M_1 = \lambda x{:}A.\, M_{11}$: part of beta redex, or
2. $M_1 = V\langle c \to d\rangle$: part of coercion redex.

We proceed with these two cases.

1. $(\lambda x{:}A.\, M_{11})\, V_2 \longrightarrow_{\mathsf{C}} M_{11}[x := V]$ We have

$$\begin{array}{ccc} (\lambda x{:}A.\, M_{11})\, V_2 & \rightsquigarrow & M_1'\, M_2' \\ \big\downarrow & & \big\downarrow \\ M_{11}[x := V_2] & \rightsquigarrow & V_1'\, V_2' \end{array} \qquad \text{(by Part 3)}$$

then proceed by case analysis on $(\lambda x{:}A.\, M_{11}) \approx V_1'$.

Subcase $\dfrac{M_{11} \approx M_{11}'}{\lambda x{:}A.\, M_{11} \approx \lambda x{:}A.\, M_{11}'}$

$$\begin{array}{ccc} (\lambda x{:}A.\, M_{11})\, V_2 & \rightsquigarrow & (\lambda x{:}A.\, M_{11}')\, V_2' \\ \big\downarrow & & \big\downarrow \\ M_{11}[x := V_2] & \rightsquigarrow & M_{11}'[x := V_2'] \end{array} \qquad \text{(by Lemma 35)}$$

Subcase $\dfrac{\lambda x{:}A.\, M_{11} \approx U'}{\lambda x{:}A.\, M_{11} \approx U'\langle |\mathtt{id}_{A\to B}|^{\mathsf{CS}}\rangle}$(i)

We have $U' = \lambda x{:}A.\, M_{11}'$ because $\lambda x{:}A.\, M_{11} \approx U'$.

$$\begin{array}{ccccc} (\lambda x{:}A.\, M_{11})\, V_2 & \rightsquigarrow & (\lambda x{:}A.\, M_{11}')\langle |\mathtt{id}_A|^{\mathsf{CS}} \to |\mathtt{id}_B|^{\mathsf{CS}}\rangle\, V_2' & & \\ \big\downarrow & & \big\downarrow & & \\ & & ((\lambda x{:}A.\, M_{11}')\, V_2'\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle)\langle |\mathtt{id}_B|^{\mathsf{CS}}\rangle & & \\ & & \big\downarrow & & \\ & & ((\lambda x{:}A.\, M_{11}')\, V_2'')\langle |\mathtt{id}_B|^{\mathsf{CS}}\rangle & & \text{(Part 3)} \\ & & \big\downarrow & & \\ M_{11}[x := V_2] & \rightsquigarrow\rightsquigarrow & M_{11}'[x := V_2'']\langle |\mathtt{id}_B|^{\mathsf{CS}}\rangle & & \text{(Lemma 35)} \end{array}$$

2. $(V\langle c \to d\rangle)\, W \longrightarrow_{\mathsf{C}} (V\, W\langle c\rangle)\langle d\rangle$

We proceed by induction on $V\langle c \to d\rangle \approx M_1'$. There are three cases to consider.

Subcase $\dfrac{V\langle c \to d\rangle \approx M_{11}'}{V\langle c \to d\rangle \approx M_{11}'\langle |\mathtt{id}_{A\to B}|^{\mathsf{CS}}\rangle}$(i)

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

By induction we have $M'_{11} \longrightarrow^* V'_{11}$ and $V\langle c \to d\rangle \approx V'_{11}$. We proceed by inversion on the later, noting that rule (iii) cannot apply because it would require V'_{11} to be an application.

Suppose $V\langle c \to d\rangle \approx V'_{11}$ was by rule (i). So $V'_{11} = U'\langle |\mathtt{id}_{A\to B}|^{\mathsf{CS}}\rangle$ and $V\langle c \to d\rangle \approx U'$. But that's impossible because there are no rules that relate a cast on the left with a lambda abstraction on the right.

Suppose $V\langle c \to d\rangle \approx V'_{11}$ was by rule (ii). So we have $V'_{11} = U'\langle (s_1 \to s_2) \fatsemi |c \to d|^{\mathsf{CS}}\rangle$ and $V \approx U'\langle s_1 \to s_2\rangle$.

$$\begin{array}{ccc}
(V\langle c \to d\rangle)\, W & \rightsquigarrow & (M'_{11}\langle |\mathtt{id}_{A\to B}|^{\mathsf{CS}}\rangle)\, M'_2 \\
\big\downarrow & & \big\downarrow{\scriptstyle *} \\
 & & U'\langle (s_1 \to s_2) \fatsemi |c \to d|^{\mathsf{CS}}\rangle\langle |\mathtt{id}_{A\to B}|^{\mathsf{CS}}\rangle\, V'_2 \\
 & & \big\downarrow \\
 & & U'\langle (s_1 \to s_2) \fatsemi |c \to d|^{\mathsf{CS}} \fatsemi |\mathtt{id}_{A\to B}|^{\mathsf{CS}}\rangle\, V'_2 \\
 & & \big\| \text{Lemma 32} \\
(V\, W\langle c\rangle)\langle d\rangle & \rightsquigarrow & (U'\langle (s_1 \to s_2) \fatsemi |c \to d|^{\mathsf{CS}}\rangle)\, V'_2
\end{array}$$

The bottom left is related to the bottom right by rule (iii).

