Graduality and Parametricity: Together Again for the First Time

MAX S. NEW, Northeastern University, USA
DUSTIN JAMNER, Northeastern University, USA
AMAL AHMED, Northeastern University, USA

Parametric polymorphism and gradual typing have proven to be a difficult combination, with no language yet produced that satisfies the fundamental theorems of each: parametricity and graduality. Notably, Toro, Labrada, and Tanter (POPL 2019) conjecture that for any gradual extension of System F that uses dynamic type generation, graduality and parametricity are "simply incompatible". However, we argue that it is not graduality and parametricity that are incompatible per se, but instead that combining the syntax of System F with dynamic type generation as in previous work necessitates type-directed computation, which we show has been a common source of graduality and parametricity violations in previous work.

We then show that by modifying the syntax of universal and existential types to make the type name generation explicit, we remove the need for type-directed computation, and get a language that satisfies both graduality and parametricity theorems. The language has a simple runtime semantics, which can be explained by translation to a statically typed language where the dynamic type is interpreted as a dynamically extensible sum type. Far from being in conflict, we show that the parametricity theorem follows as a direct corollary of a relational interpretation of the graduality property.

Additional Key Words and Phrases: gradual typing, graduality, polymorphism, parametricity, logical relation

1 INTRODUCTION

Gradually typed languages support freely mixing statically typed and dynamically code within a single language and enable a transition from dynamic to static typing [Siek and Taha 2006; Tobin-Hochstadt and Felleisen 2006, 2008]. They allow for stable, typed libraries to be used by ephemeral dynamically typed scripts with no manual programming overhead, streamlining a commonplace pattern in systems software. Furthermore, when some of these dynamically typed scripts inevitably become feature-rich software, static types can be gradually added to help with optimization, refactoring, type-based IDEs and documentation.

Gradually typed languages in the tradition of Siek and Taha [2006] are based on the presence of a dynamic type, written ?, which is the type of dynamically typed code and is treated specially by the type checker. For instance, if f is a statically typed function with type $\mathbb{I} \to \mathbb{B}$—where $\mathbb{I}$ and $\mathbb{B}$ represent integer and boolean types, respectively—and x is a dynamically typed input, then the application $f\,x$ is allowed by the static type checker because it is "plausible" that x will actually satisfy the type $\mathbb{I}$ at runtime. But note here that since f has type $\mathbb{I} \to \mathbb{B}$, it was written with the expectation that it should only be applied to integers and may, for instance, use arithmetic operations on its argument. In a *sound* gradually typed language, this type information should be

Authors' addresses: Max S. New, Khoury College of Computer Sciences, Northeastern University, USA, maxnew@ccs.neu.edu; Dustin Jamner, Khoury College of Computer Sciences, Northeastern University, USA, jamner.d@husky.neu.edu; Amal Ahmed, Khoury College of Computer Sciences, Northeastern University, USA, amal@ccs.neu.edu.

2475-1421/2020/1-ART46
https://doi.org/10.1145/3371114

reliable: the programmer and compiler should be able to refactor or optimize the function f based on its type, which says it will only be used on values of type $\mathbb{I}$. In order to ensure that this expectation is met at runtime, the application $f\,x$ is elaborated to a core language called a *cast calculus* where a cast is inserted and the application becomes $f(\langle \mathbb{I} \Leftarrow ? \rangle x)$. If at runtime x is a value that is incompatible with the type $\mathbb{I}$, such as a function value, then the cast will error and signal that the input failed to meet the function's type. While this means that the gradual language admits runtime errors, it ensures the soundness of the type for programmer reasoning and compiler optimization.

When designing the semantics of a gradual language, we must consider not just how programs *run*, but how their behavior *changes* throughout the development process. Specifically, a gradual language should ensure a smooth transition from dynamic to static typing, which is formalized in two properties called the *static* and *dynamic gradual guarantee* [Siek et al. 2015]. The static gradual guarantee states that making types more precise in a program makes it less likely that the program type-checks. Our focus in this paper is on the dynamic gradual guarantee, also called *graduality* [New and Ahmed 2018]. The graduality theorem provides a formalization for the intuition that making types more precise should not impact the *partial* correctness of the program itself. Specifically, it says that if the types in a program are made more precise, then either the more precise program errors, or exhibits the same behavior as before. This means that a programmer can add types to a portion of their program and know that the program as a whole still operates the same way, unless a new dynamic error is raised, in which case there is a flaw either in the code or in the new annotation that was introduced.

Languages can fail to satisfy the graduality theorem for a variety of reasons but a common culprit is *type-directed computation*. Whenever a form in a gradual language has behavior that is defined by inspection of the *type* of an argument, rather than by its *behavior*, there is a potential for a graduality violation, because the computation must be ensured to be *monotone* in the type. For instance, the Grace language supports a construct the designers call "structural type tests". That is, it includes a form $M\ \texttt{is}\ A$ that checks if M has type A at runtime. Boyland [2014] show that care must be taken in designing the semantics of this construct if A is allowed to be an arbitrary type. For instance, it might seem reasonable to say that $(\lambda x : ?.x)\ \texttt{is}\ \mathbb{I} \to \mathbb{I}$ should run to false because the function has type $? \to ?$. However, if we increase the precision of the types by changing the annotation, we get $(\lambda x : \mathbb{I}.x)\ \texttt{is}\ \mathbb{I} \to \mathbb{I}$ which should clearly evaluate to true, violating the graduality principle. In such a system, we can't think of types as just properties whose precision can be tuned up or down: we also need to understand how changing the type might influence our use of type tests at runtime.

Gradual typing researchers have designed languages that support reasoning principles enabled by a variety of advanced static features—such as objects [Siek and Taha 2007; Takikawa et al. 2012], refinement types [Lehmann and Tanter 2017], union and intersection types [Castagna and Lanvin 2017], typestates [Wolff et al. 2011], effect tracking [Bañados Schwerter et al. 2014], subtyping [Garcia et al. 2016], ownership [Sergey and Clarke 2012], session types [Igarashi et al. 2017b], and secure information flow [Disney and Flanagan 2011; Fennell and Thiemann 2013; Toro et al. 2018]. As these typing features become more complicated, the behavior of casts can become sophisticated as well, and the graduality principle is a way of ensuring that these sophisticated mechanisms stay within programmer expectations.

1.1 Polymorphism and Runtime Sealing

Parametric polymorphism, in the form of universal and existential types, allows for abstraction over types within a program. Universal types, written $\forall X.A$, allow for the definition of functions that can be used at many different types. Dually, existential types provide a simple model of a module system. A value of type $\exists X.A$ can be thought of as a module that exports a newly defined type X and then a value A that may include X that gives the interface to the type. Languages with parametric polymorphism provide very strong reasoning principles regarding data abstraction, formalized by the relational parametricity theorem [Reynolds 1983].

The relational parametricity theorem captures the idea that an abstract type is truly opaque to its users: for instance, a consumer of a value of existential type $\exists X.A$ can only interact with X values using the capabilities provided by the interface type A. This allows programmers to use existential types to model abstract data types [Mitchell and Plotkin 1985]. For instance, the existential type $\exists X.X \times (X \to X) \times (X \to \mathbb{I})$ represents the type of an abstract functional counter. The X represents the state, the first component of the tuple is the initial state, the second component is an increment function, and the final component reads out an observable integer value from the state. One obvious example implementation would use $\mathbb{I}$ for X, 0 as the initial state, addition by 1 as the increment, and the identity function as the read-out. In a language with proper data abstraction, we should be able to guarantee that with this implementation, the read-out function should only ever produce positive numbers, because even though the type $\mathbb{I}$ allows for negative numbers, the interface only enables the construction of positive numbers. This pattern of reasoning naturally generalizes to sophisticated data structure invariants such as balanced trees, sorted lists, etc.

Polymorphic languages can fail to satisfy the parametricity theorem for a variety of reasons but one common culprit is *type-directed computation* on abstract types. For instance in Java, values of a generic type T can be cast to an arbitrary object type. If the type T happens to be instantiated with the same type as the cast, then all information about the value will be revealed, and data abstraction is entirely lost. The problem is that the behavior of this runtime type-cast is directed by the type of the input: at runtime the input must carry some information indicating its type so that this cast can be performed. A similar problem arises when naïvely combining gradual typing with polymorphism, as we will see in §2.

While parametric polymorphism ensures data abstraction by means of a static type discipline, *dynamic sealing* provides a means of ensuring data abstraction even in a dynamically typed language. To protect abstract data from exposure, a fresh "key" is generated and implementation code must "seal" any abstract values before sending them to untrusted parties, "unsealing" them when they are passed into the exposed interface. For instance, we can ensure data abstraction for an untyped abstract functional counter by generating a fresh key σ, and producing a tuple where the first component is a 0 sealed with σ, and the increment and read-out function unseal their inputs and the increment function seals its output appropriately. If this is the only way the seal σ is used in the program, then the abstraction is ensured. While the programmer receives less support from the static type checker, this runtime sealing mechanism gives much of the same abstraction benefits.

One ongoing research area has been to satisfactorily combine the static typing discipline of parametric polymorphism with the runtime mechanism of dynamic sealing in a gradually typed language [Ahmed et al. 2011, 2017; Igarashi et al. 2017a; Ina and Igarashi 2011; Toro et al. 2019; Xie et al. 2018]. However, no such language design so far proposed has satisfied both of the desired fundamental theorems: graduality for gradual typing and relational parametricity for parametric polymorphism. Recent work by Toro et al. [2019] claims to prove that graduality and parametricity are inherently incompatible, which backed by analogous difficulties for secure information flow [Toro et al. 2018] has led to the impression that the graduality property is incompatible with parametric reasoning. This would be the *wrong conclusion to draw*, for the following two reasons. First, the claimed proof has a narrow applicability. It is based on the definition of their logical relation, which we show in §2.3 does not capture a standard notion of parametricity. Second, and more significantly, we should be careful not to conclude that graduality and parametricity are incompatible properties, and that language designs must choose one. In this paper, we reframe the problem: both are desirable, and should be demanded of any gradual or parametric language. The failure of graduality and parametricity in previous work can be interpreted not as an indictment of these properties, but rather points us to reconsider the combination of System F's syntax with runtime semantics based on dynamic sealing. In this paper, we will show that graduality and parametricity are not in conflict *per se*, by showing that by modifying System F's syntax to make the sealing visible, both properties are achieved. Far from being in opposition to each other, both graduality and parametricity can be proven using a *single* logical relation theorem (§6).

1.2 Overview

We summarize the contributions of this work as follows

- We identify type-directed computation as the common cause of graduality and parametricity violations in previous work on gradual polymorphism.
- We show that certain polymorphic programs in Toro et al. [2019]'s language GSF exhibit non-parametric behavior.
- We present a new surface language PolyG$^\nu$ that supports a novel form of universal and existential types where the creation of fresh types is exposed in a controlled way. The semantics of PolyG$^\nu$ is similar to previous gradual parametric languages, but the explicit type creation and sealing eliminates the need for type-directed computation.
- We elaborate PolyG$^\nu$ into an explicit cast calculus PolyC$^\nu$. We then give a translation from PolyC$^\nu$ into a typed target language, CBPV$_{\text{OSum}}$, essentially call-by-push-value with polymorphism and an extensible sum type.
- We develop a novel logical relation that proves both graduality and parametricity for PolyG$^\nu$. Thus, we show that parametricity and graduality are compatible, and we strengthen the connection alluded to by New and Ahmed [2018] that graduality and parametricity are analogous properties.

Complete typing rules, definitions, and proofs are in the technical appendix [New et al. 2020].

2 GRADUALITY AND PARAMETRICITY, FRIENDS OR ENEMIES?

Next, we review the issues in constructing a polymorphic gradual language that satisfies parametricity and graduality that have arisen in previous work. We see in each case that the common obstacle to parametricity and graduality is the presence of type-directed computation. This motivates our own language design, which obviates the need for type-directed computation by making dynamic sealing explicit in code.

2.1 "Naïve" Attempt

Before considering any dynamic sealing mechanisms, let's see why the most obvious combination of polymorphism with gradual typing produces a language that does not maintain data abstraction. Consider a polymorphic function of type $\forall X.X \to \mathbb{B}$. In a language satisfying relational parametricity, we know that the function must treat its input as having abstract type X and so this input cannot have any influence on what value is returned. However, in a gradually typed language, any value can be *cast* using type ascriptions, such as in the function $\Lambda X.\lambda x : X.(x :: ?) :: \mathbb{B}$. Here :: represents a type ascription. In a gradually typed language, a term M of type A can be ascribed a type B if it is "plausible" that an A is a B. This is typically formalized using a type consistency relation $\sim$ or more generally consistent subtyping relation $\lesssim$, but in either case, it is always plausible that an A is a ? and vice-versa, so in effect a value of any type can be cast to any other by taking a detour through the dynamic type. These ascriptions would then be elaborated to casts producing the term $\Lambda X.\lambda x : X.\langle \mathbb{B} \Leftarrow ?\rangle\langle ? \Leftarrow X\rangle x$ If this function is applied to any value that is not compatible with

$\mathbb{B}$, then the function will error, but if passed a boolean, the natural substitution-based semantics would result in the value being completely revealed:

$$(\Lambda X.\lambda x : X.\langle \mathbb{B} \Leftarrow ?\rangle\langle ? \Leftarrow X\rangle x)[\mathbb{B}]\mathtt{true} \mapsto^* \langle \mathbb{B} \Leftarrow ?\rangle\langle ? \Leftarrow \mathbb{B}\rangle\mathtt{true} \mapsto^* \mathtt{true}$$

The root-cause of this parametricity violation is that we allow casts like $\langle ? \Leftarrow X\rangle$ whose behavior depends on how X is instantiated. To construct a gradual language with strong data abstraction we must somehow avoid the dependency of $\langle ? \Leftarrow X\rangle$ on X. One option, is to ban casts like $\langle ? \Leftarrow X\rangle$ altogether. Syntactically, this means changing the notion of plausibility to say that ascribing a term of type X with the dynamic type ? is not allowed. This is possible using the system presented by Igarashi et al. [2017a] if you only allow Λs that use the "static" label. This is compatible with parametricity and graduality, but is somewhat against the spirit of gradual typing, where typically all programs could be written as dynamically typed programs, and dynamically typed functions can be used on values of any type. An alternative is to use dynamic sealing to allow casts like $\langle ? \Leftarrow X\rangle$, but ensure that their behavior does not depend on *how X is instantiated*.

2.2 Type-directed Sealing

In sealing-based gradual parametric languages like λB[Ahmed et al. 2011, 2017], we ensure that casts of abstract type do not depend on their instantiation by adding a layer of indirection. Instead of the usual β rule for polymorphic functions

$$(\Lambda X.M)[A] \mapsto M[A/X],$$

in λB, we dynamically generate a fresh type α and pass that in for X. This first of all means the runtime state must include a store of fresh types, written Σ. When reducing a type application, we generate a fresh type α and instantiate the function with this new type

$$\Sigma; (\Lambda X.M)[A] \mapsto \Sigma, \alpha := A; M[\alpha/X]$$

In this case, we interpret α as being a new tag on the dynamic type that tags values of type A but is different from all previously used tags. The casts involving α are treated like a new base type, incompatible with all existing types. However, if we look at the resulting term, it is not well-typed: if the polymorphic function has type $\forall X.B$, then $M[\alpha/X]$ has type $B[\alpha/X]$, but the context of this term expects it to be of type $B[A/X]$. To paper over this difference, λB wraps the substitution with a *type-directed coercion*, distinct from casts, that mediates between the two types:

$$\Sigma; (\Lambda X.M)[A] \mapsto \Sigma, \alpha := A; M[\alpha/X] : B[\alpha/X] \stackrel{+\alpha}{\Longrightarrow} B[A/X]$$

This type-directed coercion $M[\alpha/X] : B[\alpha/X] \stackrel{+\alpha}{\Longrightarrow} B[A/X]$ is the part of the system that performs the actual sealing and unsealing, and is defined by recursion on the type B. The $+\alpha$ indicates that we are *unsealing* values in positive positions and sealing at negative positions. For instance if $B = X \times \mathbb{B}$, and $X = \mathbb{B}$, then on a pair $(\mathtt{seal}_\alpha\mathtt{true}, \mathtt{false})$ the coercion will unseal the sealed boolean on the left and leave the boolean on the right alone. If B is of function type, the definition will involve the dual coercion using $-\alpha$, which *seals* at positive positions. So for instance applying the polymorphic identity function will reduce as follows

$$\begin{array}{ll}
\Sigma; (\Lambda X.\lambda x : X.x)[\mathbb{B}]\mathtt{true} & \mapsto \Sigma, \alpha := \mathbb{B}; (\lambda x : \alpha.x : \alpha \to \alpha \stackrel{+\alpha}{\Longrightarrow} \mathbb{B} \to \mathbb{B})\mathtt{true} \\
\mapsto \Sigma, \alpha := \mathbb{B}; (\lambda x : \alpha.x)(\mathtt{true} : X \stackrel{-\alpha}{\Longrightarrow} \alpha) : \alpha \stackrel{+\alpha}{\Longrightarrow} X & \mapsto \Sigma, \alpha := \mathbb{B}; (\lambda x : \alpha.x)(\mathtt{seal}_\alpha\mathtt{true}) : \alpha \stackrel{+\alpha}{\Longrightarrow} X \\
\mapsto \Sigma, \alpha := \mathbb{B}; \mathtt{seal}_\alpha\mathtt{true} : \alpha \stackrel{+\alpha}{\Longrightarrow} X & \mapsto \mathtt{true}
\end{array}$$

While this achieves the goal of maintaining data abstraction, it unfortunately violates graduality, as first pointed out by Igarashi et al. [2017a]. The reason is that the coercion is a type-directed computation, this time directed by the type $\forall X.B$ of the polymorphic function, whose behavior observably differs at type X from its behavior at type ?. Specifically, a coercion $M : X \overset{-\alpha}{\Longrightarrow} \alpha$ results in sealing the result of M, whereas if X is replaced by dynamic, then $M : ? \overset{-\alpha}{\Longrightarrow} \alpha$ is an identity function. An explicit counter-example is given by modifying the identity function to include an explicit annotation. The term $M_1 = (\Lambda X.\lambda x : X.x :: X)[\mathbb{B}]\mathtt{true}$ reduces by generating a seal α, sealing the input `true` with α, then unsealing it, finally producing `true`. On the other hand, if the type of the input were dynamic rather than X, we would get a term $M_2 = (\Lambda X.\lambda x : ?.(x :: X))[\mathbb{B}]\mathtt{true}$. In this case, the input is *not* sealed by the implementation, and the ascription of X results in a failed cast since $\mathbb{B}$ is incompatible with α. The only difference between the two terms is a type annotation, meaning that $M_1 \sqsubseteq M_2$ in the term precision ordering (M_1 is more precise than M_2), and so the graduality theorem states that if M_1 does not error, it should behave the same as M_2, but in this case M_2 errors while M_1 does not. The problem here is that the type of the polymorphic function determines whether to seal or unseal the inputs and outputs, but graduality says that the behavior of the dynamic type must align with both abstract types X (indicating sealing/unsealing) and concrete types like $\mathbb{B}$ (indicating no sealing/unsealing). These demands are contradictory since dynamic code would have to simultaneously be opaque until unsealing and available to interact with. So we see that the attempt to remove the type-directed casts which break parametricity by using dynamic sealing led to the need for a type-directed coercion which breaks graduality.