Subcase rule (ii).

$$\frac{V \approx M'_{11}\langle s\rangle}{V\langle c \to d\rangle \approx M'_{11}\langle s \fatsemi |c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}}\rangle}$$

$$\begin{array}{ccc}
(V\langle c \to d\rangle)\, W & \rightsquigarrow & (M'_{11}\langle s \fatsemi |c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}}\rangle)\, M'_2 \\
\big\downarrow & \nearrow & \\
(V\, W\langle c\rangle)\langle d\rangle & &
\end{array}$$

because

$$\dfrac{V \approx M'_{11}\langle s\rangle \qquad \dfrac{\dfrac{W \approx M'_2}{W \approx M'_2\langle \mathtt{id}\rangle}\text{(i)}}{W\langle c\rangle \approx M'_2\langle |c|^{\mathsf{CS}}\rangle}\text{(ii)}}{V\, W\langle c\rangle \approx (M'_{11}\langle s\rangle)\,(M'_2\langle |c|^{\mathsf{CS}}\rangle)}\text{(iii)}}{(V\, W\langle c\rangle)\langle d\rangle \approx (M'_{11}\langle s \fatsemi |c|^{\mathsf{CS}} \to |d|^{\mathsf{CS}}\rangle)\, M'_2}$$

Subcase rule (iii).

For this rule to apply, M'_1 must be an application. But $V\langle c \to d\rangle$ is a value and $V\langle c \to d\rangle \approx M'_1$, so Lemma 36 tells us that M'_1 cannot be an application, yielding a contradiction.

Case $\dfrac{M_1 \approx M'_1 \qquad \vdash M_1 : A \qquad |\mathtt{id}_A|^{\mathsf{CS}} = s}{M_1 \approx M'_1\langle s\rangle}$(i)

Assume $M_1 \longrightarrow_{\mathsf{C}} N$. By induction, we have $M'_1 \longrightarrow^*_{\mathsf{S}} N'$ and $N \approx N'$. Thus, we also have $M'_1\langle s\rangle \longrightarrow^*_{\mathsf{S}} N'\langle s\rangle$ and $N \approx N'\langle i\rangle$ by rule (i).

$$\begin{array}{ccc}
M_1 & \rightsquigarrow & M'_1\langle s\rangle \\
\big\downarrow & & \big\downarrow{\scriptstyle *} \\
N & \rightsquigarrow & N'\langle s\rangle
\end{array}$$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

Case $\dfrac{M_1 \approx M_1'\langle s\rangle \qquad |c|^{\mathsf{CS}} = t}{M_1\langle c\rangle \approx M_1'\langle s \fatsemi t\rangle}$ (ii)

We proceed by case analysis on $M_1\langle c\rangle \longrightarrow_{\mathsf{C}} N$.

1. Case $V_1\langle \mathtt{id}_A\rangle \longrightarrow_{\mathsf{C}} V_1$

$$\begin{array}{ccc} V_1\langle \mathtt{id}_A\rangle & \rightsquigarrow & M_1'\langle s \fatsemi |\mathtt{id}_A|^{\mathsf{CS}}\rangle \\ \downarrow & \swarrow & \\ V_1 & & \end{array}$$

2. Case $V_1\langle G!\rangle\langle G?^p\rangle \longrightarrow_{\mathsf{C}} V_1$

$$\begin{array}{ccc} V_1\langle G!\rangle\langle G?^p\rangle & \rightsquigarrow & M_1'\langle s' \fatsemi G! \fatsemi G?^p\rangle \\ \downarrow & & \| \\ V_1 & \rightsquigarrow & M_1'\langle s'\rangle \end{array}$$

3. Case $V_1\langle G!\rangle\langle H?^p\rangle \longrightarrow_{\mathsf{C}} \mathtt{blame}\, p$

$$\begin{array}{ccccc} V_1\langle G!\rangle\langle H?^p\rangle & \rightsquigarrow & M_1\langle s' \fatsemi G! \fatsemi H?^p; |\mathtt{id}_H|^{\mathsf{CS}}\rangle & & \\ \downarrow & & \| & & \\ & & M_1\langle \bot^{GpH}\rangle & & \\ & & \downarrow * & & \\ & & V_1\langle \bot^{GpH}\rangle & & \text{(by Part 3)} \\ \downarrow & & \downarrow & & \\ \mathtt{blame}\, p & \rightsquigarrow & \mathtt{blame}\, p & & \end{array}$$