2.3 To Seal, or not to Seal

The language GSF was introduced by Toro et al. [2019] to address several criticisms of the type system and semantics of λB. We agree with the criticisms of the type system and so we will focus on the semantic differences. GSF by design has the same violation of graduality as λB, but has different behavior when using casts.

One motivating example for GSF is what happens when casting the polymorphic identity function to have a dynamically typed output: $(((\Lambda X.\lambda x : X.x) :: \forall X.X \to ?)[\mathbb{I}]1) + 2$. In λB, the input 1 is sealed as dictated by the type, but the dynamically typed output is not unsealed when it is returned from the function, resulting in an error when we try to add it. Ahmed et al. [2011] argue that it should be a free theorem that the behavior of a function of type $\forall X.X \to ?$ should be independent of its argument: it always errors, diverges or it always returns the same dynamic value, based on the intuition that the dynamic type ? does not syntactically contain the free variable X, and that this free theorem holds in System F. This reasoning is suspect since at runtime, the dynamic type does include a case for the freshly allocated type X, so intuitively we should consider ? to include X (and any other abstract types in scope).

Toro et al. [2019] argue on the other hand that intuitively the identity function was written with the intention of having a sealed input that is returned and then unsealed, and so casting the program to be more dynamic should result in the same behavior and so the program should succeed. The function application runs to the equivalent of $\langle ? \Leftarrow \mathbb{I}\rangle 1$ which is then cast to $\mathbb{I}$ and added to 2, resulting in the number 3. The mechanism for achieving this semantics is a system of runtime *evidence*, based on the *Abstracting Gradual Typing* (AGT) framework [Garcia et al. 2016]. An intuition for the behavior is that the sealing is still type-directed, but rather than being directed by the *static* type of the function being instantiated, it is based on the most precise type that the function has had. So here because the function was originally of type $\forall X.X \to X$, the sealing behavior is given by that type.

However, while we agree that the analysis in Ahmed et al. [2011] is incomplete, the behavior in GSF is inherently non-parametric, because the polymorphic program produces values with *different* dynamic type tags based on what the input *type* is. As a user of this function, we should be able to replace the instantiating type $\mathbb{I}$ with $\mathbb{B}$ and give any boolean input and get related behavior at the type ?, but in the program $(((\Lambda X.\lambda x : X.x) :: \forall X.X \to ?)[\mathbb{B}]\mathtt{true}) + 2$ the function application reduces to $\langle ? \Leftarrow \mathbb{B}\rangle\mathtt{true}$ which errors when cast to $\mathbb{I}$. Intuitively, this behavior is not parametric because the first program places an $\mathbb{I}$ tag on its input, and the second places a $\mathbb{B}$ tag on its input.

The non-parametricity is clearer if we look at a program of type $\forall X.? \to \mathbb{B}$ and consider the following function, a constant function with abstract input type cast to have dynamic input:

$$\mathrm{const} = (\Lambda X.\lambda x : X.\mathtt{true}) :: \forall X.? \to \mathbb{B}$$

X now has no effect on static typing, so both $\mathrm{const}[\mathbb{I}]3$ and $\mathrm{const}[\mathbb{B}]$ are well-typed. However, since the sealing behavior is actually determined by the type $\forall X.X \to \mathbb{B}$, the program will try to seal its input after downcasting it to whatever type X is instantiated at. So the first program casts $\langle \mathbb{I} \Leftarrow ?\rangle\langle ? \Leftarrow \mathbb{I}\rangle 3$, which succeeds and returns $\mathtt{true}$, while the second program performs the cast $\langle \mathbb{B} \Leftarrow ?\rangle\langle ? \Leftarrow \mathbb{I}\rangle 3$ which fails. In effect, we have implemented a polymorphic function that for any type X, is a recognizer of dynamically typed values for that type, returning $\mathtt{true}$ if the input matches X and erroring otherwise. Any implementation of this behavior would clearly require passing of some syntactic representation of types at runtime.

Formally, the GSF language does not satisfy the following *defining principle of relational parametricity*, as found in standard axiomatizations of parametricity such as Dunphy [2002]; Ma and Reynolds [1991]; Plotkin and Abadi [1993]. In a parametric language, the user of a term M of a polymorphic function type $\forall X.A \to B$ should be guaranteed that M will behave uniformly when instantiated multiple times. Specifically, a programmer should be able to instantiate M with two different types B_1, B_2 and choose any relation $R \in Rel[B_1, B_2]$ (where the notion of relation depends on the type of effects present), and be ensured that if they supply related inputs to the functions, they will get related outputs. Formally, for a Kripke-style relation, the following principle should hold:

$$\frac{M : \forall X.A \to B \qquad R \in \mathrm{Rel}[B_1, B_2] \qquad (w, V_1, V_2) \in \mathcal{V}[\![A]\!]\rho[X \mapsto R]}{(w, M[B_1]V_1, M[B_2]V_2) \in \mathcal{E}[\![B]\!]\rho[X \mapsto R]}$$

Here w is a "world" that gives the invariants in the store and ρ is the relational interpretation of free variables. $\mathcal{V}[\![\cdot]\!]$ and $\mathcal{E}[\![\cdot]\!]$ are value and expression relations formalizing an approximation ordering on values and expressions respectively, and $X \mapsto R$ means that the relational interpretation of X is given by R.

Toro et al. [2019] use an unusual logical relation for their language based on a similar relation in Ahmed et al. [2017], so there is no direct analogue of the relational mapping $X \mapsto R$. Instead, the application extends the world with the association of α to R and the interpretation sends X to α. However, we can show that this parametricity principle is violated by *any* ρ we pick for the term const above, using the definition of $\mathcal{E}[\![\cdot]\!]$ given in [Toro et al. 2019][1]. Instantiating the lemma would give us that $(w, \mathrm{const}[\mathbb{I}]3, \mathrm{const}[\mathbb{B}]3) \in \mathcal{E}[\![\mathbb{B}]\!]\rho$ since $(w, 3, 3) \in \mathcal{V}[\![?]\!]\rho$ for any ρ. The definition of $\mathcal{E}[\![\mathbb{B}]\!]\rho$ then says (again for any ρ) that it should be the case that since $\mathrm{const}[\mathbb{I}]3$ runs to a value, it should also be the case that $\mathrm{const}[\mathbb{B}]3$ runs to a value as well, but in actuality it errors, and so this parametricity principle must be false.

How can the above parametricity principle be false when Toro et al. [2019] prove a parametricity theorem for GSF? We have not found a flaw in their *proof*, but rather a mismatch between their *theorem* statement and the expected meaning of parametricity. The definition of $\mathcal{V}[\![\forall X.A]\!]$ in Toro et al. [2019] is not the usual interpretation, but rather is an adaptation of a non-standard definition used in Ahmed et al. [2017]. Neither of their definitions imply the above principle, so we argue that neither paper provides a satisfying proof of parametricity. With GSF, we see that the above behavior violates some expected parametric reasoning, using the definition of $\mathcal{V}[\![?]\!]$ given in Toro

[1]they use slightly different notation, but we use notation that matches the logical relation we present later

et al. [2019]. With λB, we know of no counterexample to the above principle, and we conjecture that it would satisfy a more standard formulation of parametricity.

It is worth noting that the presence of effects—such as nontermination, mutable state, control effects—requires different formulations of the logical relation that defines parametricity. However, those logical relations capture parametricity in that they always formalize *uniformity of behavior* across different type instantiations. For instance, for a language that supports nontermination, the logical relation for parametricity ensures that two different instantiations have the same termination behavior: either both diverge, or they both terminate with related values. Because of this, the presence of effects usually leads to weaker free theorems—in pure System F all inhabitants of $\forall X.X \to X$ are equivalent to the identity function, but in System F with non-termination, every inhabitant is either the identity or *always* errors. Though the free theorems are weaker, parametricity still ensures *uniformity* of behavior. As our counterexample above (const[$\mathbb{I}$]3 vs. const[$\mathbb{B}$]3) illustrates, GSF is non-parametric since it does not ensure uniform behavior. However, since the difference in behavior was between error and termination, it is possible that GSF satisfies a property that could be called "partial parametricity" (or parametricity modulo errors) that weakens the notion of uniformity of behavior: either one side errors or we get related behaviors. However, it is not clear to us how to formulate the logical relation for the dynamic type to prove this. We show how this weakened reasoning in the presence of ? compares to reasoning in our language PolyG$^{\nu}$ in §6.4.

Our counter-example crucially uses the dynamic type, and we conjecture that when the dynamic type does not appear under a quantifier, that the usual parametric reasoning should hold in GSF. This would mean that in GSF once polymorphic functions become "fully static", they support parametric reasoning, but we argue that it should be the goal of gradual typing to support type-based reasoning even in the presence of dynamic typing, since migration from dynamic to static is a gradual process, possibly taking a long time or never being fully completed.

2.4 Resolution: Explicit Sealing

Summarizing the above examples, we see that

(1) The naïve semantics leads to type-directed casts at abstract types, violating parametricity.
(2) λB's type-directed sealing violates graduality because of the ambiguity of whether or not the dynamic type indicates sealing/unsealing or not.
(3) GSF's variant of type-directed sealing based on the most precise type violates graduality and parametricity because the polymorphic function gets to determine which dynamically typed values are sealed (i.e. abstract) and which are not.

We see that in each case, the use of a type-directed computational step leads to a violation of graduality or parametricity. The GSF semantics makes the type-directed sealing of λB more flexible by using the runtime evidence attached to the polymorphic function rather than the type at the instantiation point, but unfortunately this makes it impossible for the continuation to reason about which dynamically typed values it passes will be treated as abstract or concrete. This analysis motivates our own language design PolyG$^{\nu}$, where

(1) We depart from the syntax of System F.
(2) Sealing/unsealing of values is explicit and programmable, rather than implicit and type-directed.
(3) The party that *instantiates* an abstract type is the party that determines which values are sealed and unsealed. For existential types, this is the package (i.e., the module) and dually for universal types it is the continuation of the instantiation.

The dynamic semantics of PolyG$^\nu$ are similar to λB without the type-directed coercions, removing the obstacle to proving the graduality theorem. By allowing user-programmable sealing and unsealing, more complicated forms of sealing and unsealing are possible: for instance, we can seal every prime number element of a list, which would require a very rich type system to express using type-directed sealing! We conjecture that the language is strictly more expressive than λB in the sense of Felleisen [1990]: λB should be translatable into PolyG$^\nu$ in a way that simulates its operational semantics. Because the sealing is performed by the instantiating party rather than the abstracting party, the expressivity of PolyG$^\nu$ is incomparable to GSF.

3 PolyG$^\nu$: A GRADUAL LANGUAGE WITH POLYMORPHISM AND SEALING

Next, we present a our gradual language, PolyG$^\nu$, that supports a variant of existential and universal quantification while satisfying parametricity and graduality. The language has some unusual features, so we start with an extended example to illustrate what programs look like, and then in § 3.2 introduce the formal syntax and typing rules.

3.1 PolyG$^\nu$ Informally

Let's consider an example of existential types, since they are simpler than universal types in PolyG$^\nu$. In a typed, non-gradual language, we can define an abstract "flipper" type, $\mathtt{FLIP} = \exists X.X \times (X \to X) \times (X \to \mathbb{B})$. The first element is the initial state, the second is a "toggle" function and the last element reads out the value as a concrete boolean.

Then we could create an instance of this abstract flipper using booleans as the carrier type X and negation as the toggle function $\mathtt{pack}(\mathbb{B}, (\mathtt{true}, (\mathtt{NOT}, \mathtt{ID})))\ \mathtt{as}\ \mathtt{FLIP}$. Note that we must explicitly mark the existential package with a type annotation, because otherwise we wouldn't be able to tell which occurrences of $\mathbb{B}$ should be hidden and which should be exposed. With different type annotations, the same package could be given types $\exists X.\mathbb{B} \times (\mathbb{B} \to \mathbb{B}) \times (\mathbb{B} \to \mathbb{B})$ or $\exists X.X \times (X \to X) \times (X \to X)$.

The PolyG$^\nu$ language existential type works differently in a few ways. We write $\exists^\nu$ rather than $\exists$ to emphasize that we are only quantifying over fresh types, and not arbitrary types. The equivalent of the above existential package would be written as

$$\mathtt{pack}^\nu(X \cong \mathbb{B}, (\mathtt{seal}_X \mathtt{true}, ((\lambda x : X.\mathtt{seal}_X(\mathtt{NOT}(\mathtt{unseal}_X x))), (\lambda x : X.\mathtt{unseal}_X x))) : \mathtt{FLIP}$$

The first thing to notice is that rather than just providing a type $\mathbb{B}$ to instantiate the existential, we write a declaration $X \cong \mathbb{B}$. The X here is a *binding position* and the body of the package is typed under the assumption that $X \cong \mathbb{B}$. Then, rather than *substituting* $\mathbb{B}$ for X when typing the body of the package, the type checker checks that the body has type $X \times ((X \to X) \times (X \to \mathbb{B}))$ under the assumption that $X \cong \mathbb{B}$:

$$X \cong \mathbb{B} \vdash (\mathtt{seal}_X \mathtt{true}, ((\lambda x : X.\mathtt{seal}_X(\mathtt{NOT}(\mathtt{unseal}_X x))), (\lambda x : X.\mathtt{unseal}_X x))) : X \times ((X \to X) \times (X \to \mathbb{B}))$$

Crucially, $X \cong \mathbb{B}$ is a *weaker* assumption than $X = \mathbb{B}$. In particular, there are no *implicit* casts from X to $\mathbb{B}$ or vice-versa, but the programmer can *explicitly* "seal" $\mathbb{B}$ values to be X using the form $\mathtt{seal}_X M$, which is only well-typed under the assumption that $X \cong A$ for some A consistent with $\mathbb{B}$. We also get a corresponding unseal form $\mathtt{unseal}_X M$, and the runtime semantics in § 4.4 defines these to be a bijection. At runtime, X will be a freshly generated type with its own tag on the dynamic type. An interesting side-effect of making the difference between X and $\mathbb{B}$ explicit in the term is that existential packages do not require type annotations to resolve any ambiguities. For instance, unlike in the typed case, the gradual package above could *not* be ascribed the type $\exists^\nu X.\mathbb{B} \times ((\mathbb{B} \to \mathbb{B}) \times (\mathbb{B} \to \mathbb{B}))$ because the functions explicitly take X values, and not $\mathbb{B}$ values.

The corresponding elimination form for $\exists^\nu$ is a standard unpack: $\mathtt{unpack}\ (X, x) = M; N$, where the continuation for the unpack is typed with just X and x added to the context, it doesn't know that $X \cong A$ for any particular A. We call this ordinary type variable assumption an *abstract type variable*, whereas the new assumption $X \cong A$ is a *known type variable* which acts more like a *type definition* than an abstract type. At runtime, when an existential is unpacked, a fresh type X is created that is isomorphic to A but whose behavior with respect to casts is different.

While explicit sealing and unsealing might seem burdensome to the programmer, note that this is directly analogous to a common pattern in Haskell, where modules are used in combination with `newtype` to create a datatype that at runtime is represented in the same way as an existing type, but for type-checking purposes is considered distinct. We give an analogous Haskell module as follows:

```
module Flipper(State, start, toggle, toBool) where
newtype State = Seal { unseal :: Bool }
start :: State
start = Seal True
inc :: State -> State
inc s = Seal (not (unseal s)
toBool :: State -> Bool
toBool = unseal
```

Then a different module that imports `Flipper` is analogous to an unpack, as its only interface to the `State` type is through the functions provided.

We also add universal quantification to the language, using the duality between universals and existentials as a guide. Again we write the type differently, as $\forall^\nu X.A$. In an ordinary polymorphic language, we would write the type of the identity function as $\forall X.X \to X$ and implement it using a Λ form: $\Lambda X.\lambda x : X.x$. The elimination form passes in a type for X. For instance applying the identity function to a boolean would be written as $\mathtt{ID}\,[\mathbb{B}]\ \mathtt{true}$. And a free theorem tells us that this term must either diverge, error, or return `true`.

The introduction form Λ is dual to the unpack form, and correspondingly looks the same as the ordinary Λ, for example in the identity function $\mathtt{ID}^\nu = \Lambda^\nu X.\lambda x : X.x : \forall^\nu X.X \to X$. The body of the Λ^ν is typed with an abstract type variable X in scope. The elimination form of type application is dual to the *pack* form, and so similarly introduces a *known* type variable assumption. Instantiating the identity function as above would be written as $\mathtt{unseal}_X(\mathtt{ID}^\nu\{X \cong \mathbb{B}\}(\mathtt{seal}_X\mathtt{true})) : \mathbb{B}$. which introduces a known type variable $X \cong \mathbb{B}$ into the context. Rather than the resulting type being $\mathbb{B} \to \mathbb{B}$, it is $X \to X$ with the assumption $X \cong \mathbb{B}$. Then the argument to the function must be explicitly sealed as an X to be passed to the function. The output of the function is also of type X and so must be explicitly *unsealed* to get a boolean out. However, there is something quite unusual about this term: the $X \cong \mathbb{B}$ binding site is not binding X in a *subterm* of the application, but rather into the *context*: the argument is sealed, and the continuation is performing an unseal! These bindings in $\forall^\nu$ instantiations follow this "inside-out" structure and complicate the typing rules: every term in the language "exports" known type variable bindings that go outwards in addition to the other typing assumptions coming inwards from the context. While unusual, they are intuitively justified by the duality with existentials: we can think of the *continuation* for an instantiation of a $\forall^\nu$ as being analogous to the *body* of the existential package.