4. Case $V_1\langle c; d\rangle \longrightarrow_{\mathsf{C}} V_1\langle c\rangle\langle d\rangle$.

$$\begin{array}{ccc} V_1\langle c; d\rangle & \rightsquigarrow & M_1'\langle s \fatsemi t\rangle \\ \downarrow & \swarrow & \\ V_1\langle c\rangle\langle d\rangle & & \end{array}$$

We have $V_1 \approx M_1'\langle s\rangle$ and $|c; d|^{\mathsf{CS}} = (|c|^{\mathsf{CS}} \fatsemi |d|^{\mathsf{CS}}) = t$. We conclude that $V_1\langle c\rangle\langle d\rangle \approx M_1'\langle s \fatsemi |c|^{\mathsf{CS}} \fatsemi |d|^{\mathsf{CS}}\rangle$ by the associativity of composition and two uses of rule (ii).

5. Case $V_1\langle \bot^{GpH}\rangle \longrightarrow_{\mathsf{C}} \mathtt{blame}\, p$

We have $V_1 \approx M_1'\langle s\rangle$. By Part 3 we have $M_1'\langle s\rangle \longrightarrow_{\mathsf{S}}^* V_1'$. Also, $V_1'\langle \bot^{GpH}\rangle \longrightarrow_{\mathsf{S}} \mathtt{blame}\, p$. Thus, $M_1'\langle s \fatsemi \bot^{GpH}\rangle \longrightarrow_{\mathsf{S}}^* \mathtt{blame}\, p$ by Lemma 34.

$$\begin{array}{ccc} V_1\langle \bot^{GpH}\rangle & \rightsquigarrow & M_1'\langle s \fatsemi \bot^{GpH}\rangle \\ \downarrow & & \downarrow * \\ \mathtt{blame}\, p & \rightsquigarrow & \mathtt{blame}\, p \end{array}$$

Case $\dfrac{M_1 \approx M_1'\langle r\rangle \qquad M_2 \approx M_2'\langle s_n \fatsemi \cdots \fatsemi s_1\rangle \qquad |d_i|^{\mathsf{CS}} = t_i \;{}^{\forall i \in 1\ldots n}}{(M_1\; M_2)\langle d_1\rangle \cdots \langle d_n\rangle \approx M_1'\langle r \fatsemi (s_1 \to t_1) \fatsemi \cdots \fatsemi (s_n \to t_n)\rangle\; M_2'}$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

By inversion on $(M_1\ M_2)\langle d_1\rangle \cdots \langle d_n\rangle \longrightarrow_{\mathsf{C}} N$, we have

$$\frac{\begin{array}{c} M_1\ M_2 \longrightarrow_{\mathsf{C}} N_1 \\ \vdots \end{array}}{(M_1\ M_2)\langle d_1\rangle \cdots \langle d_n\rangle \longrightarrow_{\mathsf{C}} N_1\langle d_1\rangle \cdots \langle d_n\rangle}$$

The proof of this case then proceeds like the case of the congruence rule for application.

Part 2. We proceed by induction on $M \approx M'$, proving the statement:

If $M' \longrightarrow_{\mathsf{S}} N'$ *then* $M \longrightarrow_{\mathsf{C}}^{*} N$ *and* $N \approx N'$ *for some* N.

Case $\dfrac{}{k \approx k}$

The statement is vacuously true because k cannot reduce.

Case $\dfrac{\vec{M} \approx \vec{M}'}{op(\vec{M}) \approx op(\vec{M}')}$

$$\begin{array}{lccc} & op(\vec{M}) & \rightsquigarrow & op(\vec{k}) \\ & \downarrow & \nearrow & \\ \text{(by Part 4)} & op(\vec{k}) & & \downarrow \\ & \downarrow & & \\ & \delta(op,\vec{k}) & \rightsquigarrow & \delta(op,\vec{k}) \end{array}$$

Case $\dfrac{}{x \approx x}$

The statement is vacuously true because x cannot reduce.

Case $\dfrac{M \approx M'}{\lambda x{:}A.\, M \approx \lambda x{:}A.\, M'}$

The statement is vacouously true because lambda terms cannot reduce.

Case $\dfrac{M_1 \approx M_1' \qquad M_2 \approx M_2'}{M_1\ M_2 \approx M_1'\ M_2'}$

We proceed by case analysis on $M_1'\ M_2' \longrightarrow_{\mathsf{S}} N'$.

1. Case $(\lambda x{:}A.\, M_{11}')\ V_2' \longrightarrow_{\mathsf{S}} M_{11}'[x := V_2']$

$$\begin{array}{lccc} & M_1\ M_2 & \rightsquigarrow & (\lambda x{:}A.\, M_{11}')\ V_2' \\ & \downarrow^{*} & \nearrow & \\ \text{(by Part 4)} & (\lambda x{:}A.\, M_{11})\ V_2 & & \downarrow \\ & \downarrow & & \\ & M_{11}[x := V_2] & \rightsquigarrow & M_{11}'[x := V_2'] \end{array} \qquad \text{(by Lemma 35)}$$

2. Case $(U'\langle s \to t\rangle)\ W' \longrightarrow_{\mathsf{S}} (U'\ W'\langle s\rangle)\langle t\rangle$

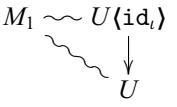
$$\begin{array}{lccc} & M_1\ M_2 & \rightsquigarrow & (U'\langle s \to t\rangle)\ W' \\ & \downarrow^{*} & \nearrow & \\ \text{(by Part 4)} & (V_1\langle c_1 \to d_1\rangle \cdots \langle c_n \to d_n\rangle)\ V_2 & & \downarrow \\ & \downarrow^{*} & & \\ & (V_1\ (V_2\langle c_n\rangle \cdots \langle c_1\rangle))\langle d_1\rangle \cdots \langle d_n\rangle & \rightsquigarrow & (U'\ W'\langle s\rangle)\langle t\rangle \end{array}$$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

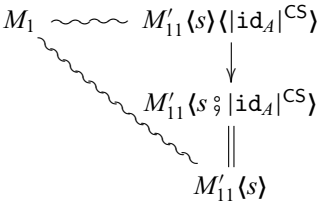
Case $\dfrac{M_1 \approx M_1' \qquad \vdash M_1 : A \qquad |\mathtt{id}_A|^{\mathsf{CS}} = s}{M_1 \approx M_1'\langle s\rangle}$ (i)

We proceed by cases on $M_1'\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle \longrightarrow_{\mathsf{S}} N'$.

1. $U\langle \mathtt{id}_\iota\rangle \longrightarrow_{\mathsf{S}} U$



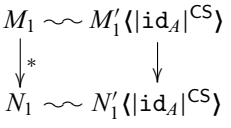
2. $M_{11}'\langle s\rangle\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle \longrightarrow_{\mathsf{S}} M_{11}'\langle s \fatsemi |\mathtt{id}_A|^{\mathsf{CS}}\rangle$.
We have $M_1 \approx M_{11}'\langle s\rangle$.



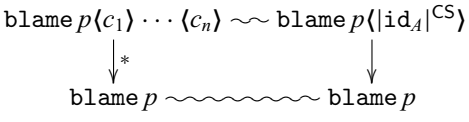
Lemma 32

3. $\dfrac{M_1' \longrightarrow_{\mathsf{S}} N_1'}{M_1'\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle \longrightarrow_{\mathsf{S}} N_1'\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle}$

By induction, we have $M_1 \longrightarrow_{\mathsf{C}}^{*} N_1$ and $N_1 \approx N_1'$. Thus, $N_1 \approx N_1'\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle$ by rule (i).



4. $(\mathtt{blame}\, p)\langle |\mathtt{id}_A|^{\mathsf{CS}}\rangle \longrightarrow_{\mathsf{S}} \mathtt{blame}\, p$
We have $M_1 \approx \mathtt{blame}\, p$. So M_1 is $\mathtt{blame}\, p$ surrounded by zero or more coercion applications: $M_1 = \mathtt{blame}\, p\langle c_1\rangle \cdots \langle c_n\rangle$.

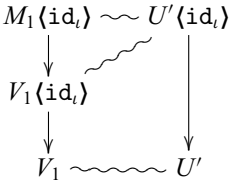


Case $\dfrac{M_1 \approx M_1'\langle s\rangle \qquad |c|^{\mathsf{CS}} = t}{M_1\langle c\rangle \approx M_1'\langle s \fatsemi t\rangle}$ (ii)

We proceed by case analysis on $M_1'\langle s \fatsemi t\rangle \longrightarrow_{\mathsf{S}} N'$.

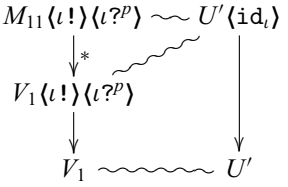
1. Case $U'\langle \mathtt{id}_\iota\rangle \longrightarrow_{\mathsf{S}} U'$.
There are two cases for $s \fatsemi t = \mathtt{id}_\iota$:

   a. $s = t = \mathtt{id}_\iota$

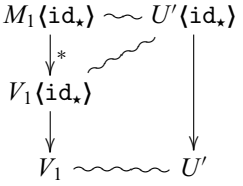


https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

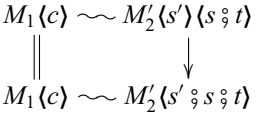
b. $s = \mathtt{id}_\iota ; \iota\,!$ and $t = \iota?^p ; \mathtt{id}_\iota$. In that case, the assumption is $M_1 \approx U'\langle \mathtt{id}_\iota ; \iota\,! \rangle$. By inversion, $M_1 = M_{11}\langle \iota\,! \rangle$ and $M_{11} \approx U'\langle \mathtt{id}_\iota \rangle$. By further inversion, $M_{11} \approx U'$. Hence:



2. Case $U'\langle \mathtt{id}_\star \rangle \longrightarrow_{\mathsf{S}} U'$



3. Case $M_2'\langle s' \rangle\langle s \fatsemi t \rangle \longrightarrow_{\mathsf{S}} M_2'\langle s' \fatsemi s \fatsemi t \rangle$

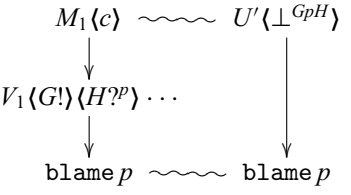


We have $M_1 \approx M_2'\langle s' \rangle\langle s \rangle$ and therefore $M_1 \approx M_2'\langle s' \fatsemi s \rangle$. With $|t|^{\mathsf{CS}} = c$ we conclude $M_1\langle c \rangle \approx M_2'\langle s' \fatsemi s \fatsemi t \rangle$ by rule (ii).

4. Case $U'\langle \bot^{GpG} \rangle \longrightarrow_{\mathsf{S}} \mathtt{blame}\, p$

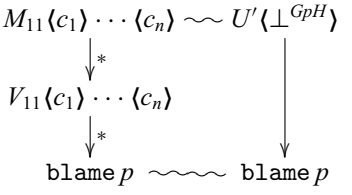
There are three ways that we could have $s \fatsemi t = \bot^{GpH}$.

a. $s = (g; G!)$, $t = (H?^p; i)$, and $G \neq H$



b. $s = \bot^{GpH}$

We have $M_1 \approx U'\langle \bot^{GpH} \rangle$, so $M_1 = M_{11}\langle c_1 \rangle \cdots \langle c_n \rangle$ with $c_n = c$, $|c_1|^{\mathsf{CS}} \fatsemi \cdots \fatsemi |c_n|^{\mathsf{CS}} = \bot^{GpH}$, and $M_{11} \approx U'$. By Part 4, $M_{11} \longrightarrow_{\mathsf{C}}^* V_{11}$ and $V_{11} \approx U'$. Then because $|c_1|^{\mathsf{CS}} \fatsemi \cdots \fatsemi |c_n|^{\mathsf{CS}} = \bot^{GpH}$, we have $V_{11}\langle c_1 \rangle \cdots \langle c_n \rangle \longrightarrow_{\mathsf{C}}^* \mathtt{blame}\, p$.



https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

c. $t = \bot^{GpH}$

$$\begin{array}{ccc} M_1\langle\bot^{GpH}\rangle & \rightsquigarrow & U'\langle\bot^{GpH}\rangle \\ \downarrow * & \nearrow & \downarrow \\ V_1\langle\bot^{GpH}\rangle & & \downarrow \\ \downarrow & & \downarrow \\ \texttt{blame}\, p & \rightsquigarrow & \texttt{blame}\, p \end{array}$$

Case $\dfrac{M \approx M_1'\langle r\rangle\ M_2'\langle s\rangle \qquad |d|^{\mathsf{CS}} = t}{M\langle d\rangle \approx M_1'\langle r \fatsemi (s \to t)\rangle\ M_2'}$

We proceed by induction on $M_1'\langle r \fatsemi (s \to t)\rangle\ M_2' \longrightarrow_{\mathsf{S}} N'$.

1. $U'\langle s' \to t'\rangle\ W' \longrightarrow_{\mathsf{S}}^{\mathcal{E}} (U'\ W'\langle s'\rangle)\langle t'\rangle$
   So $M_1' = U'$, $M_2' = W'$, $r = r_1 \to r_2$, $s' = s \fatsemi r_1$, $t' = r_2 \fatsemi t$.
   By the induction hypothesis

$$\begin{array}{ccc} M & \rightsquigarrow & U'\langle r_1 \to r_2\rangle\ W'\langle s\rangle \\ \downarrow * & & \downarrow \\ N & \rightsquigarrow & (U'\ W'\langle s\rangle\langle r_1\rangle)\langle r_2\rangle \end{array}$$

   so we also have

$$M\langle d\rangle \longrightarrow_{\mathsf{C}}^{*} N\langle d\rangle$$

   we need to show that

$$N\langle d\rangle \approx (U'\ W'\langle s \fatsemi r_1\rangle)\langle r_2 \fatsemi t\rangle$$

   Using rule (ii), it suffices to show that

$$N \approx (U'\ W'\langle s \fatsemi r_1\rangle)\langle r_2\rangle$$

   Because $N \approx (U'\ W'\langle s\rangle\langle r_1\rangle)\langle r_2\rangle$, there must be some subterm of N, call it L, such that $L \approx W'\langle s\rangle\langle r_1\rangle$, and therefore $L \approx W'\langle s \fatsemi r_1\rangle$ by Lemma 37, from which we conclude.