To get an understanding of how PolyG$^\nu$ compares to λB and GSF and why it avoids their violation of graduality, let's consider how we might write the examples from the previous section. In PolyG$^\nu$, if we apply a function of type $\forall X.X \to X$, we have to mark explicitly that the input is sealed, and furthermore if we want to use the output as a boolean, we must *unseal* the output:

$$\mathtt{unseal}_X((\Lambda X.\lambda x : X.x :: X)\{X \cong \mathbb{B}\}(\mathtt{seal}_X\mathtt{true})) \mapsto^* \mathtt{true}$$

Then if we change the type of the input from X to ? the explicit sealing and unsealing remain, so even though the input is dynamically typed it will still be a sealed boolean, and the program exhibits the same behavior:

types	A	$::=$	$? \mid X \mid \mathbb{B} \mid A \times A \mid A \to A \mid \exists^\nu X.A \mid \forall^\nu X.A$
Ground types	G	$::=$	$X \mid \mathbb{B} \mid ? \times ? \mid ? \to ? \mid \exists^\nu X.? \mid \forall^\nu X.?$
terms	M	$::=$	$x \mid M :: A \mid \mathtt{seal}_X M \mid \mathtt{unseal}_X M \mid \mathtt{is}(G)?\ M \mid \mathtt{true} \mid \mathtt{false}$
		$\mid$	$\mathtt{if}\ M\ \mathtt{then}\ M\ \mathtt{else}\ M \mid (M, M) \mid \mathtt{let}\ (x, x) = M; M$
		$\mid$	$M\ M \mid \lambda x : A.M \mid \mathtt{pack}^\nu(X \cong A, M) \mid \mathtt{unpack}\ (X, x) = M; N$
		$\mid$	$\Lambda^\nu X.M \mid M\{X \cong A\} \mid \mathtt{let}\ x = M; M$
environment	Γ	$::=$	$\cdot \mid \Gamma, x : A \mid \Gamma, X \mid \Gamma, X \cong A$

Fig. 1. PolyG$^\nu$ Syntax

$$\mathtt{unseal}_X((\Lambda X.\lambda x : ?.x :: X)\{X \cong \mathbb{B}\}(\mathtt{seal}_X \mathtt{true})) \mapsto^* \mathtt{true}$$

If we remove the seal of the input, then the cast to X in the function will fail, giving us the behavior of λB/GSF:

$$\mathtt{unseal}_X((\Lambda X.\lambda x : ?.x :: X)\{X \cong \mathbb{B}\}\mathtt{true}) \mapsto^* \mho$$

but crucially this involved changing the term, not just the type, so the graduality theorem does not tell us that the programs should have related behavior.

Next, let's consider the parametricity violation from GSF. When we instantiate the constant function, we need to decide if the argument is sealed or not. We get the behavior of GSF when we instantiate with $\mathbb{I}$ and seal the input 3:

$$\mathrm{const}\{X \cong \mathbb{I}\}\mathtt{seal}_X 3 \mapsto^* \mathtt{true}$$

However, if we try to write the analogous program with $\mathbb{B}$: instead of $\mathbb{I}$ $\mathrm{const}\{X \cong \mathbb{B}\}\mathtt{seal}_X 3$ then the program is not well typed because $X \cong \mathbb{B}$ and 3 has type $\mathbb{I}$ which is not compatible. We can replicate the outcome of the GSF program by *not* sealing the 3:

$$\mathrm{const}\{X \cong \mathbb{B}\}3 \mapsto^* \mho$$

But this is not a parametricity violation because the 3 here will be embedded at the dynamic type with the $\mathbb{I}$ tag, whereas above the 3 was tagged with the X tag, which is not related.

3.2 PolyG$^\nu$ Formal Syntax and Semantics

Figure 1 presents the syntax of the surface language types, terms and environments. Most of the language is a typical gradual functional language, using ? as the dynamic type, and including type ascription $M :: A$. The unusual aspects of the language are the $\mathtt{seal}_X M$ and $\mathtt{unseal}_X M$ forms and the "fresh" existential $\exists^\nu X.A$ and universal $\forall^\nu X.A$. Note also the non-standard environments Γ, which include ordinary typing assumptions $x : A$, abstract type variable assumptions X and known type variable assumptions $X \cong A$. For simplicity, we assume freshness of all type variable bindings, i.e. when we write Γ, X or $\Gamma, X \cong A$ that X does not occur in Γ.

The typing rules are presented in Figure 2. On a first pass, we suggest ignoring all shaded parts of the rules, which only concern the inside-out scoping needed for the $\forall^\nu X.A$ forms and would not be necessary if this type was removed. We follow the usual formulation of gradual surface languages in the style of [Siek and Taha 2006]: type checking is strict when checking compatibility of different connectives, but lax when the dynamic type is involved. The first $M :: B$ form is type-ascription, which is well formed when the types are *consistent* with each other, written $A \sim B$. We define this in the standard way in as being the least congruence relation including equality and rules making ? consistent with every type.

We include variable and let-binding rules, which are standard other than the shaded parts. Next, we include sealing $\mathtt{seal}_X M$ and unsealing $\mathtt{unseal}_X M$ forms. The sealing and unsealing forms are valid when the assumption $X \cong A$ is in the environment and give the programmer access to an explicit bijection between the types X and A. It is crucial for graduality to hold that this bijection is explicit and not implicit, because the behavior of casts involving X and A are very different. To

$$\frac{\Gamma \vdash M : A;\ \Gamma_o \qquad A \sim B}{\Gamma \vdash (M :: B) : B;\ \Gamma_o} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A;\ \cdot} \qquad \frac{\Gamma \vdash M : A;\ \Gamma_M \qquad \Gamma, \Gamma_M, x : A \vdash N : B;\ \Gamma_N}{\Gamma \vdash \mathtt{let}\ x = M; N : B;\ \Gamma_M, \Gamma_N}$$

$$\frac{\Gamma \vdash M : B;\ \Gamma_o \qquad X \cong A \in \Gamma, \Gamma_o \qquad B \sim A}{\Gamma \vdash \mathtt{seal}_X M : X;\ \Gamma_o} \qquad \frac{\Gamma \vdash M : B;\ \Gamma_o \qquad X \cong A \in \Gamma, \Gamma_o \qquad B \sim X}{\Gamma \vdash \mathtt{unseal}_X M : A;\ \Gamma_o}$$

$$\frac{\Gamma \vdash M : A;\ \Gamma_o \qquad \Gamma, \Gamma_o \vdash G}{\Gamma \vdash \mathtt{is}(G)?\ M : \mathbb{B};\ \Gamma_o} \qquad \Gamma \vdash \mathtt{true} : \mathbb{B};\ \cdot \qquad \Gamma \vdash \mathtt{false} : \mathbb{B};\ \cdot$$

$$\frac{\begin{array}{c}\Gamma \vdash M : A;\ \Gamma_M \qquad A \sim \mathbb{B}\\ \Gamma, \Gamma_M \vdash N_t : B_t;\ \Gamma_t \qquad \Gamma, \Gamma_M \vdash N_f : B_f;\ \Gamma_f\end{array}}{\Gamma \vdash \mathtt{if}\ M\ \mathtt{then}\ N_t\ \mathtt{else}\ N_f : B_t \sqcap B_f;\ \Gamma_M, \Gamma_t \cap \Gamma_f}$$

$$\frac{\Gamma \vdash M_1 : A_1;\ \Gamma_1 \qquad \Gamma, \Gamma_1 \vdash M_2 : A_2;\ \Gamma_2}{\Gamma \vdash (M_1, M_2) : A_1 \times A_2;\ \Gamma_1, \Gamma_2} \qquad \frac{\Gamma \vdash M : A;\ \Gamma_M \qquad \Gamma, \Gamma_M, x : \pi_1(A), y : \pi_2(A) \vdash N : B;\ \Gamma_N}{\Gamma \vdash \mathtt{let}\ (x, y)\ =\ M; N : B;\ \Gamma_M, \Gamma_N}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash M : B;\ \Gamma_o}{\Gamma \vdash \lambda x : A.M : A \to B;\ \cdot} \qquad \frac{\Gamma \vdash M : A;\ \Gamma_M \qquad \Gamma, \Gamma_M \vdash N : B;\ \Gamma_N \qquad \mathrm{dom}(A) \sim B}{\Gamma \vdash M\ N : \mathrm{cod}(A);\ \Gamma_M, \Gamma_N}$$

$$\frac{\Gamma, X \cong A \vdash M : B;\ \Gamma_o}{\Gamma \vdash \mathtt{pack}^\nu(X \cong A, M) : \exists^\nu X.B;\ \cdot} \qquad \frac{\begin{array}{c}\Gamma \vdash M : A;\ \Gamma_M\\ \Gamma, \Gamma_M, X, x : \mathrm{un}\exists^\nu(A) \vdash N : B;\ \Gamma_N \qquad \Gamma, \Gamma_M, \Gamma_N|_X \vdash B\end{array}}{\Gamma \vdash \mathtt{unpack}\ (X, x) = M; N : B;\ \Gamma_M, \Gamma_N|_X}$$

$$\frac{\Gamma, X \vdash M : A; \Gamma_o}{\Gamma \vdash \Lambda^\nu X.M : \forall^\nu X.A; \cdot} \qquad \frac{\Gamma \vdash M : A; \Gamma_M \qquad \Gamma, \Gamma_M \vdash B}{\Gamma \vdash M\{X \cong B\} : \mathrm{un}\forall^\nu(A); \Gamma_M, X \cong B}$$

$$? \sim A \qquad A \sim ? \qquad \mathbb{B} \sim \mathbb{B} \qquad X \sim X$$

$$\frac{A_i \sim B_i \qquad A_o \sim B_o}{A_i \to A_o \sim B_i \to B_o} \qquad \frac{A_1 \sim B_1 \qquad A_2 \sim B_2}{A_1 \times A_2 \sim B_1 \times B_2} \qquad \frac{A \sim B}{\exists^\nu X.A \sim \exists^\nu X.B} \qquad \frac{A \sim B}{\forall^\nu X.A \sim \forall^\nu X.B}$$

$$\begin{array}{rcl}
\mathrm{dom}(A \to B) & = & A\\
\mathrm{dom}(?) & = & ?\\
\mathrm{cod}(A \to B) & = & B\\
\mathrm{cod}(?) & = & ?\\
\pi_i(A_1 \times A_2) & = & A_i\\
\pi_i(?) & = & ?\\
\mathrm{un}\forall^\nu(\forall^\nu X.A) & = & A\\
\mathrm{un}\forall^\nu(?) & = & ?
\end{array}
\qquad
\begin{array}{rcl}
\mathrm{un}\exists^\nu(\exists^\nu X.A) & = & A\\
\mathrm{un}\exists^\nu(?) & = & ?
\end{array}$$

$$\begin{array}{ll}
\cdot|_{\Gamma'} = \cdot & \\
(X \cong A, \Gamma)|_{\Gamma'} = X \cong A, (\Gamma|_{\Gamma'}) & (\mathrm{FV}(A) \cap \Gamma' = \emptyset)\\
(X \cong A, \Gamma)|_{\Gamma'} = \Gamma|_{\Gamma', X} & (\mathrm{FV}(A) \cap \Gamma' \neq \emptyset)
\end{array}$$

Fig. 2. PolyG$^\nu$ Type System

show that it has no adverse effect on the calculus, we also include a form $\mathtt{is}(G)?\ M$ that checks at runtime whether M returns a value that is compatible with the ground type G. M can have any type in this case because it is always a safe operation, but the result is either trivially `true` or `false` unless M has type ?.

Next, we have booleans, whose values are `true` and `false`, and whose elimination form is an if-statement. The if-statement checks that the scrutinee has a type compatible with $\mathbb{B}$, and as in previous work uses gradual meet $B_t \sqcap B_f$ for the output type [Garcia and Cimini 2015]. Gradual meet is only partially defined, since this ensures that if the two sides have different (non-?) head connectives then type checking errors, in keeping with the philosophy of strict checking when precise types are used.

Next, we have pairs and functions, which are fairly standard. We use pattern-matching as the elimination form for pairs. To reduce the number of rules, we present the elimination forms in the style of Garcia and Cimini [2015], using partial functions π_i, dom, cod and later un$\forall^\nu$, un$\exists^\nu$ to extract the subformula from a type "up to ?". For the correct type this extracts the actual subformula, but for ? is defined to be ? and for other connectives is undefined. We define these at the bottom of the figure, where uncovered cases are undefined. Next, we have existentials, which are as described in §3.1.

Finally, we consider the shaded components of the judgment. The full form of the judgment is $\Gamma \vdash M : A; \Gamma_o$ where Γ_o is the list of bindings that are *generated by* M and exported outward. Note that the type A of M can use variables in Γ_o as well as variables in Γ. Also, while we write these as Γ_o, Γ_M, etc., they only contain sequences of known type variables, and never any abstract type variables or typing assumptions $x : A$. These bindings are generated in the $\forall^\nu$ elimination rule, where the instantiation $M\{X \cong B\}$ adds $X \cong B$ to the output context. Rules that produce delayed thunks—the function, existential and universal introduction rules—have bodies that generate bindings, but these are not exported because these bindings will only be generated at the point where the thunk is forced to evaluate. The rest of the rules work similarly to an effect system: for instance in the function application rule $M\ N$ the bindings generated in M are bound in N, and the application produces all of the bindings they generate, and similarly for product introduction. In the unpack form, care must be taken to make sure that the X from the unpack is not leaked in the output Γ_N, in addition to making sure the output type B does not mention X. Any known type variables that mention X are removed from the output context, using the *restriction* form $\Gamma_{\Gamma'}$ defined at the bottom of Figure 2. Finally, in the if form, each branch might export different known type variables, so the if statement as a whole only exports the intersection of the two branches, since these are the only ones guaranteed to be generated.

4 PolyC$^\nu$: CAST CALCULUS

As is standard in gradual languages, rather than giving the surface language an operational semantics directly, we define a *cast calculus* that makes explicit the casts that perform the dynamic type checking in gradual programs. We present the cast calculus syntax in Figure 3. The cast calculus syntax is almost the same as the surface syntax, though the typing is quite different. First, the type ascription form is removed, and several forms are added to replace it. Based on the analysis in [New and Ahmed 2018], we add two cast forms: an upcast $\langle A^\sqsubseteq \rangle\!\uparrow M$ and a downcast $\langle A^\sqsubseteq \rangle\!\downarrow M$, whereas most prior work includes a single cast form $\langle A \Leftarrow B \rangle$. The $A^\sqsubseteq$ used in the upcast and downcast forms here is a proof that $A_l \sqsubseteq A_r$ for some types A_l, A_r, i.e., that A_l is a more precise (less dynamic) type than A_r. This type precision definition is key to formalizing the graduality property, but previous work has shown that it is useful for formalizing the semantics of casts as well. We emphasize the structure of these proofs because the central semantic constructions of this work: the operational semantics of casts, the translation of casts into functions and finally our graduality logical relation are all naturally defined by recursion on these derivations.

4.1 PolyC$^\nu$ Type Precision

We present the definition of type precision in Figure 4. The judgment $\Gamma \vdash A^\sqsubseteq : A_l \sqsubseteq A_r$ is read as "using the variables in Γ", $A^\sqsubseteq$ proves that A_l is more precise/less dynamic than A_r. If you ignore the

$$
\begin{array}{rrcl}
\text{type names} & \alpha & ::= & \sigma \mid X \\
\text{types} & A, B & + ::= & \sigma \\
\text{ground types} & G & ::= & \alpha \mid \mathbb{B} \mid ? \times ? \mid ? \to ? \mid \exists^\nu X.? \mid \forall^\nu X.? \\
\text{precision derivations} & A^{\sqsubseteq}, B^{\sqsubseteq} & ::= & ? \mid \mathrm{tag}_G(A^{\sqsubseteq}) \mid \alpha \mid \mathbb{B} \mid A^{\sqsubseteq} \times A^{\sqsubseteq} \mid A^{\sqsubseteq} \to B^{\sqsubseteq} \\
& & \mid & \exists^\nu X.A^{\sqsubseteq} \mid \forall^\nu X.A^{\sqsubseteq} \\
\text{values} & V & ::= & \mathtt{seal}_\alpha V \mid \mathtt{true} \mid \mathtt{false} \mid x \mid (V, V) \mid \lambda(x : A).M \\
& & \mid & \Lambda^\nu X.M \mid \mathtt{inj}_G\ V \mid \langle A_1^{\sqsubseteq} \to A_2^{\sqsubseteq} \rangle\uparrow M \mid \langle A_1^{\sqsubseteq} \to A_2^{\sqsubseteq} \rangle\downarrow M \\
& & \mid & \langle \forall^\nu X.A^{\sqsubseteq} \rangle\uparrow M \mid \langle \forall^\nu X.A^{\sqsubseteq} \rangle\downarrow M \mid \mathtt{pack}^\nu(X \cong A', [\overline{A^{\sqsubseteq} \updownarrow}], M) \\
\text{expressions} & M, N & - ::= & (M :: A) \\
& & + ::= & \mho \mid \langle A^{\sqsubseteq} \rangle\uparrow M \mid \langle A^{\sqsubseteq} \rangle\downarrow M \mid \mathtt{hide}\ X \cong A; M \mid \mathtt{inj}_G\ M \\
& & \mid & \mathtt{pack}^\nu(X \cong A', [A^{\sqsubseteq} \updownarrow, \ldots], M) \mid \mathtt{seal}_\sigma M \mid \mathtt{unseal}_\sigma M \\
\text{Evaluation Context} & E & ::= & [] \mid (E, M) \mid (V, E) \mid E[A] \mid E\ M \mid V\ E \mid \mathtt{inj}_G\ E \\
& & \mid & \mathtt{if}\ E\ \mathtt{then}\ M\ \mathtt{else}\ M \mid \mathtt{let}\ (x, y) = E; M \mid \langle A^{\sqsubseteq} \rangle\uparrow E \\
& & \mid & \mathtt{unpack}\ (X, x) = E; M \mid \mathtt{seal}_\alpha E \mid \mathtt{unseal}_\alpha E \mid \langle A^{\sqsubseteq} \rangle\downarrow E
\end{array}
$$

Fig. 3. PolyC$^\nu$ Syntax

$$
\frac{\Gamma \vdash A^{\sqsubseteq} : A \sqsubseteq G}{\Gamma \vdash \mathrm{tag}_G(A^{\sqsubseteq}) : A \sqsubseteq ?}
\qquad
\Gamma \vdash ? : ? \sqsubseteq ?
\qquad
\Gamma \vdash \mathbb{B} : \mathbb{B} \sqsubseteq \mathbb{B}
\qquad
\frac{X \in \Gamma}{\Gamma \vdash X : X \sqsubseteq X}
$$

$$
\frac{\Gamma \vdash A_1^{\sqsubseteq} : A_{l1} \sqsubseteq A_{r1} \qquad \Gamma \vdash A_2^{\sqsubseteq} : A_{l2} \sqsubseteq A_{r2}}{\Gamma \vdash A_1^{\sqsubseteq} \times A_2^{\sqsubseteq} : A_{l1} \times A_{l2} \sqsubseteq A_{r1} \times A_{r2}}
\qquad
\frac{\Gamma \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r \qquad \Gamma \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma \vdash A^{\sqsubseteq} \to B^{\sqsubseteq} : A_l \to B_l \sqsubseteq A_r \to B_r}
$$

$$
\frac{\Gamma, X \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma \vdash \exists^\nu X.A^{\sqsubseteq} : \exists^\nu X.A_l \sqsubseteq \exists^\nu X.A_r}
\qquad
\frac{\Gamma, X \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma \vdash \forall^\nu X.A^{\sqsubseteq} : \forall^\nu X.A_l \sqsubseteq \forall^\nu X.A_r}
$$

Fig. 4. PolyC$^\nu$ Type Precision

precision derivations, our definition of type precision is a simple extension of the usual notion: type variables are only related to the dynamic type and themselves, and similarly for $\forall$ and $\exists$. Since we have quantifiers and type variables, we include a context Γ of known and abstract type variables. Crucially, even under the assumption that $X \cong A$, X and A are *unrelated* precision-wise unless A is ?. As before, $X \in \Gamma$ ranges over both known and abstract type variables. It is easy to see that precision reflexive and transitive, and that ? is the greatest element. Finally, ? is the least precise type, meaning for any type A there is a derivation that $A \sqsubseteq ?$. The precision notation is a natural extension of the syntax of types: with base types $?, \mathbb{B}$ serving as the proof of reflexivity at the type and constructors $\times, \to$, etc. serving as syntax for congruence proofs. It is important to note that while we give a syntax for derivations, there is at most *one* derivation $A^{\sqsubseteq}$ that proves any given $A_l \sqsubseteq A_r$. We prove these and several more lemmas about type precision in the appendix [New et al. 2020].

4.2 PolyC$^\nu$ Type System

The static type system for the cast calculus is given Figure 5. The cast calculus type system differs from the surface language in that all type checking is *strict* and precise. This manifests in two ways. First, the dynamic type is not considered implicitly compatible with other types. Instead, in the translation from PolyG$^\nu$ to PolyC$^\nu$, we insert casts wherever consistency is used in the judgment. Second, in the if rule, the branches must have the *same* type, and an upcast is inserted in the translation to make the two align. Finally, the outward scoping of known type variables is handled more explicitly. We add a new form $\mathtt{hide}\ X \cong A; M$ that delimits the scope of $X \cong A$ from going further outward, enforced by the side condition that $\Gamma, \Gamma_M \vdash \Gamma'_M$. Then in rules that include

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A; \cdot} \qquad \frac{\Gamma \vdash M : A_l; \Gamma_M \qquad \Gamma \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma \vdash \langle A^{\sqsubseteq} \rangle\!\!\upharpoonleft M : A_r; \Gamma_M} \qquad \frac{\Gamma \vdash M : A_r; \Gamma_M \qquad \Gamma \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma \vdash \langle A^{\sqsubseteq} \rangle\!\!\downharpoonleft M : A_l; \Gamma_M}$$

$$\frac{\Gamma \vdash M : A; \Gamma_M \qquad \Gamma, \Gamma_M, x : A \vdash N : B; \Gamma_N}{\Gamma \vdash \mathtt{let}\ x = M; N : B; \Gamma_M, \Gamma_N} \qquad \frac{\Gamma \vdash M : A; \Gamma_o \qquad X \cong A \in \Gamma, \Gamma_o}{\Gamma \vdash \mathtt{seal}_X M : X; \Gamma_o}$$

$$\frac{\Gamma \vdash M : X; \Gamma_o \qquad X \cong A \in \Gamma, \Gamma_o}{\Gamma \vdash \mathtt{unseal}_X M : A; \Gamma_o} \qquad \frac{\Gamma \vdash M : ?; \Gamma_o \qquad \Gamma \vdash G}{\Gamma \vdash \mathtt{is}(G)?\ M : \mathbb{B}; \Gamma_o} \qquad \Gamma \vdash \mathtt{true} : \mathbb{B}; \cdot \qquad \Gamma \vdash \mathtt{false} : \mathbb{B}; \cdot$$

$$\frac{\Gamma \vdash M : \mathbb{B}; \Gamma_M \qquad \Gamma, \Gamma_M \vdash N_t : B; \Gamma_N \qquad \Gamma, \Gamma_M \vdash N_f : B; \Gamma_N}{\Gamma \vdash \mathtt{if}\ M\ \mathtt{then}\ N_t\ \mathtt{else}\ N_f : B; \Gamma_M, \Gamma_N} \qquad \frac{\Gamma \vdash M_1 : A_1; \Gamma_1 \qquad \Gamma, \Gamma_1 \vdash M_2 : A_2; \Gamma_2}{\Gamma \vdash (M_1, M_2) : A_1 \times A_2; \Gamma_1, \Gamma_2}$$

$$\frac{\Gamma \vdash M : A_1 \times A_2; \Gamma_M \qquad \Gamma, \Gamma_M, x : A_1, y : A_2 \vdash N : B; \Gamma_N}{\Gamma \vdash \mathtt{let}\ (x, y) = M; N : B; \Gamma_M, \Gamma_N} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash M : B; \cdot}{\Gamma \vdash \lambda x : A.M : A \to B; \cdot}$$

$$\frac{\Gamma \vdash M : A \to B; \Gamma_M \qquad \Gamma, \Gamma_M \vdash N : A; \Gamma_N}{\Gamma \vdash M\ N : B; \Gamma_M, \Gamma_N} \qquad \frac{\Gamma, X \cong A \vdash M : B; \cdot}{\Gamma \vdash \mathsf{pack}^{\nu}(X \cong A, M) : \exists^{\nu} X.B; \cdot}$$

$$\frac{\Gamma \vdash M : \exists^{\nu} X.A; \Gamma_M \qquad \Gamma, \Gamma_M, X, x : A \vdash N : B; \Gamma_N \qquad \Gamma, \Gamma_M \vdash \Gamma_N \qquad \Gamma, \Gamma_M, \Gamma_N \vdash B}{\Gamma \vdash \mathsf{unpack}\ (X, x) = M; N : B; \Gamma_M, \Gamma_N}$$

$$\frac{\Gamma, X \vdash M : A; \cdot}{\Gamma \vdash \Lambda^{\nu} X.M : \forall^{\nu} X.A; \cdot} \qquad \frac{\Gamma \vdash M : \forall^{\nu} X.A; \Gamma_M \qquad \Gamma, \Gamma_M \vdash B}{\Gamma \vdash M\{X \cong B\} : A; \Gamma_M, X \cong B} \qquad \frac{\Gamma \vdash M : \Gamma_M, X \cong A, \Gamma'_M \qquad \Gamma, \Gamma_M \vdash \Gamma'_M}{\Gamma \vdash \mathtt{hide}\ X \cong A; M; \Gamma_M, \Gamma'_M}$$

Fig. 5. PolyC$^{\nu}$ Typing

delayed computations, i.e., values of function, existential and universal type, whereas in the surface language the delayed term could produce any names, now in PolyC$^{\nu}$, they must all be manually hidden. Similarly in the branches of an if statement, the two sides must have the same generated names, and hides must be used in the elaboration to make them align.

4.3 Elaboration from PolyG$^{\nu}$ to PolyC$^{\nu}$

We define the elaboration of PolyG$^{\nu}$ into the cast calculus PolyC$^{\nu}$ in Figure 6. Following [New and Ahmed 2018], an ascription is interpreted as a cast up to ? followed by a cast down to the ascribed type. Most of the elaboration is standard, with elimination forms being directly translated to the corresponding PolyC$^{\nu}$ form if the head connective is correct, and inserting a downcast if the elimination position has type ?. We formalize this using the metafunction $G \wr M$ defined towards the bottom of the figure. For the if case, in PolyC$^{\nu}$ the two branches of the if have to have the same output type and export the same names, so we downcast each branch, but also we hide any names not generated by both sides, using the metafunction $\mathtt{hide}\ \Gamma \subseteq \Gamma'; M$, defined at the bottom of the figure, which hides all names present in Γ' that are not in Γ. Finally, in the values that are thunks (pack, λ and Λ), the bodies of the thunks must not generate names in PolyC$^{\nu}$, so we hide names there as well.

4.4 PolyC$^{\nu}$ Dynamic Semantics

The dynamic semantics of PolyC$^{\nu}$, presented in Figure 7, extends traditional cast semantics with appropriate rules for our name-generating universals and existentials. The runtime state is a pair of a term M and a *case store* Σ. A case store Σ represents the set of cases allocated so far in the

$$
\begin{aligned}
(M :: B)^+ &= \langle B?^{\sqsubseteq}\rangle\!\Downarrow \langle A?^{\sqsubseteq}\rangle\!\Uparrow M^+ && \text{(where } M : A,\ A?^{\sqsubseteq} : A \sqsubseteq ?,\ B?^{\sqsubseteq} : B \sqsubseteq ?)\\
x^+ &= x\\
(\mathtt{let}\ x = M; N)^+ &= \mathtt{let}\ x = M^+; N^+\\
(\mathtt{seal}_X M)^+ &= \mathtt{seal}_X (M :: A)^+ && \text{(where } X \cong A)\\
(\mathtt{unseal}_X M)^+ &= \mathtt{unseal}_X (X \wr M)\\
(\mathtt{is}(G)?\ M)^+ &= \mathtt{is}(G)?\ (\langle A?^{\sqsubseteq}\rangle\!\Uparrow M) && \text{(where } M : A,\ A?^{\sqsubseteq} : A \sqsubseteq ?)\\
b^+ &= b && (b \in \{\mathtt{true}, \mathtt{false}\})\\
(\mathtt{if}\ M\ \mathtt{then}\ N_t\ \mathtt{else}\ N_f)^+ &= \mathtt{if}\ \mathbb{B} \wr M\ \mathtt{then}\ (\langle B_t^{\sqsubseteq}\rangle\!\Downarrow \mathtt{hide}\ \Gamma_t \subseteq \Gamma_t \cap \Gamma_f; N_t^+)\\
&\qquad \mathtt{else}\ (\langle B_f^{\sqsubseteq}\rangle\!\Downarrow \mathtt{hide}\ \Gamma_f \subseteq \Gamma_t \cap \Gamma_f; N_f^+)\\
&&& \text{(where } \Gamma \vdash \mathtt{if}\ M\ \mathtt{then}\ N_t\ \mathtt{else}\ N_f : B_t \sqcap B_f; \Gamma_M, \Gamma_t \cap \Gamma_f)\\
&&& \text{(and } B_t^{\sqsubseteq} : B_t \sqcap B_f \sqsubseteq B_t,\ B_f^{\sqsubseteq} : B_t \sqcap B_f \sqsubseteq B_f)\\
(M_1, M_2)^+ &= (M_1^+, M_2^+)\\
(\mathtt{let}\ (x, y)\ =\ M; N)^+ &= \mathtt{let}\ (x, y)\ =\ ? \times ? \wr M; N^+\\
(\lambda x : A.M)^+ &= \lambda x : A.\mathtt{hide}\ \Gamma_o \subseteq \Gamma_o; M^+ && \text{(where } \Gamma, x : A \vdash M : B; \Gamma_o)\\
(M\ N)^+ &= (? \to ? \wr M)\ (N :: \mathrm{dom}(A))^+ && \text{(where } M : A)\\
(\mathtt{pack}^\nu(X \cong A, M))^+ &= \mathtt{pack}^\nu(X \cong A, \mathtt{hide}\ \Gamma_o \subseteq \Gamma_o; M^+) && \text{(where } M : B; \Gamma_o)\\
(\mathtt{unpack}\ (X, x) = M; N)^+ &= \mathtt{unpack}\ (X, x) = \exists^\nu X.? \wr M; \mathtt{hide}\ \Gamma_N|_X \subseteq \Gamma_N; N^+\\
\Lambda^\nu X.M^+ &= \Lambda^\nu X.\mathtt{hide}\ \Gamma_o \subseteq \Gamma_o; M^+ && \text{(where } M : A; \Gamma_o)\\
M\{X \cong B\}^+ &= (\forall^\nu X.? \wr M)\{X \cong B\}\\
G \wr M &= \langle \mathrm{tag}_G(G)\rangle\!\Downarrow M^+ && \text{(when } M : ?)\\
G \wr M &= M^+ && \text{(otherwise)}\\
\mathtt{hide}\ \Gamma_s \subseteq (\Gamma_b, X \cong A); M &= \mathtt{hide}\ \Gamma_s \subseteq \Gamma_b; \mathtt{hide}\ X \cong A; M && (X \notin \Gamma_s)\\
\mathtt{hide}\ (\Gamma_s, X \cong A) \subseteq (\Gamma_b, X \cong A); M &= \mathtt{hide}\ \Gamma_s \subseteq \Gamma_b; M\\
\mathtt{hide}\ \cdot \subseteq \cdot; M &= M
\end{aligned}
$$

Fig. 6. Elaborating PolyG$^\nu$ to PolyC$^\nu$

program. Formally, a store Σ is just a pair of a number $\Sigma.n$ and a function $\Sigma.f : [n] \to \mathrm{Ty}$ where Ty is the set of all types and $[n] = \{m \in \mathbb{N} \mid m < n\}$ is from some prefix of natural numbers to types. All rules take configurations $\Sigma \triangleright M$ to configurations $\Sigma' \triangleright M'$. When the step does not change the store, we write $M \mapsto M'$ for brevity.

The first rule states that all non-trivial evaluation contexts propagate errors. Next, unsealing a seal gets out the underlying value, and $\mathtt{is}(G)?\ V$ literally checks if the tag of V is G. The hide form generates a fresh seal $\sigma : A$ and substitutes it into the continuation. The pack form steps to an intermediate state used for building up a stack of casts that will be used again in the existential cast rule. The unpack rule generates a fresh seal for the $X \cong A$ and then applies all of the accumulated casts to the body of the pack. Here we use $\Updownarrow$ to indicate one of $\Uparrow$ and $\Downarrow$. For $\forall^\nu$ instantiation, we do not need to generate the seal, because it must have already been generated by a hide form further up the term, so the rule is just a substitution. As is typical for a cast calculus, the remaining types have ordinary call-by-value β reduction and so we elide them.

The remaining rules give the behavior of casts. Other than the use of type precision derivations, the behavior of our casts is mostly standard: identity casts for $\mathbb{B}$, σ and ? are just the identity, and the product cast proceeds structurally. Function casts are values, and when applied to a value, the

$$
\begin{array}{lcl}
E[\mho] & \mapsto & \mho \text{ where } E \neq [\,] \\
E[\mathtt{unseal}_\sigma(\mathtt{seal}_\sigma V)] & \mapsto & E[V] \\
E[\mathtt{is}(G)?\ (\mathtt{inj}_G\ V)] & \mapsto & E[\mathtt{true}] \\
E[\mathtt{is}(G)?\ (\mathtt{inj}_H\ V)] & \mapsto & E[\mathtt{false}] \text{ where } G \neq H \\
\Sigma \triangleright E[\mathtt{hide}\ X \cong A; M] & \mapsto & \Sigma, \sigma : A \triangleright E[M[\sigma/X]] \\
\Sigma \triangleright E\left[\begin{array}{l}\mathsf{unpack}\ (X, x) = \mathsf{pack}^\nu(X \cong A', [\overline{A^{\sqsubseteq} \updownarrow}], M); \\ N\end{array}\right] & \mapsto & \Sigma, \sigma : A' \triangleright E\left[\begin{array}{l}\mathtt{let}\ x = \overline{\langle A^{\sqsubseteq}[\sigma/X]\rangle\updownarrow}\ M[\sigma/X]; \\ N[\sigma/X]\end{array}\right] \\
E\,[\mathsf{pack}^\nu(X \cong A, M)] & \mapsto & E[\mathsf{pack}^\nu(X \cong A', [\,], M)] \\
E[(\Lambda^\nu X.M)\{\sigma \cong A\}] & \mapsto & E[M[\sigma/X]] \\
E[\langle A^{\sqsubseteq}\rangle\updownarrow\ V] & \mapsto & E[V] \text{ where } A^{\sqsubseteq} \in \{\mathbb{B}, \sigma, ?\} \\
E[\langle A_1^{\sqsubseteq} \times A_2^{\sqsubseteq}\rangle\updownarrow\ (V_1, V_2)] & \mapsto & E[(\langle A_1^{\sqsubseteq}\rangle\updownarrow\ V_1, \langle A_2^{\sqsubseteq}\rangle\updownarrow\ V_2)] \\
E[(\langle A_1^{\sqsubseteq} \to A_2^{\sqsubseteq}\rangle\updownarrow\ V_1)\ V_2] & \mapsto & E[\langle A_2^{\sqsubseteq}\rangle\updownarrow\ (V_1\ \langle A_1^{\sqsubseteq}\rangle\updownarrow^-\ V_2)] \\
E[\langle \exists^\nu X.A^{\sqsubseteq}\rangle\updownarrow\ \mathsf{pack}^\nu(X \cong A', [A'^{\sqsubseteq} \updownarrow', \ldots], M)] & \mapsto & E[\mathsf{pack}^\nu(X \cong A', [A^{\sqsubseteq} \updownarrow, A'^{\sqsubseteq} \updownarrow', \ldots], M)] \\
E[(\langle \forall^\nu X.A^{\sqsubseteq}\rangle\updownarrow\ V)\{\sigma \cong A\}] & \mapsto & E[\langle A^{\sqsubseteq}[\sigma/X]\rangle\updownarrow\ (V\{\sigma \cong A\})] \\
E[\langle \mathrm{tag}_G(A^{\sqsubseteq})\rangle\uparrow\ V] & \mapsto & E[\mathtt{inj}_G\ \langle A^{\sqsubseteq}\rangle\uparrow\ V] \\
E[\langle \mathrm{tag}_G(A^{\sqsubseteq})\rangle\downarrow\ \mathtt{inj}_G\ V] & \mapsto & E[\langle A^{\sqsubseteq}\rangle\downarrow\ V] \\
E[\langle \mathrm{tag}_G(A^{\sqsubseteq})\rangle\downarrow\ \mathtt{inj}_H\ V] & \mapsto & \mho \text{ where } H \neq G
\end{array}
$$

Fig. 7. PolyC$^\nu$ Operational Semantics (fragment)

cast is performed on the output and the oppositely oriented case on the input. We use $\updownarrow^-$ to indicate the opposite arrow, so $\uparrow^- = \downarrow$ and $\downarrow^- = \uparrow$ to cut down the number of rules. Next, the $\forall^\nu$ casts are also values that reduce when the instantiating type is supplied. As with existentials, the freshly generated type σ is substituted for X in the precision derivation guiding the cast. Finally, the upcast case for $\mathrm{tag}_G(A^{\sqsubseteq})$ simply injects the result of upcasting with $A^{\sqsubseteq}$ into the dynamic type using the tag G. For the downcast case, the opposite is done if the input has the right tag, and otherwise a dynamic type error is raised.