2. $\dfrac{M_1'\langle r \fatsemi (s \to t)\rangle \longrightarrow_{\mathsf{S}} N_1'}{M_1'\langle r \fatsemi (s \to t)\rangle\ M_2' \longrightarrow_{\mathsf{S}}^{\mathcal{E}} N_1'\ M_2'}(\mathcal{F} = \Box\ M_2')$
   From $M_1'\langle r \fatsemi (s \to t)\rangle \longrightarrow_{\mathsf{S}} N_1'$ we have $N_1' = N_{11}'\langle r \fatsemi (s \to t)\rangle$ and $M_1' \longrightarrow_{\mathsf{S}} N_{11}'$. So $N'\langle r\rangle\ M_2'\langle s\rangle \longrightarrow_{\mathsf{S}} N_{11}'\langle r\rangle\ M_2'\langle s\rangle$. Then by the induction hypothesis, we have $M \longrightarrow_{\mathsf{C}}^{*} N$ and $N \approx N_{11}'\langle r\rangle\ M_2'\langle s\rangle$. We conclude by rule (iii). To summarize, we have the following diagram.

$$\begin{array}{ccc} M\langle d\rangle & \rightsquigarrow & M_1'\langle r \fatsemi (s \to t)\rangle\ M_2' \\ \downarrow * & & \downarrow \\ N\langle d\rangle & \rightsquigarrow & N_{11}'\langle r \fatsemi (s \to t)\rangle\ M_2' \end{array}$$

3. $\dfrac{M_2' \longrightarrow_{\mathsf{S}} N_2'}{M_1'\langle r \fatsemi (s \to t)\rangle\ M_2' \longrightarrow_{\mathsf{S}}^{\mathcal{E}} N_2'\ M_2'}(\mathcal{F} = M_1'\langle r \fatsemi (s \to t)\rangle\ \Box)$

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

We have $M_1'\langle r\rangle\, M_2'\langle s\rangle \longrightarrow_{\mathsf{S}} M_1'\langle r\rangle\, N_2'\langle s\rangle$ and then by the induction hypothesis, $M \longrightarrow_{\mathsf{C}}^{*} N$ and $N \approx M_1'\langle r\rangle\, N_2'\langle s\rangle$ for some N. We conclude by rule (iii). To summarize, we have the following diagram.

$$\begin{array}{ccc} M\langle d\rangle & \sim\!\sim & M_1'\langle r \fatsemi (s{\to}t)\rangle\, M_2' \\ \Big\downarrow{\scriptstyle *} & & \Big\downarrow \\ N\langle d\rangle & \sim\!\sim & M_1'\langle r \fatsemi (s \to t)\rangle\, N_2' \end{array}$$

4. $\dfrac{}{M_1'\langle r \fatsemi (s{\to}t)\rangle\, (\texttt{blame}\, p) \longrightarrow_{\mathsf{S}}^{\mathcal{E}} \texttt{blame}\, p}\,(\mathcal{F} = M_1'\langle r \fatsemi (s{\to}t)\rangle\, \Box)$

We have $M_1'\langle r\rangle\, (\texttt{blame}\, p)\langle s\rangle \longrightarrow_{\mathsf{S}} M_1'\langle r\rangle\, (\texttt{blame}\, p)$. So by the induction hypothesis, $M \longrightarrow_{\mathsf{C}}^{*} N$ and $N \approx M_1'\langle r\rangle\, (\texttt{blame}\, p)$. So N must be of the form $(L\, \texttt{blame}\, p)\langle d_1\rangle \cdots \langle d_n\rangle$. Therefore $N \longrightarrow_{\mathsf{C}}^{*} \texttt{blame}\, p$. To summarize, we have the following diagram.

$$\begin{array}{ccc} M\langle d\rangle & \sim\!\sim\!\sim\!\sim\!\sim\!\sim\!\sim\!\sim & M_1'\langle r \fatsemi (s{\to}t)\rangle\, M_2' \\ \Big\downarrow{\scriptstyle *} & & \Big\downarrow \\ (L\, (\texttt{blame}\, p))\langle d_1\rangle \cdots \langle d_n\rangle\langle d\rangle & & \\ \Big\downarrow{\scriptstyle *} & & \\ \texttt{blame}\, p & \sim\!\sim\!\sim\!\sim\!\sim\!\sim\!\sim\!\sim\!\sim\!\sim & \texttt{blame}\, p \end{array}$$

Part 5.

$M = \texttt{blame}\, p$. We need to show that $M' \longrightarrow_{\mathsf{S}}^{*} \texttt{blame}\, p$. We proceed by cases on $\texttt{blame}\, p \approx M'$.