In the appendix, we extend the typing to runtime terms which are typed with respect to a case-store and prove a standard type safety theorem for the language that terms either take a step or are values or errors [New et al. 2020].

5 TYPED INTERPRETATION OF THE CAST CALCULUS

In the previous section we developed a cast calculus with an operational semantics defining the behavior of the name generation and gradual type casts. However, this ad hoc design addition of new type connectives and inside-out scoping of $\forall^\nu$-instantiations make the cast calculus less than ideal for proving meta-theoretic properties of the system.

Instead of directly proving metatheoretic properties of the cast calculus, we give a *contract translation* of the cast calculus into a statically typed core language, translating the gradual type casts to ordinary terms in the typed language that raise errors. The key benefit of the typed language is that it does not have built-in notions of fresh existential and universal quantification. Instead, the type translation decomposes these features into the combination of *ordinary* existential and universal quantification combined with a somewhat well-studied programming feature: a dynamically extensible "open" sum type we call OSum. Finally, it gives a static type interpretation of the dynamic type: rather than being a finitary sum of a few statically fixed cases, the dynamic type is implemented as the open sum type which includes those types allocated at runtime.

5.1 Typed Metalanguage

We present the syntax of our typed language CBPV$_{\text{OSum}}$ in Figure 8, an extension of Levy's Call-by-push-value calculus [Levy 2003], which we use as a convenient metalanguage to extend with features

$$
\begin{array}{rrcl}
\text{value types} & A & ::= & X \mid \text{Case } A \mid \text{OSum} \mid A \times A \mid \mathbb{B} \mid \exists X.A \mid UB \\
\text{computation types} & B & ::= & A \to B \mid \forall X.B \mid FA \\
\text{values} & V & ::= & \sigma \mid \mathtt{inj}_V\ V \mid \mathtt{pack}(A, V)\ \mathtt{as}\ \exists X.A \mid x \mid (V, V) \\
& & \mid & \mathtt{true} \mid \mathtt{false} \mid \mathtt{thunk}\ M \\
\text{computations} & M & ::= & \mho \mid \mathtt{force}\ V \mid \mathtt{ret}\ V \mid x \leftarrow M; N \mid M\ V \mid \lambda x : A.M \\
& & \mid & \mathtt{newcase}_A\ x; M \mid \mathtt{match}\ V\ \mathtt{with}\ V\{\mathtt{inj}\ x.M \mid N\} \\
& & \mid & \mathtt{unpack}\ (X, x) = V; M \mid \mathtt{let}\ (x, x) = V; M \mid \Lambda X.M \mid M[A] \\
& & \mid & \mathtt{if}\ V\ \mathtt{then}\ M\ \mathtt{else}\ M \\
\text{stacks} & S & ::= & \bullet \mid S\ V \mid S\ [A] \mid x \leftarrow S; M \\
\text{value typing context} & \Gamma & ::= & \cdot \mid \Gamma, x : A \\
\text{type variable context} & \Delta & ::= & \cdot \mid \Delta, X
\end{array}
$$

Fig. 8. $\text{CBPV}_{\text{OSum}}$ Syntax

of interest. Call-by-push-value (CBPV) is a typed calculus with highly explicit evaluation order, providing similar benefits to continuation-passing style and A-normal form [Sabry and Felleisen 1992]. The main distinguishing features of CBPV are that values V and effectful computations M are distinct syntactic categories, with distinct types: value types A and computation types B. The two "shift" types U and F mediate between the two worlds. A value of type UB is a first-class "thUnk" of a computation of type B that can be forced, behaving as a B. A computation of type FA is a computation that can perform effects and return a value of type A, and whose elimination form is a monad-like *bind*. Notably while sums and (strict) tuples are value types, function types $A \to B$ are computations since a function interacts with its environment by receiving an argument. We include existentials as value type and universals as computation types, that in each case quantify over *value* types because we are using it as the target of a translation from a call-by-value language.

We furthermore extend CBPV with two new value types: OSum and Case A, which add an open sum type similar to the extensible exception types in ML, but with an expression-oriented interface more suitable to a core calculus. The open sum type OSum is initially empty, but can have new cases allocated at runtime. A value of Case A is a first class representative of a case of OSum. The introduction form $\mathtt{inj}_{V_c}\ V$ for OSum uses a case V_c : Case A to inject a value $V : A$ into OSum. The elimination form $\mathtt{match}\ V_o\ \mathtt{with}\ V_c\{\mathtt{inj}\ x.M \mid N\}$ for OSum is to use a V_c : Case A to do a pattern match on a value V_o : OSum. Since OSum is an *open* sum type, it is unknown what cases V_o might use, so the pattern-match has two branches: the one $\mathtt{inj}\ x.M$ binds the underlying value to $x : A$ and proceeds as M and the other is a catch-all case N in case V_o was not constructed using V_c. Finally, there is a form $\mathtt{newcase}_A\ x; M$ that allocates a fresh Case A, binds it to x and proceeds as M. In addition to the similarity to ML exception types, they are also similar to the dynamically typed sealing mechanism introduced in Sumii and Pierce [2004].

5.2 Static and Dynamic Semantics

We show a fragment of the typing rules for $\text{CBPV}_{\text{OSum}}$ in Figure 9. There are two judgments corresponding to the two syntactic categories of terms: $\Delta; \Gamma \vdash V : A$ for typing a value and $\Delta; \Gamma \vdash M : A$ for typing a computation. Δ is the environment of type variables and Γ is the environment for term variables. Unlike in PolyG^{ν} and PolyC^{ν}, these are completely standard, and there is no concept of a known type variable.

First, an error $\mho$ is a computation and can be given any type. Variables are standard and the OSum/Case forms are as described above. Existentials are a value form and are standard as in CBPV using ordinary substitution in the pack form. In all of the value type elimination rules, the discriminee is restricted to be a *value*. A computation $M : B$ can be thunked to form a value $\mathtt{thunk}\ M : UB$, which can be forced to run as a computation. Like the existentials, the universal quantification

$$\frac{}{\Delta;\Gamma \vdash \mho : B} \qquad \frac{}{\Delta;\Gamma, x : A, \Gamma' \vdash x : A} \qquad \frac{\Delta;\Gamma \vdash V_c : \text{Case } A \qquad \Delta;\Gamma \vdash V : A}{\Delta;\Gamma \vdash \mathtt{inj}_{V_c}\ V : \text{OSum}}$$

$$\frac{\Delta;\Gamma \vdash V : \text{OSum} \qquad \Delta;\Gamma \vdash V_c : \text{Case } A \qquad \Delta;\Gamma, x : A \vdash M : B \qquad \Delta;\Gamma \vdash N : B}{\Delta;\Gamma \vdash \mathtt{match}\ V\ \mathtt{with}\ V_c\{\mathtt{inj}\ x.M \mid N\} : B}$$

$$\frac{\Delta \vdash A \qquad \Delta;\Gamma, x : \text{Case } A \vdash M : B}{\Delta;\Gamma \vdash \mathtt{newcase}_A\ x; M : B} \qquad \frac{\Delta;\Gamma \vdash V : A[A'/X]}{\Delta;\Gamma \vdash \mathtt{pack}(A', V)\ \mathtt{as}\ \exists X.A : \exists X.A}$$

$$\frac{\Delta;\Gamma \vdash V : \exists X.A \qquad \Delta \vdash B \qquad \Delta, X;\Gamma, x : A \vdash M : B}{\Delta;\Gamma \vdash \mathtt{unpack}\ (X, x) = V; M : B} \qquad \frac{\Delta;\Gamma \vdash M : B}{\Delta;\Gamma \vdash \mathtt{thunk}\ M : UB} \qquad \frac{\Delta;\Gamma \vdash V : UB}{\Delta;\Gamma \vdash \mathtt{force}\ V : B}$$

$$\frac{\Delta, X;\Gamma \vdash M : B}{\Delta;\Gamma \vdash \Lambda X.M : \forall X.B} \qquad \frac{\Delta;\Gamma \vdash M : \forall X.B \qquad \Delta \vdash A}{\Delta;\Gamma \vdash M[A] : B[A/X]} \qquad \frac{\Delta;\Gamma \vdash V : A}{\Delta;\Gamma \vdash \mathtt{ret}\ V : FA} \qquad \frac{\Delta;\Gamma \vdash M : FA \qquad \Delta;\Gamma \vdash N : B}{\Delta;\Gamma \vdash x \leftarrow M; N : B}$$

Fig. 9. $\text{CBPV}_{\text{OSum}}$ Type System (fragment)

$$\begin{aligned}
S[\mho] &\mapsto^0 \mho \\
\Sigma \triangleright S[\mathtt{newcase}_A\ x; M] &\mapsto^0 \Sigma, \sigma : A \triangleright S[M[\sigma/x]] \\
S[\mathtt{match}\ \mathtt{inj}_\sigma\ V\ \mathtt{with}\ \sigma\{\mathtt{inj}\ x.M \mid N\}] &\mapsto^1 S[M[V/x]] \\
S[\mathtt{match}\ \mathtt{inj}_{\sigma_1}\ V\ \mathtt{with}\ \sigma_2\{\mathtt{inj}\ x.M \mid N\}] &\mapsto^1 S[N] \qquad \text{(where } \sigma_1 \neq \sigma_2\text{)} \\
S[\mathtt{force}\ (\mathtt{thunk}\ M)] &\mapsto^0 S[M] \\
S[\mathtt{unpack}\ (X, x) = \mathtt{pack}(A, V); M] &\mapsto^0 S[M[A/X, V/x]] \\
S[(\Lambda X.M)[A]] &\mapsto^0 S[M[A/X]] \\
S[x \leftarrow \mathtt{ret}\ V; N] &\mapsto^0 S[N[V/x]]
\end{aligned}$$

Fig. 10. $\text{CBPV}_{\text{OSum}}$ Operational Semantics (fragment)

type is standard, using substitution in the elimination form. Finally, the introduction form for FA returns a value $V : A$, and the elimination form is a bind, similar to a monadic semantics of effects, except that the continuation can have any computation type B, rather than restricted to FA.

A representative fragment of the operational semantics is given in Figure 10, the full semantics are in the appendix [New et al. 2020]. S represents a *stack*, the CBPV analogue of an evaluation context, defined in Figure 8. Here Σ is like the Σ in PolyC^ν, but maps to *value types*. The semantics is standard, other than the fact that we assign a count to each step of either 0 or 1. The only steps that count for 1 are those that introduce non-termination to the language, which is used later as a technical device in our logical relation in §6.

5.3 Translation

Next, we present the "contract translation" of PolyC^ν into $\text{CBPV}_{\text{OSum}}$. This translation can be thought of as an alternate semantics to the operational semantics for PolyC^ν, but with a tight correspondence given in §5.4. Since $\text{CBPV}_{\text{OSum}}$ is a typed language that uses ordinary features like functions, quantification and an open sum type, this gives a simple explanation of the semantics of PolyC^ν in terms of fairly standard language features.

In the left side of Figure 11, we present the type translation from PolyC^ν to $\text{CBPV}_{\text{OSum}}$. Since PolyC^ν is a call-by-value language, types are translated to $\text{CBPV}_{\text{OSum}}$ *value* types. Booleans and

$$
\begin{aligned}
[\![\Sigma;\Gamma \vdash ?]\!] &= \text{OSum} \\
[\![\Sigma;\Gamma \vdash X]\!] &= X \quad (\text{where } X \in \Gamma) \\
[\![\Sigma;\Gamma \vdash X]\!] &= [\![A]\!] \quad (\text{where } X \cong A \in \Gamma) \\
[\![\Sigma;\Gamma \vdash \sigma]\!] &= [\![A]\!] \quad (\text{where } \sigma : A \in \Sigma) \\
[\![\Sigma;\Gamma \vdash \mathbb{B}]\!] &= \mathbb{B} \\
[\![\Sigma;\Gamma \vdash A \to B]\!] &= U([\![\Sigma;\Gamma \vdash A]\!] \to F[\![\Sigma;\Gamma \vdash B]\!]) \\
[\![\Sigma;\Gamma \vdash A_1 \times A_2]\!] &= [\![\Sigma;\Gamma \vdash A_1]\!] \times [\![\Sigma;\Gamma \vdash A_2]\!] \\
[\![\Sigma;\Gamma \vdash \exists^\nu X.A]\!] &= \exists X.U(\text{Case } X \to F[\![\Sigma;\Gamma, X \vdash A]\!]) \\
[\![\Sigma;\Gamma \vdash \forall^\nu X.A]\!] &= U(\forall X.\text{Case } X \to F[\![\Sigma;\Gamma, X \vdash A]\!])
\end{aligned}
$$

$$
\begin{aligned}
[\![\Sigma \vdash \cdot]\!] &= \cdot;\cdot \\
[\![\Sigma \vdash \Gamma, x : A]\!] &= \Delta';\Gamma', x : [\![\Sigma;\Gamma \vdash A]\!] \\
&\qquad (\text{where } [\![\Sigma \vdash \Gamma]\!] = \Delta';\Gamma') \\
[\![\Sigma \vdash \Gamma, X]\!] &= \Delta', X;\Gamma', c_X : \text{Case } X \\
&\qquad (\text{where } [\![\Sigma \vdash \Gamma]\!] = \Delta';\Gamma') \\
[\![\Sigma \vdash \Gamma, X \cong A]\!] &= \Delta';\Gamma', c_X : \text{Case } [\![\Sigma;\Gamma \vdash A]\!] \\
&\qquad (\text{where } [\![\Sigma \vdash \Gamma]\!] = \Delta';\Gamma')
\end{aligned}
$$

Fig. 11. PolyC$^\nu$ type and environment translation

pairs are translated directly, and the function type is given the standard CBPV translation for call-by-value functions, $U([\![A]\!] \to F[\![B]\!])$: a call-by-value function is a thunked computation of a function that takes an $[\![A]\!]$ as input and may return a $[\![B]\!]$ as output. The dynamic type ? is interpreted as the open sum type. The meaning of a type variable depends on the context: if it is an abstract type variable, it is translated to a type variable, but if it is a known type variable $X \cong A$, it is translated to $[\![A]\!]$! That is, at runtime, values of a known type variable X are just values of the type isomorphic to X, and as we will see later, sealing and unsealing are no-ops. Similarly, a runtime type tag σ is translated to the type that the corresponding case maps to. These are inductively well-defined because Σ stays constant in the type translation and Γ only adds abstract type variables.

The final two cases are the most revealing. First the fresh universal quantifier, $\forall^\nu X.A$, translates to *not just* a thunk of a universally quantified computation, but also takes in a Case X as input. The body of a Λ will then use that Case X in order to interpret casts involving X. This is precisely why parametricity is more complex for our source language: if it were translated to just $U(\forall X.F[\![A]\!])$, then parametricity would follow directly by parametricity for CBPV$_{\text{OSum}}$, but the Case X represents additional information that the function is being passed that potentially provides information about the type X. It is only because code translated from PolyC$^\nu$ always generates a fresh case that this extra input is benign. The fresh existential $\exists^\nu X.A$ is translated to a real existential of a thunk that expects a Case X and returns a $[\![A]\!]$. Note that while the quantification is the dual of the $\forall^\nu$ case, both of them receive a Case X from the environment, which is freshly generated.

Next, while PolyG$^\nu$ and PolyC$^\nu$ have a single environment Γ that includes type variables and term variables, in CBPV$_{\text{OSum}}$, these are separated into a type variable environment Δ and a term variable environment Γ. For this reason in the right side of Figure11 we define the translation of an environment Γ to be a pair of environments in CBPV$_{\text{OSum}}$. The term variable $x : A$ is just translated to a variable $x : [\![A]\!]$, but the type variables are more interesting. An abstract type variable X is translated to a pair of a type variable X but also an associated term variable c_X : Case X, which represents the case of the dynamic type that will be instantiated with a freshly generated case. On the other hand, since known type variables $X \cong A$ are translated to $[\![A]\!]$, we do not extend Δ with a new variable, but still produce a variable $c_X : [\![A]\!]$ as with an unknown type variable. Finally, the empty context $\cdot$ is translated to a pair of empty contexts.

To translate a whole program, written $[\![\Sigma; \cdot \vdash M]\!]_p$, we insert a preamble that generates the cases of the open sum type for each ground type. In Figure 12, we show our whole-program translation which inserts a preamble to generate a case of the OSum type for each ground type. This allows us to assume the existence of these cases in the rest of the translation. These can be conveniently

$$
\begin{array}{lcl}
[\![M]\!]_p & = & \texttt{newcase}_{[\![\mathbb{B}]\!]}\ c_{\text{Bool}};\\
& & \texttt{newcase}_{[\![?\to?]\!]}\ c_{\text{Fun}};\\
& & \texttt{newcase}_{[\![?\times?]\!]}\ c_{\text{Times}};\\
& & \texttt{newcase}_{[\![\exists^\nu X.?]\!]}\ c_{\text{Ex}};\\
& & \texttt{newcase}_{[\![\forall^\nu X.?]\!]}\ c_{\text{All}};\\
& & [\![M]\!]
\end{array}
\qquad
\begin{array}{lcl}
\text{case}(\mathbb{B}) & = & c_{\text{Bool}}\\
\text{case}(A\to B) & = & c_{\text{Fun}}\\
\text{case}(A\times B) & = & c_{\text{Times}}\\
\text{case}(\exists^\nu X.A) & = & c_{\text{Ex}}\\
\text{case}(\forall^\nu X.A) & = & c_{\text{All}}\\
\text{case}(X) & = & c_X\\
\text{case}(\sigma) & = & \sigma
\end{array}
$$

$$\Gamma_p \quad = \quad \text{Bool} \cong \mathbb{B}, \text{Fun} \cong ? \to ?, \text{Times} \cong ? \times ?, \text{Ex} \cong \exists^\nu X.?, \text{All} \cong \forall^\nu X.?$$

Fig. 12. Ground type tag management

$$
\begin{array}{lcl}
[\![x]\!] & = & \texttt{ret}\ x\\
[\![\texttt{let}\ x = M; N]\!] & = & x \leftarrow [\![M]\!]; [\![N]\!]\\
[\![\mho_A]\!] & = & \mho\\
[\![\texttt{seal}_X M]\!] & = & [\![M]\!]\\
[\![\texttt{unseal}_X M]\!] & = & [\![M]\!]\\
[\![\texttt{inj}_G\ M]\!] & = & r \leftarrow [\![M]\!]; \texttt{ret inj}_{\text{case}(G)}\ r\\
[\![\texttt{is}(G)?\ M]\!] & = & r \leftarrow [\![M]\!]; \texttt{match}\ r\,\texttt{with}\ \text{case}(G)\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\}\\
[\![\texttt{hide}\ X \cong A; M]\!] & = & \texttt{newcase}_{[\![A]\!]}\ c_X; [\![M]\!]\\
[\![\langle A^{\sqsubseteq}\rangle\updownarrow M]\!] & = & [\![A^{\sqsubseteq}]\!]\updownarrow[[\![M]\!]]\\
[\![\lambda(x:A).M]\!] & = & \texttt{ret thunk}\ \lambda(x:[\![A]\!]).[\![M]\!]\\
[\![M\ N]\!] & = & f \leftarrow [\![M]\!]; a \leftarrow [\![N]\!]; (\texttt{force}\ f)\ a\\
[\![\texttt{pack}^\nu(X \cong A, M)]\!] & = & \texttt{ret pack}(A, \texttt{thunk}\ (\lambda c_X : \text{Case}\ A.[\![M]\!]))\\
[\![\texttt{unpack}\ (X,x) = M; N]\!] & = & r \leftarrow [\![M]\!]; \texttt{unpack}\ (X,f) = r; \texttt{newcase}_X\ c_X; x \leftarrow (\texttt{force}\ f)\ c_X; [\![N]\!]\\
[\![\Lambda^\nu X.M]\!] & = & \texttt{ret}\ (\texttt{thunk}\ (\Lambda X.\lambda(c_X : \text{Case}\ X).[\![M]\!]))\\
[\![M\{X \cong A\}]\!] & = & f \leftarrow [\![M]\!]; (\texttt{force}\ f)[A]\ c_X
\end{array}
$$

$$
\begin{array}{lcl}
[\![G]\!]\updownarrow & = & \bullet \quad (\text{when } G = ?,\ \alpha,\ \text{or}\ \mathbb{B})\\
[\![\text{tag}_G(A^{\sqsubseteq})]\!]\nwarrow & = & r \leftarrow [\![A^{\sqsubseteq}]\!]\nwarrow[\bullet]; \texttt{ret inj}_{\text{case}(G)}\ r\\
[\![\text{tag}_G(A^{\sqsubseteq})]\!]\searrow & = & x \leftarrow \bullet; \texttt{match}\ x\,\texttt{with}\ \text{case}(G)\{\texttt{inj}\ y.[\![A^{\sqsubseteq}]\!]\searrow[\texttt{ret}\ y] \mid \mho\}\\
[\![A_1^{\sqsubseteq} \times A_2^{\sqsubseteq}]\!]\updownarrow & = & x \leftarrow \bullet; \texttt{let}\ (x_1,x_2) = x; x_1' \leftarrow [\![A_1^{\sqsubseteq}]\!]\updownarrow[\texttt{ret}\ x_1]; x_2' \leftarrow [\![A_2^{\sqsubseteq}]\!]\updownarrow[\texttt{ret}\ x_2]; \texttt{ret}\ (x_1', x_2')\\
[\![A_1^{\sqsubseteq} \to A_2^{\sqsubseteq}]\!]\updownarrow & = & x \leftarrow \bullet; \texttt{ret thunk}\ (\lambda y : A'.a \leftarrow [\![A_1^{\sqsubseteq}]\!]\updownarrow^-[\texttt{ret}\ y]; [\![A_2^{\sqsubseteq}]\!]\updownarrow[\texttt{force}\ x\ a])\\
& & \text{where } A_1^{\sqsubseteq} : A_{1l} \sqsubseteq A_{1r} \text{ and if } \updownarrow = \nwarrow, A' = A_{1r}, \text{ else } A' = A_{1l}\\
[\![\forall^\nu X.A^{\sqsubseteq}]\!]\updownarrow & = & x \leftarrow \bullet; \texttt{ret thunk}\ (\Lambda X.\lambda c_X : \text{Case}\ X.[\![A^{\sqsubseteq}]\!]\updownarrow[\texttt{force}\ x\ [X]\ c_X])\\
[\![\exists^\nu X.A^{\sqsubseteq}]\!]\updownarrow & = & x \leftarrow \bullet; \texttt{unpack}\ (Y,f) = x; \texttt{ret pack}(Y, \texttt{thunk}\ (\lambda c_X : \text{Case}\ Y.[\![A^{\sqsubseteq}]\!]\updownarrow[(\texttt{force}\ f)\ c_X]))
\end{array}
$$

Fig. 13. PolyC$^\nu$ term translation (fragment)

modeled as a sequence of "global" definitions of some known type variables, which we write as Γ_p. We also define a function case$(\cdot)$ from types to their corresponding case value, which is a case variable for all types except those generated at runtime σ.

Next, we consider the term translation, which is defined with the below type preservation Theorem 5.1 in mind. First, all PolyC$^\nu$ terms of type A are translated to CBPV$_{\text{OSum}}$ *computations*, with type $F[\![A]\!]$, which is standard for translating CBV to CBPV. Also, note that the *output* environment of fresh type names in a term is just translated as an extension to the input environment, the difference is irrelevant in the translated code, because the names themselves are actually generated in the translation of the *hide* form. Finally, we include the preamble context Γ_p to the front of the terms to account for the fact that all terms can use the cases generated in the preamble.

Theorem 5.1. *If* $\Gamma_1 \vdash M : A, \Gamma_2$ *then* $\Delta; \Gamma \vdash [\![M]\!] : F[\![\Sigma; \Gamma_1, \Gamma_2 \vdash A]\!]$ *where* $[\![\Gamma_p, \Gamma_1, \Gamma_2]\!] = \Delta; \Gamma$.

We show most of the term translation in Figure 13. To reduce the context clutter in the translations, we elide the contexts Σ, Γ in the definition of the semantics. While they are technically needed to translate type annotations, they do not affect the operational semantics and so can be safely ignored. We put the bool, pair, and our pack-cast intermediate form cases in the appendix [New et al. 2020].

Variables translate to a return of the variable, let is translated bind, and errors are translated to errors. Since the type translation maps known type variables to their bound types, the target language `seal` and `unseal` disappear in the translation. Injection into the dynamic type translates to injection into the open sum type and ground type checks in PolyC$^\nu$ are implemented using pattern matching on OSum in CBPV$_{\text{OSum}}$. Next, the hide form is translated to a newcase form.

Next, we cover the cases involving thunks. As a warmup, the functions follow the usual CBV translation into CBPV: a CBV λ is translated to a thunk of a CBPV λ, and the application translation makes the evaluation order explicit and forces the function with an input. We translate existential packages in the cast calculus to CBPV$_{\text{OSum}}$ packages containing functions from a case of the open sum type to the body of the package. In PolyC$^\nu$ we delay execution of `pack` bodies, so the translation inserts a `thunk` to make the order of execution explicit. Since `pack` bodies translate to functions, the translation of an `unpack` must provide a case of the open sum type to the package it unwraps. Type abstractions (Λ^ν), like `pack`s, wrap their bodies in functions that, on instantiation, expect a case of the open sum type matching the instantiating type. Since `hide` generates the requisite type name, it translates to a `newcase`. A type application then simply plugs its given type and the tag associated with its type variable into the supplied type abstraction.

Next, we define the implementation of casts as "contracts", i.e., ordinary functions in the CBPV$_{\text{OSum}}$. Reflexive casts at atomic types, ?, α, and $\mathbb{B}$, translate away. Structural casts at composite types, pair types, function types, universals, and existentials, push casts for their sub-parts into terms of each type. Function and product casts are entirely standard, noting that we use $r(A^{\sqsubseteq}) = A_r$. Universal casts delay until type application and then cast the output. Existential casts push their subcasts into whatever package they are given.

5.4 Simulation

In §6, we establish graduality and parametricity theorems for PolyG$^\nu$/PolyC$^\nu$ by analysis of the semantics of terms translated into CBPV$_{\text{OSum}}$. But since we take the operational semantics of PolyC$^\nu$ as definitional, we need to bridge the gap between the operational semantics in CBPV$_{\text{OSum}}$ and PolyC$^\nu$ by proving the following adequacy theorem that says that the final behavior of terms in PolyC$^\nu$ is the same as the behavior of their translations:

Theorem 5.2 (Adequacy). *If $\cdot \vdash M : A; \cdot$, then $M \Uparrow$ if and only if $[\![M]\!]_p \Uparrow$ and $M \Downarrow V$ if and only if $[\![M]\!]_p \Downarrow V'$ and $M \Downarrow \mho$ if and only if $[\![M]\!]_p \Downarrow \mho$.*

The proof of the theorem follows by a forward *simulation* argument given in the appendix, adapting a similar CBPV simulation given by Levy [2003], and proves that the V and V' in the adequacy proof are in the simulation relation [New et al. 2020].

6 GRADUALITY AND PARAMETRICITY

In this section we prove the central metatheoretic results of the paper: that our surface language satisfies both graduality and parametricity theorems. Each of these is considered a technical challenge to prove: parametricity is typically proven by a logical relation and graduality is proven either by a simulation argument [Siek et al. 2015] or a logical relation [New and Ahmed 2018; New et al. 2019], so in the worst case this would require two highly technical developments. However, we show that this is not necessary: the logical relations proof for graduality is general enough that

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vdash (M_l :: B_l) \sqsubseteq (M_r :: B_r) : B^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}} \qquad \frac{\Gamma^{\sqsubseteq}, X \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_o}{\Gamma^{\sqsubseteq} \vdash \Lambda^{\nu} X.M_l \sqsubseteq \Lambda^{\nu} X.M_r : \forall^{\nu} X.A^{\sqsubseteq}; \cdot}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vdash M_l\{X \cong B_l\} \sqsubseteq M_r\{X \cong B_r\} : \mathrm{un}\forall^{\nu}(A^{\sqsubseteq}); \Gamma^{\sqsubseteq}_M, X \cong B^{\sqsubseteq}}$$

Fig. 14. PolyG$^{\nu}$ Term Precision (fragment)

the parametricity theorem is a corollary of the *reflexivity* of the logical relation. This substantiates the analogy between parametricity and graduality originally proposed in [New and Ahmed 2018].

The key to sharing this work is that we give a novel *relational* interpretation of type precision derivations. That is, our logical relation is indexed not by types, but by type precision derivations. For any derivation $A^{\sqsubseteq} : A_l \sqsubseteq A_r$, we define a relation $\mathcal{V}[\![A^{\sqsubseteq}]\!]$ between values of A_l and A_r. By taking the reflexivity case $A : A \sqsubseteq A$, we recover the parametricity logical relation. Previous logical relations proofs of graduality defined a logical relation indexed by types, and used casts to define a second relation based on type precision judgments, but the direct relational approach simplifies the proofs and immediately applies to parametricity as well.

6.1 Term Precision

To state the graduality theorem, we begin by formalizing the syntactic *term* precision relation. The intuition behind a precision relation $M \sqsubseteq M'$ is that M' is a (somewhat) dynamically typed term and we have changed some of its type annotations to be more precise, producing M. This is one of the main intended use cases for a gradual language: hardening the types of programs over time. Restated in a less directed way, a term M is (syntactically) more precise than M' when the types and annotations in M are more precise than M' and otherwise the terms have the same structure. We formalize this as a judgment $\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}$, where $\Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r$ is a type precision derivation and $\Gamma^{\sqsubseteq} : \Gamma_l \sqsubseteq \Gamma_r$ and $\Gamma^{\sqsubseteq\prime} : \Gamma_l' \sqsubseteq \Gamma_r'$ are precision *contexts* and $\Gamma_l \vdash M_l : A_l; \Gamma_l'$ and $\Gamma_r \vdash M_r : A_r; \Gamma_r'$. A precision context $\Gamma^{\sqsubseteq}$ is like a precision derivation between two contexts: everywhere a type would be in an ordinary context, a precision derivation is used instead.

We show term precision rules for annotations and $\forall^{\nu}$ introduction and elimination for the surface language in Figure 14, with full rules in the appendix [New et al. 2020]. The rules are all completely structural: just check that the two terms have the same term constructor and all of the corresponding arguments of the rule are $\sqsubseteq$. As exhibited by the $\forall^{\nu}$ elimination rule, the metafunctions $\mathrm{dom}, \mathrm{cod}, \mathrm{un}\forall^{\nu}, \mathrm{un}\exists^{\nu}$ are extended in the obvious way to work on precision derivations. We define a similar notion of term precision for PolyC$^{\nu}$. Again we show the rules for casts and $\forall^{\nu}$ in Figure 15, the full definition is in the appendix [New et al. 2020]. The main difference is that, following [New et al. 2019], we include four rules involving casts: two for downcasts and two for upcasts. We can summarize all four by saying that if $M_l \sqsubseteq M_r$, then adding a cast to either M_l or M_r still maintains that the left side is more precise than the right, as long as the type on the left is more precise than the right. Semantically, these are the most important term precision rules, as they bridge the worlds of type and term precision.

Then the key lemma is that the elaboration process from PolyG$^{\nu}$ to PolyC$^{\nu}$ preserves term precision. The proof, presented in the appendix, follows by induction on term precision proofs [New et al. 2020].

Lemma 6.1. *If* $\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}$ *in the surface language, then* $\Gamma^{\sqsubseteq} \vdash M_l^{+} \sqsubseteq M_r^{+} : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \\ \Gamma, \Gamma' \vdash AB^{\sqsubseteq} : A \sqsubseteq B \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash BC^{\sqsubseteq} : B \sqsubseteq C}{\Gamma^{\sqsubseteq} \vdash \langle AB^{\sqsubseteq} \rangle \!\uparrow M_l \sqsubseteq M_r : BC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : BC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \\ \Gamma, \Gamma' \vdash AB^{\sqsubseteq} : A \sqsubseteq B \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash BC^{\sqsubseteq} : B \sqsubseteq C}{\Gamma^{\sqsubseteq} \vdash \langle AB^{\sqsubseteq} \rangle \!\downarrow M_l \sqsubseteq M_r : AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : AB^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \\ \Gamma, \Gamma' \vdash BC^{\sqsubseteq} : B \sqsubseteq C \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AB^{\sqsubseteq} : A \sqsubseteq B}{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq \langle BC^{\sqsubseteq} \rangle \!\uparrow M_r : AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \\ \Gamma, \Gamma' \vdash BC^{\sqsubseteq} : B \sqsubseteq C \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AB^{\sqsubseteq} : A \sqsubseteq B}{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq \langle BC^{\sqsubseteq} \rangle \!\downarrow M_r : AB^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : \forall^{\nu} X.A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vdash M_l\{X \cong B_l\} \sqsubseteq M_r\{X \cong B_r\} : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}, X \cong B^{\sqsubseteq}}$$

$$\frac{\Gamma^{\sqsubseteq}, X \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \cdot}{\Gamma^{\sqsubseteq} \vdash \Lambda^{\nu} X.M_l \sqsubseteq \Lambda^{\nu} X.M_r : \forall^{\nu} X.A^{\sqsubseteq}; \cdot}$$

Fig. 15. PolyC$^{\nu}$ Term Precision (fragment)

6.2 Graduality Theorem

The graduality theorem states that if a term M is *syntactically* more precise than a term M', then M *semantically* refines the behavior of M': it may error, but otherwise it has the same behavior as M': if it diverges so does M' and if it terminates at V, M' terminates with some V' as well. If we think of M as the result of hardening the types of M', then this shows that hardening types semantically only increases the burden of runtime type checking and doesn't otherwise interfere with program behavior. We call this *operational* graduality, as we will consider some related notions later.