Case $\dfrac{}{\texttt{blame}\, p \approx \texttt{blame}\, p}$ We immediately conclude that $\texttt{blame}\, p \longrightarrow_{\mathsf{C}}^{*} \texttt{blame}\, p$.

Case $\dfrac{\texttt{blame}\, p \approx M_1' \qquad |\texttt{id}_A|^{\mathsf{CS}} = s}{\texttt{blame}\, p \approx M_1'\langle s\rangle}$

By the induction hypothesis, we have $M_1' \longrightarrow_{\mathsf{S}}^{*} \texttt{blame}\, p$. So we conclude via the following diagram.

$$\begin{array}{ccc} \texttt{blame}\, p & \sim\!\sim\!\sim & M_1'\langle s\rangle \\ \Big\| & & \Big\downarrow{\scriptstyle *} \\ & & (\texttt{blame}\, p)\langle s\rangle \\ & & \Big\downarrow \\ \texttt{blame}\, p & \sim\!\sim\!\sim & \texttt{blame}\, p \end{array}$$

Part 6.

$M' = \texttt{blame}\, p$. We need to show that $M \approx \texttt{blame}\, p$. We proceed by cases on $M' \approx \texttt{blame}\, p$, but there is just one case.

Case $\dfrac{}{\texttt{blame}\, p \approx \texttt{blame}\, p}$

□

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

D Translation is bisimilar

Here we sketch the proof of Proposition 20.

Proposition 20. $M \approx |M|^{\mathsf{CS}}$.

Proof (Sketch). By induction on M. The only non-trivial case is for $M\langle c\rangle$ where we need to apply rules (i) and (ii) to establish $\approx$. In all other cases, the congruence rules are sufficient. □

E Proof of Proposition 10

Lemma 38. *For all* $c : A \Longrightarrow B$ *the sequence* $Z = |c|^{\mathsf{CB}}$ *is* admissible *as it has the form* $Z = [A_1 \overset{p_1}{\Longrightarrow} A_2, \cdots, A_m \overset{p_m}{\Longrightarrow} A_{m+1}]$ *where* $A_i \sim A_{i+1}$ *and* $A = A_1$ *and* $B = A_{m+1}$.

Proof This result follows by straightforward induction on c.

Case $\mathtt{id}_A : A \Longrightarrow A$. Immediate because $|\mathtt{id}_A|^{\mathsf{CB}} = [\,]$.

Case $G! : G \Longrightarrow \star$. We have $|G!|^{\mathsf{CB}} = [G \overset{\bullet}{\Longrightarrow} \star]$, which is admissible.

Case $G?^p : \star \overset{p}{\Longrightarrow} G$. We have $|G?^p|^{\mathsf{CB}} = [\star \overset{p}{\Longrightarrow} G]$, which is admissible.

Case $c \to d : A \to B \Longrightarrow A' \to B'$ where $c : A' \Longrightarrow A$ and $d : B \Longrightarrow B'$. We have $|c \to d|^{\mathsf{CB}} = \overline{(|c|^{\mathsf{CB}} \to B)} +\!\!+ (A' \to |d|^{\mathsf{CB}})$. As $|c|^{\mathsf{CB}}$ is admissible from A' to A by induction, we find that $(|c|^{\mathsf{CB}} \to B)$ is admissible from $A \to B$ to $A' \to B$. As $|d|^{\mathsf{CB}}$ is admissible from B to B' by induction, we find that $(A' \to |d|^{\mathsf{CB}})$ is admissible from $A' \to B$ to $A' \to B'$. Hence, their concatenation is admissible from $A \to B$ to $A' \to B'$ as required.

Case $c\,;d : A \Longrightarrow C$ where $c : A \Longrightarrow B$ and $d : B \Longrightarrow C$. Immediate by the induction hypotheses.

Case $\bot^{GpH} : A \Longrightarrow B$. Immediate because the sequence is admissible from A to B by construction. □

Proof of Proposition 10 We prove a slightly more general statement where Item 2 is an equivalence.

1. $\Gamma \vdash_{\mathsf{C}} M' : A$ implies $\Gamma \vdash_{\mathsf{B}} |M'|^{\mathsf{CB}} : A$.
2. $M'\ \mathsf{safe}_{\mathsf{C}}\ q$ if and only if $|M'|^{\mathsf{CB}}\ \mathsf{safe}_{\mathsf{B}}\ q$.

To start off, we define a syntactic operation $M \div Z$ to apply a admissible list of casts Z to a $\lambda\mathsf{B}$ expression M.

$$M \div [\,] = M \qquad\qquad M \div ([A \overset{p}{\Longrightarrow} B] +\!\!+ Z) = (M : A \overset{p}{\Longrightarrow} B) \div Z$$

It is easy to see that $M \div (Z_1 +\!\!+ Z_2) = (M \div Z_1) \div Z_2$.