Theorem 6.2 (Operational Graduality). *If* $\cdot \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \cdot$, *then either* $M_l^+ \Downarrow \mho$ *or both terms diverge* $M_l^+, M_r^+ \Uparrow$ *or both terms terminate successfully* $M_l^+ \Downarrow V_l$ *and* $M_r^+ \Downarrow V_r$.

6.3 Logical Relation

The basic idea of the logical relations proof to proving graduality is to interpret a term precision judgment $\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}$ in a *relational* manner. That is, to every type precision derivation $A^{\sqsubseteq} : A_l \sqsubseteq A_r$, we associate a relation $\mathcal{V}[\![A^{\sqsubseteq}]\!]$ between closed values of types A_l and A_r. Then we define a semantic version of the term precision judgment $\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in A^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}$ which says that given inputs satisfying the relations in $\Gamma^{\sqsubseteq}, \Gamma_o^{\sqsubseteq}$, then either M_l will error, both sides diverge, or M_l and M_r will terminate with values in the relation $\mathcal{V}[\![A^{\sqsubseteq}]\!]$. We define this relation over $\text{CBPV}_{\text{OSum}}$ translations of PolyC$^{\nu}$ terms, rather than PolyC$^{\nu}$ terms because the operational semantics is simpler.

More precisely, we use the now well established technique of Kripke, step-indexed logical relations [Ahmed et al. 2009]. Because the language includes allocation of fresh type names at runtime, the set of values that should be in the relation grows as the store increases. This is modeled *Kripke* structure, which indexes the relation by a "possible world" that attaches invariants to the allocated cases. Because our language includes diverging programs (due to the open sum type), we need to use a *step-indexed* relation that decrements when pattern matching on OSum, and "times out" when the step index hits 0. Finally, following [New and Ahmed 2018; New et al. 2019], to model graduality we need to associate two relations to each type precision derivation: one which times out when the left hand hand term runs out of steps, but allows the right hand side to take any number of steps and vice-versa one that times out when the right runs out of steps.

Figure 16 includes preliminary definitions we need for the logical relation. First, $\text{Atom}_n[A_l, A_r]$ and $\text{CAtom}_n[A_l, A_r]$ define the world-term-term triples that the relations are defined over. A relation $R \in \text{Rel}_n[A_l, A_r]$ at stage n consists of triples of a world, and a value of type A_l and a value of type A_r (ignore the n for now) such that it is monotone in the world. The world $w \in \text{World}_n$ contains the number of steps remaining $w.j$, the current state of each side $w.\Sigma_l$, $w.\Sigma_r$, and finally an interpretation of how the cases in the two stores are related $w.\eta$. An interpretation $\eta \in \text{Interp}_n[\Sigma_l, \Sigma_r]$ consists of a cardinality $\eta.size$ which says how many cases are related and a

$$
\begin{array}{rcl}
\mathrm{Atom}_n[A_l, A_r] & = & \{(w, V_l, V_r) \mid w \in \mathrm{World}_n \wedge (w.\Sigma_l \mid \cdot \vdash V_l : A_l) \wedge (w.\Sigma_r \mid \cdot \vdash V_r : A_r)\} \\
\mathrm{CAtom}_n[A_l, A_r] & = & \{(w, V_l, V_r) \mid w \in \mathrm{World}_n \wedge w.\Sigma_l \mid \cdot \vdash M_l : FA_l \wedge w.\Sigma_r \mid \cdot \vdash M_r : FA_r\} \\
\mathrm{Rel}_n[A_l, A_r] & = & \{R \subseteq \mathrm{Atom}_n[A_l, A_r] \mid \forall (w, V_l, V_r) \in R, w' \sqsupseteq w.(w', V_l, V_r) \in R\} \\
\mathrm{World}_n & = & \{(j, \Sigma_l, \Sigma_r, \eta) \mid j < n \wedge \eta \in \mathrm{Interp}_j[\Sigma_l, \Sigma_r]\} \\
\mathrm{Interp}_n[\Sigma_l, \Sigma_r] & = & \{(size, f, \rho) \mid size \in \mathbb{N} \wedge f \in [size] \to ([\Sigma_l.size] \times [\Sigma_2.size]) \\
& & \quad \wedge \rho : (i < size) \to Rel_n[\Sigma_l(f(i)); \Sigma_r(f(i))] \\
& & \quad \wedge \forall i < j < size. f(i)_l \neq f(j)_l \wedge f(i)_r \neq f(j)_r\} \\
w' \sqsupset w. & = & (w' \sqsupseteq w) \wedge w'.j > w.j \\
w' \sqsupseteq w & = & w'.j \leq w.j \wedge w'.\Sigma_l \sqsupseteq w.\Sigma_l \wedge w'.\Sigma_r \sqsupseteq w.\Sigma_r \wedge w'.\eta \sqsupseteq \lfloor w.\eta \rfloor_{w'.j} \\
\Sigma' \sqsupseteq \Sigma & = & \Sigma'.size \geq \Sigma.size \wedge \forall i < \Sigma.size.\ \Sigma'(i) = \Sigma(i) \\
\eta' \sqsupseteq \eta & = & \eta'.size \geq \eta \wedge \forall i < \eta.size.\ \eta'.f(i) = \eta.f(i) \wedge \eta'.\rho(i) = \eta.\rho(i) \\
\triangleright R & = & \{(w, V_l, V_r) \mid \forall w' \sqsupset w.(w', V_l, V_r) \in R\} \\
\mathrm{Rel}_\omega[A_l, A_r] & = & \{R \subseteq \bigcup_{n \in \mathbb{N}} \mathrm{Atom}_n[A_l, A_r] \mid \forall n \in \mathbb{N}.\ \lfloor R \rfloor_n \in \mathrm{Rel}_n[A_l, A_r]\} \\
\lfloor \eta \rfloor_n & = & (\eta.size, \eta.f, \lambda i. \lfloor \rho(i) \rfloor_n) \\
\lfloor R \rfloor_n & = & \{(w, V_l, V_r) \mid w.j < n \wedge (w, V_l, V_r) \in R\} \\
\eta \vDash (\sigma_l, \sigma_r, R) & = & \exists i < \eta.size.\ \eta.f(i) = (\sigma_l, \sigma_r) \wedge \eta.\rho(i) = R\}
\end{array}
$$

Fig. 16. Logical Relation Auxiliary Definitions

function $\eta.f$ which says *which* cases are related, i.e., for each $i \in \eta.size$ it gives a pair of cases, one valid in the left hand store and one in the right. Finally, $\eta.\rho$ gives a relation between the types of these two cases. The final side-condition says this association is a *partial bijection*: a case on one side is associated to at most one case on the other side. Staging the relations and worlds is necessary due to a circularity here: a relation is (contravariantly) dependent on the current world, which contains relations. A relation in Rel_n is indexed by a World_n, but a World_n contains relations in $\mathrm{Rel}_{w.j}$, and $w.j < n$. In particular, $\mathrm{World}_0 = \emptyset$, so the definition is well-founded.

The next portion of the figure contains the definition of *world* extension $w' \sqsupseteq w$[2], representing the idea that w' is a possible "future" of w: the step index j is smaller and the states of the two sides have increased the number of allocated cases, but the old invariants are still there. We define strict extension $w' \sqsupset w$ to mean extension where the step has gotten strictly smaller. This allows us to define the *later* relation $\triangleright R$ which is used to break circular definitions in the logical relation. Next, we define our notion of non-indexed relation Rel_ω, which is what we quantify over in the interpretation of $\forall^\nu, \exists^\nu$. Then we define the restriction of interpretations and relations to a stage n. An infinitary relation can be "clamped" to any stage n using $\lfloor R \rfloor_n$. Finally, we define when two cases are related in an interpretation as $\eta \vDash (\sigma_l, \sigma_r, R)$.

The top of Figure 17 contains the definition of the logical relation on values and computations, except for the standard cases for booleans, products and functions, which are included in the appendix [New et al. 2020]. First, we write $\sim$ as a metavariable that ranges over two symbols: $\prec$ which indicates that we are counting steps on the left side, and $\succ$ which indicates we are counting steps on the right side. We then define the value relation $\mathcal{V}_n^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta \in \mathrm{Rel}_n[\delta_l(A_l), \delta_r(A_r)]$. Here γ maps the free term variables to pairs of values and δ maps free type variables to triples of two types and a relation between them. First, the definition for type variables looks up the relation in the relational substitution δ. Next, two values in ? are related when they are both injections into OSum, and the "payloads" of the injections are *later* related in the relation R which the world associates to the corresponding cases. The $\triangleright$ is used because we count pattern matching on OSum as a step. This also crucially lines up with the fact that pattern matching on the open sum type is the only reduction that consumes a step in our operational semantics. Note that this is a generalization of the logical relation definition for a recursive sum type, where each injection corresponds to a

[2]there is a clash of notation between precision $\sqsubseteq$ and world extension $\sqsupseteq$ but it should be clear which is meant at any time.

case of the sum. Here since the sum type is open, we must look in the world to see what cases are allocated. Next, the $\mathrm{tag}_G(A^{\sqsubseteq})$ case relates values on the left at some type A_l and values on the right of type ?. The definition states that the dynamically typed value must be an injection using the tag given by G, and that the payload of that injection must be related to V_l with the relation given by $A^{\sqsubseteq}$. This case splits into two because we are pattern matching on a value of the open sum type, and so in the $\succ$ case we must decrement because we are consuming a step on the right, whereas in the $\prec$ case we do not decrement because we are only counting steps on the left. In the $\forall^{\nu}X.B^{\sqsubseteq}$ case, two values are related when in any future world, and any relation $R \in \mathrm{Rel}_{\omega}[A_l, A_r]$, and any pair of cases σ_l, σ_r that have $\lfloor R \rfloor_{w'.j}$ as their associated relation, if the values are instantiated with A_l, σ_l and A_r, σ_r respectively, then they behave as related computations. The intuition is that values of type $\forall^{\nu}X.B$ are parameterized by a type A and a tag for that type σ, but the relational interpretation of the two must be the *same*. This is the key to proving the $\mathtt{seal}_X$ and $\mathtt{unseal}_X$ cases of graduality. The fresh existential is dual in the choice of relation, but the same in its treatment of the case σ.

Next, we define the relation on expressions. The two expression relations, $\mathcal{E}^{\prec}[\![A^{\sqsubseteq}]\!]$ and $\mathcal{E}^{\succ}[\![A^{\sqsubseteq}]\!]$ capture the semantic idea behind graduality: either the left expression raises an error, or the two programs have the same behavior: diverge or return related values in $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]$. However, to account for step-indexing, each is an *approximation* to this notion where $\mathcal{E}^{\prec}[\![A^{\sqsubseteq}]\!]$ times out if the left side consumes all of the available steps $w.j$ (where $(\Sigma, M) \mapsto^{j}$ is shorthand for saying it steps to something in j steps), and $\mathcal{E}^{\succ}[\![A^{\sqsubseteq}]\!]$ times out if the right side consumes all of the available steps. relation is that when We define the *infinitary* version of the relations $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]$ and $\mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]$ a the union of all of the level n approximations.

Next, we give the relational interpretation of environments. The interpretation of the empty environment are empty substitutions with a valid world w. Extending with a value variable $x : A^{\sqsubseteq}$ means extending γ with a pair of values related by $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]$. For an abstract type variable X, first δ is extended with a pair of types and a relation between them. Then, γ must also be extended with a pair of *cases* encoding how these types are injected into the dynamic type. Crucially, just as with the $\forall^{\nu}, \exists^{\nu}$ value relations, these cases must be associated by w to the $w.j$ approximation of the *same* relation with which we extend δ. The interpretation of the *known* type variables $X \cong A^{\sqsubseteq}$ has the same basic structure, the key difference is that rather than using an arbitrary, δ is extended with the value relation $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]$.

With all of that preparation finished, we finally define the semantic interpretation of the graduality judgment $\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}$ in the bottom of Figure 17. First, it says that both $M_l \sqsubseteq\prec M_r$ and $M_l \sqsubseteq\succ M_r$ hold, where we define $\sqsubseteq\sim$ to mean that for any valid instantiation of the environments (including the preamble Γ_p), we get related computations. We can then define the "logical" Graduality theorem, that syntactic term precision implies semantic term precision, briefly, $\vdash$ implies $\vDash$.

Theorem 6.3 (Logical Graduality). *If* $\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}$, *then* $\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}$

The proof is by induction on the term precision derivation. Each case is proven as a separate lemma in the appendix [New et al. 2020]. The cases of $\forall^{\nu}, \exists^{\nu}$, sealing and unsealing follow because the treatment of type variables between the value and environment relations is the same. In the hide case, the world is extended with $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]$ as the relation between new cases. The cast cases are the most involved, following by two lemmas proven by induction over precision derivations: one for when the cast is on the left, and the other when the cast is on the right.

Finally, we prove the operational graduality theorem as a corollary of the logical graduality Theorem 6.3 and the adequacy Theorem 5.2. By constructing a suitable starting world w_{pre} that allocates the globally known tags, we ensure the operational graduality property holds for the code translated to $\mathrm{CBPV}_{\mathrm{OSum}}$, and then the simulation theorem implies the analogous property holds for the PolyC^{ν} operational semantics.

$$
\begin{array}{rcl}
\mathcal{V}_n^{\sim}[\![X]\!]\gamma\delta & = & \lfloor\delta(X)\rfloor_n \\
\mathcal{V}_n^{\sim}[\![?]\!]\gamma\delta & = & \{(w, \mathtt{inj}_{\sigma_l}\ V_l, \mathtt{inj}_{\sigma_r}\ V_r) \in \mathrm{Atom}_n[?]\delta \mid w.\eta \vDash (\sigma_l, \sigma_r, R) \wedge (w, V_l, V_r) \in \triangleright R\} \\
\mathcal{V}_n^{\prec}[\![\mathrm{tag}_G(A^{\sqsubseteq})]\!]\gamma\delta & = & \{(w, V_l, \mathtt{inj}_{\gamma_r(\mathrm{case}(G))}\ V_r) \in \mathrm{Atom}_n[\mathrm{tag}_G(A^{\sqsubseteq})]\delta \mid (w, V_l, V_r) \in \mathcal{V}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta\} \\
\mathcal{V}_n^{\succ}[\![\mathrm{tag}_G(A^{\sqsubseteq})]\!]\gamma\delta & = & \{(w, V_l, \mathtt{inj}_{\gamma_r(\mathrm{case}(G))}\ V_r) \in \mathrm{Atom}_n[\mathrm{tag}_G(A^{\sqsubseteq})]\delta \mid (w, V_l, V_r) \in (\triangleright\mathcal{V}^{\succ}[\![A^{\sqsubseteq}]\!]\gamma\delta)\} \\
\mathcal{V}_n^{\sim}[\![\forall^{\nu}X.B^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, V_l, V_r) \in \mathrm{Atom}_n[\forall^{\nu}X.A^{\sqsubseteq}]\delta \mid \\
& & \quad \forall R \in \mathrm{Rel}_{\omega}[A_l, A_r].\forall w' \sqsupseteq w.\forall \sigma_l, \sigma_r.w'.\eta \vDash (\sigma_l, \sigma_r, \lfloor R\rfloor_{w'.j}) \implies \\
& & \quad (w', \mathtt{force}\ V_l\ [A_l]\ \sigma_l, \mathtt{force}\ V_r\ [A_r]\ \sigma_r) \in \mathcal{E}_n^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta' \\
& & \quad (\text{where } \gamma' = \gamma, c_X \mapsto (\sigma_l, \sigma_r), \text{ and } \delta' = \delta, X \mapsto (A_l, A_r, R))\} \\
\mathcal{V}_n^{\sim}[\![\exists^{\nu}X.B^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, \mathtt{pack}\ (A_l, V_l), \mathtt{pack}\ (A_r, V_r)) \in \mathrm{Atom}_n[\exists^{\nu}X.B^{\sqsubseteq}]\delta \mid \\
& & \quad \exists R \in \mathrm{Rel}_{\omega}[A_l, A_r].\forall w' \sqsupseteq w.\forall \sigma_l, \sigma_r.w'.\eta \vDash (\sigma_l, \sigma_r, \lfloor R\rfloor_{w'.j}) \implies \\
& & \quad (\mathtt{force}\ V_l\ \sigma_l, \mathtt{force}\ V_r\ \sigma_r) \in \mathcal{E}_n^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta' \\
& & \quad (\text{where } \gamma' = \gamma, c_X \mapsto (\sigma_l, \sigma_r), \text{ and } \delta' = \delta, X \mapsto (A_l, A_r, R))\} \\
\mathcal{E}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, M_l, M_r) \in \mathrm{CAtom}_n[A^{\sqsubseteq}]\delta \mid (w.\Sigma_l, M_l) \mapsto^{w.j} \vee ((w.\Sigma_l, M_l) \mapsto^{<w.j} (\Sigma_l', \mho)) \\
& & \vee(\exists w' \sqsupseteq w.(w', V_l, V_r) \in \mathcal{V}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta. \\
& & \quad (w.\Sigma_l, M_l) \mapsto^{w'.j - w.j} (w'.\Sigma_l, \mathtt{ret}\ V_l) \wedge (w.\Sigma_r, M_r) \mapsto^{*} (w'.\Sigma_r, \mathtt{ret}\ V_r))\} \\
\mathcal{E}_n^{\succ}[\![A^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, M_l, M_r) \in \mathrm{CAtom}_n[A^{\sqsubseteq}]\delta \mid (w.\Sigma_r, M_r) \mapsto^{w.j} \vee ((w.\Sigma_l, M_l) \mapsto^{*} (\Sigma_l', \mho)) \\
& & \vee \exists w' \sqsupseteq w.(w', V_l, V_r) \in \mathcal{V}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta. \\
& & \quad (w.\Sigma_l, M_l) \mapsto^{*} (w'.\Sigma_l, \mathtt{ret}\ V_l) \wedge (w.\Sigma_r, M_r) \mapsto^{w'.j - w.j} (w'.\Sigma_r, \mathtt{ret}\ V_r)\}
\end{array}
$$

$$
\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta = \bigcup_{n\in\mathbb{N}} \mathcal{V}_n^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta \qquad\qquad \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta = \bigcup_{n\in\mathbb{N}} \mathcal{E}_n^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta
$$

$$
\begin{array}{rcl}
\mathcal{G}^{\sim}[\![\cdot]\!] & = & \{(w, \emptyset, \emptyset) \mid \exists n.w \in \mathrm{World}_n\} \\
\mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}, x : A^{\sqsubseteq}]\!] & = & \{(w, (\gamma, x \mapsto (V_l, V_r)), \delta) \mid (w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!] \wedge (w, V_l, V_r) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta\} \\
\mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}, X]\!] & = & \{(w, (\gamma, c_X \mapsto (\sigma_l, \sigma_r)), \delta, X \mapsto (A_l, A_r, R)) \mid (w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!] \\
& & \quad \wedge R \in \mathrm{Rel}_{\omega}[A_l, A_r] \wedge (\sigma_l, \sigma_r, \lfloor R\rfloor_{w.j}) \in w\} \\
\mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}, X \cong A^{\sqsubseteq}]\!] & = & \{(w, (\gamma, c_X \mapsto (\sigma_l, \sigma_r)), \delta, X \mapsto (A_l, A_r, \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta)) \mid (w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!] \\
& & \quad w \vDash (\sigma_l, \sigma_r, \mathcal{V}_{w.j}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta)\}
\end{array}
$$

$$
\begin{array}{rcl}
\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} & = & \Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq\prec M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \wedge \Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq\succ M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \\
\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq\sim M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} & = & \forall(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime}]\!]. \\
& & \quad (w, [\![M_l]\!][\gamma_l][\delta_l], [\![M_r]\!][\gamma_l][\delta_l]) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta
\end{array}
$$

Fig. 17. Graduality/Parametricity Logical Relation (fragment)

6.4 Parametricity and Free Theorems

Our relational approach to proving the graduality theorem is not only elegant, it also makes the theorem more general, and in particular it *subsumes* the parametricity theorem that we want for the language, because we already assign arbitrary relations to abstract type variables. Then the parametricity theorem is just the reflexivity case of the graduality theorem.