Left to right, item 1. The interesting case is proving $\Gamma \vdash_{\mathsf{C}} M'\langle c\rangle : B$ implies $\Gamma \vdash_{\mathsf{B}} |M'\langle c\rangle|^{\mathsf{CB}} : B$.

Inversion of coercion application yields $\Gamma \vdash_{\mathsf{C}} M' : A$ and $c : A \Longrightarrow B$. Induction yields $\Gamma \vdash_{\mathsf{B}} |M'|^{\mathsf{CB}} : A$ and it remains to show that $|c|^{\mathsf{CB}} = Z$ such that applying the casts in Z to $|M'|^{\mathsf{CB}}$ has type B.

By Lemma 38 we know that $|c|^{\mathsf{CB}} = Z$ is admissible from A to B.

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press

By easy induction on $Z = |c|^{\mathsf{CB}}$ we obtain that for all $\Gamma \vdash_{\mathsf{B}} M : A$ and $c : A \Longrightarrow B$ we have $\Gamma \vdash_{\mathsf{B}} (M \div |c|^{\mathsf{CB}}) : B$.

Iff, item 2. Again the interesting case is proving $M'\langle c\rangle\, \mathsf{safe}_{\mathsf{C}}\, q$ if and only if $|M'\langle c\rangle|^{\mathsf{CB}}\, \mathsf{safe}_{\mathsf{B}}\, q$, which boils down to proving $\langle c\rangle\, \mathsf{safe}_{\mathsf{C}}\, q$ if and only if $|\langle c\rangle|^{\mathsf{CB}}\, \mathsf{safe}_{\mathsf{B}}\, q$ by induction on c. For a sequence of casts Z, we say that $Z\, \mathsf{safe}_{\mathsf{B}}\, q$ if $A \overset{p}{\Longrightarrow} B\, \mathsf{safe}_{\mathsf{B}}\, q$, for all $A \overset{p}{\Longrightarrow} B \in Z$.

Case $\mathtt{id}_A : A \Longrightarrow A$. Immediate.

Case $G\mathbf{!} : G \Longrightarrow \star$, which is safe for q. We have $|G\mathbf{!}|^{\mathsf{CB}} = [G \overset{\bullet}{\Longrightarrow} \star]$ and $G \overset{\bullet}{\Longrightarrow} \star\, \mathsf{safe}_{\mathsf{B}}\, q$.

Case $G?^p : \star \overset{p}{\Longrightarrow} G$. We have $|G?^p|^{\mathsf{CB}} = [\star \overset{p}{\Longrightarrow} G]$. It holds that $G?^p\, \mathsf{safe}_{\mathsf{C}}\, q$ iff $p \neq q$. As $\star <:^- G$, we have that $\star \overset{p}{\Longrightarrow} G\, \mathsf{safe}_{\mathsf{B}}\, q$ iff $p \neq q$.

Case $c \to d : A \to B \Longrightarrow A' \to B'$. We have $|c \to d|^{\mathsf{CB}} = \overline{(|c|^{\mathsf{CB}} \to B)} +\!\!+ (A' \to |d|^{\mathsf{CB}})$. It holds that $c \to d\, \mathsf{safe}_{\mathsf{C}}\, q$ iff $c\, \mathsf{safe}_{C}\, q$ and $d\, \mathsf{safe}_{\mathsf{C}}\, q$. By induction, we have $|c|^{\mathsf{CB}}\, \mathsf{safe}_{B}\, q$ and $|d|^{\mathsf{CB}}\, \mathsf{safe}_{B}\, q$. This is equivalent to $(|c|^{\mathsf{CB}} \to B)\, \mathsf{safe}_{\mathsf{B}}\, \overline{q}$ and $(A' \to |d|^{\mathsf{CB}})\, \mathsf{safe}_{\mathsf{B}}\, q$, where the first conjunct is equivalent to $\overline{(|c|^{\mathsf{CB}} \to B)}\, \mathsf{safe}_{\mathsf{B}}\, \overline{\overline{q}} = q$.

Case $c\,;d : A \Longrightarrow C$. Immediate by the induction hypotheses.

Case $\bot^{GpH} : A \Longrightarrow B$. We have $\bot^{GpH}\, \mathsf{safe}_{C}\, q$ iff $p \neq q$. For $|\bot^{GpH}_{A \Longrightarrow B}|^{\mathsf{CB}} = [A \overset{\bullet}{\Longrightarrow} G, G \overset{\bullet}{\Longrightarrow} \star, \star \overset{p}{\Longrightarrow} H, H \overset{\bullet}{\Longrightarrow} \star, \star \overset{\bullet}{\Longrightarrow} B]$, we find that $\star <:^- H$ hence $\star \overset{p}{\Longrightarrow} H\, \mathsf{safe}_{B}\, q$ iff $p \neq q$ and the other casts are trivially safe for q. □

https://doi.org/10.1017/S0956796821000101 Published online by Cambridge University Press