Theorem 6.4 (Parametricity). *If* $\Gamma \vdash M : A; \Gamma'$, *then* $\Gamma \vDash M^{+} \sqsubseteq M^{+} : A; \Gamma'$.

To demonstrate that this really is a parametricity theorem, we show that from this theorem we can prove "free theorems" that are true in polymorphic languages. These free theorems are naturally stated in terms of *contextual equivalence*, the gold standard for operational equivalence of programs, which we define as both programs diverging, erroring, or terminating successfully when plugged into an arbitrary context. The appendix contains a formal definition [New et al. 2020].

To use our logical relation to prove contextual equivalence, we need the following lemma, which says that semantic term precision both ways is *sound* for PolyG$^{\nu}$ contextual equivalence.

Lemma 6.5. *If* $\Gamma \vDash M_l \sqsubseteq M_2 \in A; \Gamma_M$ *and* $\Gamma \vDash M_2 \sqsubseteq M_1 \in A; \Gamma_M$, *then* $\Gamma \vDash M_l \approx^{ctx} M_2 \in A; \Gamma_M$.

$$\frac{M : \forall^\nu X.X \to X \qquad V_A : A \qquad V_B : B}{\lambda\_ : ?.\mathtt{unseal}_X(M\{X \cong A\}(\mathtt{seal}_X V_A)) \approx^{\mathrm{ctx}} \lambda\_ : ?.\mathtt{let}\ y = M\{X \cong B\}V_B; V_A}$$

$$\frac{M : \forall^\nu X.\forall^\nu Y.(X \times Y) \to (Y \times X) \qquad V_A : A \qquad V_B : B}{\begin{array}{c}\lambda\_ : ?.\mathtt{let}\ (y,x)\ =\ (M\{X \cong A\}\{Y \cong B\}\ (\mathtt{seal}_X V_A, \mathtt{seal}_Y V_B)); (\mathtt{unseal}_X x, \mathtt{unseal}_Y y) \\ \approx^{\mathrm{ctx}} \lambda\_ : ?.\mathtt{let}\ (y,x)\ =\ (M\{X \cong B\}\{Y \cong A\}\ (\mathtt{seal}_X V_B, \mathtt{seal}_Y V_A)); (\mathtt{unseal}_Y y, \mathtt{unseal}_X x)\end{array}}$$

$$\frac{\begin{array}{c}\mathrm{NOT} = \lambda b : \mathbb{B}.\mathtt{if}\ b\ \mathtt{then\ false\ else\ true} \\ \mathrm{WRAPNOT} = \lambda x : X.\mathtt{seal}_X(\mathrm{NOT}(\mathtt{unseal}_X x))\end{array}}{\begin{array}{c}\mathsf{pack}^\nu(X \cong \mathbb{B}, (\mathtt{seal}_X \mathtt{true}, (\mathrm{WRAPNOT}, \lambda x : X.\mathtt{unseal}_X x))) \\ \approx^{\mathrm{ctx}} \mathsf{pack}^\nu(X \cong \mathbb{B}, (\mathtt{seal}_X \mathtt{false}, (\mathrm{WRAPNOT}, \lambda x : X.\mathrm{NOT}\,(\mathtt{unseal}_X x))))\end{array}}$$

Fig. 18. Free Theorems without ?

We now substantiate that this is a parametricity theorem by proving a few contextual equivalence results. First we present adaptations of some standard free theorems from typed languages in Figure 18. The first equivalence shows that the behavior of any term with the "identity function type" $\forall^\nu X.X \to X$ must be independent of the input it is given. We place a λ on each side to delimit the scope of the X outward. Without the X (or a similar thunking feature like $\forall^\nu$ or $\exists^\nu$), the two programs would not have the same (effect) typing. In a more realistic language, this corresponds to wrapping each side in a module boundary. The next result shows that a function $\forall^\nu X.\forall^\nu Y.(X \times Y) \to (Y \times X)$, if it terminates, must flip the values of the pair, and furthermore whether it terminates, diverges or errors does not depend on the input values. Finally, we give an example using existential types. That shows that an abstract "flipper" which uses `true` for on and `false` for off in its internal state is equivalent to one using `false` and `true`, respectively as long as they return the same value in their "readout" function.

Next, to give a flavor of what kind of relational reasoning is possible in the presence of the dynamic type, we consider what free theorems are derivable for functions of type $\forall X.? \to X$. A good intuition for this type is that the only possible outputs of the function are sealed values that are contained within the dynamically typed input. It is difficult to summarize this in a single statement, so instead we give the following three examples:

Theorem 6.6 ($\forall^\nu X.? \to X$ Free Theorems). *Let* $\cdot \vdash M : \forall^\nu X.? \to X$

(1) *For any* $\cdot \vdash V : ?$, *then* $\lambda\_ : ?.\mathtt{unseal}_X(M\{X \cong A\}\,V))\ \mathtt{true}$ *either diverges or errors.*
(2) *For any* $\cdot \vdash V : A$ *and* $\cdot \vdash V' : B$,

$$\lambda\_ : ?.\mathtt{unseal}_X(M\{X \cong A\}\,(\mathtt{seal}_X V)) \approx^{ctx} \lambda\_ : ?.\mathtt{let}\ y = (\mathtt{unseal}_X(M\{X \cong B\}\,(\mathtt{seal}_X V'))); V$$

(3) *For any* $\cdot \vdash V : A, \cdot \vdash V' : B, \cdot \vdash V_d : ?$,

$$\lambda\_ : ?.\mathtt{unseal}_X(M\{X \cong A\}\,(\mathtt{seal}_X V, V_d)) \approx^{ctx} \lambda\_ : ?.\mathtt{let}\ y = (\mathtt{unseal}_X(M\{X \cong B\}\,(\mathtt{seal}_X V', V_d))); V$$

The first example passes in a value V that does not use the seal X, so we know that the function cannot possibly return a value of type X. The second example mimics the identity function's free theorem. It passes in a sealed value V and the equivalence shows that V''s effects do not depend on what V was sealed and the only value that V' can return is the one that was passed in. The third example illustrates that there are complicated ways in which sealed values might be passed as a part of a dynamically typed value, but the principle remains the same: since there is only one sealed value that's part of the larger dynamically typed value, it is the only possible return value, and the effects cannot depend on its actual value. The proof of the first case uses the relational interpretation that X is empty. The latter two use the interpretation that X includes a single value.

Compare this reasoning to what is available in GSF, where the polymorphic *function* determines which inputs are sealed and which are not, rather than the caller. Because of this, Toro et al. [2019]

only prove "cheap" theorems involving ? where the polymorphic function is known to be a literal Λ function and not a casted function. As an example, for arbitrary $M : \forall X.? \rightarrow X$, consider the application $M\,[\mathbb{B}](\mathtt{true}, \mathtt{false})$. The continuation of this call has no way of knowing if neither, one or both booleans are members of the abstract type X. The following examples of possible terms for M illustrate these three cases:

(1) $M_1 = (\Lambda X.\lambda x : \mathbb{B} \times \mathbb{B}.\mathtt{if}\ \mathrm{or}\, x\ \mathtt{then}\ \mho\ \mathtt{else}\ \Omega) :: \forall X.? \rightarrow X$
(2) $M_2 = (\Lambda X.\lambda x : X \times \mathbb{B}.\mathtt{if}\ \mathrm{snd}\, x\ \mathtt{then}\ \mathrm{fst}\, x\ \mathtt{else}\ \Omega) :: \forall X.? \rightarrow X$
(3) $M_3 = (\Lambda X.\lambda x : X \times X.\mathrm{snd}\, x) :: \forall X.? \rightarrow X$

If $M = M_1$, both booleans are concrete so X is empty, but from the inputs the function can determine whether to diverge or error. If $M = M_2$, the first boolean is abstract and the second is concrete, so only the first can inhabit X, but the second can be used to determine whether to return a value or not. Finally if $M = M_3$, both booleans are abstract so the function cannot inspect them, but either can be returned. It is unclear what reasoning the continuation has here: it must anticipate every possible way in which the function might decide which values to seal, and so has to consider every dynamically typed value of the instantiating type as possibly abstract and possibly concrete.

7 DISCUSSION AND RELATED WORK

Dynamically typed PolyG$^\nu$ and Design Alternatives. Most gradually typed languages are based on adding types to an existing dynamically typed language, with the static types capturing some feature already existing in the dynamic language that can be migrated to use static typing. PolyG$^\nu$ was designed as a proof-of-concept standalone gradual language, so it might not be clear what dynamic typing features it supports migration of. In particular, since all sealing is explicit, PolyG$^\nu$ does not model migration from programming without seals entirely to programming with them, so its types are relevant to languages that include some kind of nominal data type generation.

The fresh existential types correspond to a particular mode of use of a module system that supports creation of nominal types. The package itself corresponds to a module with a fresh type declaration. Then sealing corresponds to the constructor of the fresh datatype, and unsealing to pattern matching against it. For example, in Racket structs can be used to make fresh nominal types and *units* provide first-class modules. It would be interesting future work to see if our logical relation can usefully be adapted to Typed Racket's typed units [Tobin-Hochstadt et al. [n. d.]].

Our fresh polymorphic types are more exotic than the fresh existentials, and don't clearly correspond to any existing programming features, but they model abstraction over nominal datatypes where the datatype is guaranteed to be freshly generated. One issue with adding this feature to a realistic language is that the outward scoping of type variables may be undesirable, so it is useful to consider alternative designs that achieve the same abstraction principles. One possible design would be to force an ANF-like [Sabry and Felleisen 1992] restriction on instantiations of polymorphic functions, where all instantiations have to be of the form $\mathtt{let}\ M\{X \cong A\} = f; N$, where $X \cong A$ is bound in N. This makes the scope of the X explicit: it is only bound in N. Our translation could easily be modified to accommodate this feature, we chose instead to consider the outward scoping since it makes it easier to compare to programming in the style of System F.

Our use of abstract and known type variables was directly inspired by Neis et al. [2009], who present a language with a fresh type creation mechanism which they show enables parametric reasoning though the language overall does not satisfy a traditional parametricity theorem. This suggests an alternative language design, where $\forall$ and $\exists$ behave normally and we add a `newtype` facility, analogous to that feature of Haskell, where `newtype` allocates a new case of the open sum type for each type it creates. Such a language would not have known type variables or PolyG$^\nu$'s inside-out scoping of type instantiation, but it would also not be parametric by default. Instead, programmers could manually create fresh types and know that they are abstract to other modules.

Since sealing is explicit in PolyG$^\nu$, it does not provide a drop-in replacement for System F, and so the additional syntactic overhead of sealing and unsealing can be quite heavy, especially when using higher-order combinators. For instance a higher-order function composition combinator has type $\forall^\nu X.\forall^\nu Y.\forall^\nu Z.(Y \to Z) \to (X \to Y) \to X \to Z$ and the System F composition $\mathrm{compose}[\mathbb{I}][\mathbb{B}][\mathbb{B}]\ \mathrm{not}\ (> 0)$ would in PolyG$^\nu$ be written

$$\begin{aligned}\lambda n : \mathbb{I}.\, \mathtt{unseal}_Z(\mathrm{compose}\{X \cong \mathbb{I}\}\{Y \cong \mathbb{B}\}\{Z \cong \mathbb{B}\}&(\lambda y : Y.\mathtt{seal}_Z(\mathrm{not}\,(\mathtt{unseal}_Y y)))\\ &(\lambda x : X.\mathtt{seal}_Y(> 0\,(\mathtt{unseal}_X x)))\\ &\mathtt{seal}_X n)\end{aligned}$$

This syntactic overhead can be mitigated via generic wrapping functions using dynamic typing:

$$\begin{aligned}\mathrm{wrap}\,\mathtt{seal}_X\,\mathtt{unseal}_Z\,(\mathrm{compose}\{X \cong \mathbb{I}\}\{Y \cong \mathbb{B}\}\{Z \cong \mathbb{B}\}&(\mathrm{wrap}\,\mathtt{unseal}_Y\,\mathtt{seal}_Z\,\mathrm{not})\\ &(\mathrm{wrap}\,\mathtt{unseal}_X\,\mathtt{seal}_Y\,(> 0))\end{aligned}$$

But the syntactic overhead cannot be completely removed or done entirely with static typing.

Tag Checking. Siek et al. [2015] claim that graduality demands that tag-checking functions like our `is(𝔹)?` form must error when applied to sealed values, and used this as a criticism of the design of Typed Racket. However, in our language, `is(𝔹)?` will simply return `false`, which matches Typed Racket's behavior. This is desirable if we are adding types to an existing dynamic language, because typically a runtime tag check should be a *safe* operation in a dynamically typed language. Explicit sealing avoids this graduality issue, an advantage over previous work.

Logical Relations. Our use of explicit sealing eliminates much of the complexity of prior logical relations [Ahmed et al. 2017; Toro et al. 2019]. To accommodate dynamic conversion and evidence insertion, those relations adopted complex value relations for universal types that in turn restricted the ways in which they could treat type variables. Additionally, we are the first to give a logical relation for fresh *existential* types, and it is not clear how to adapt the non-standard relation for universals to existentials [Ahmed et al. 2017; Toro et al. 2019].

Next, while we argue that our logical relation more fully captures parametricity than previous work on gradual polymorphism, this is not a fully formal claim. To formalize it, in future work we could show that PolyG$^\nu$ is a model of an effectful variant of an axiomatic parametricity formulation such as Dunphy [2002]; Ma and Reynolds [1991]; Plotkin and Abadi [1993].

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their in-depth comments. We also thank Matthias Felleisen for explaining the importance that tag checks be total operations. This material is based on research supported by the National Science Foundation under grants CCF-1910522, CCF-1816837, and CCF-1453796. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Amal Ahmed, Derek Dreyer, and Andreas Rossberg. 2009. State-Dependent Representation Independence. In *ACM Symposium on Principles of Programming Languages (POPL), Savannah, Georgia.*

Amal Ahmed, Robert Bruce Findler, Jeremy Siek, and Philip Wadler. 2011. Blame for All. In *ACM Symposium on Principles of Programming Languages (POPL), Austin, Texas.* 201–214.

Amal Ahmed, Dustin Jamner, Jeremy G. Siek, and Philip Wadler. 2017. Theorems for Free for Free: Parametricity, With and Without Types. In *International Conference on Functional Programming (ICFP), Oxford, United Kingdom.*

Felipe Bañados Schwerter, Ronald Garcia, and Éric Tanter. 2014. A Theory of Gradual Effect Systems. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming (ICFP '14).* 283–295.

John Boyland. 2014. The Problem of Structural Type Tests in a Gradually-Typed Language. In *21st Workshop on Foundations of Object-Oriented Languages*.

Giuseppe Castagna and Victor Lanvin. 2017. Gradual Typing with Union and Intersection Types. *Proc. ACM Program. Lang.* 1, ICFP, Article 41 (Aug. 2017), 28 pages. https://doi.org/10.1145/3110285

Tim Disney and Cormac Flanagan. 2011. Gradual Information Flow Typing. In *Workshop on Script-to-Program Evolution (STOP)*.

Brian Patrick Dunphy. 2002. *Parametricity As a Notion of Uniformity in Reflexive Graphs*. Ph.D. Dissertation. Champaign, IL, USA. Advisor(s) Reddy, Uday.

Matthias Felleisen. 1990. On the expressive power of programming languages. *ESOP'90* (1990).

Luminous Fennell and Peter Thiemann. 2013. Gradual Security Typing with References. In *CSF*. IEEE Computer Society, 224–239.

Ronald Garcia and Matteo Cimini. 2015. Principal Type Schemes for Gradual Programs *(POPL '15)*.

Ronald Garcia, Alison M. Clark, and Éric Tanter. 2016. Abstracting Gradual Typing. In *ACM Symposium on Principles of Programming Languages (POPL)*.

Atsushi Igarashi, Peter Thiemann, Vasco Vasconcelos, and Philip Wadler. 2017b. Gradual Session Types. In *International Conference on Functional Programming (ICFP)*.

Yuu Igarashi, Taro Sekiyama, and Atsushi Igarashi. 2017a. On Polymorphic Gradual Typing. In *International Conference on Functional Programming (ICFP), Oxford, United Kingdom*.

Lintaro Ina and Atsushi Igarashi. 2011. Gradual typing for generics. In *Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications (OOPSLA '11)*.

Nico Lehmann and Éric Tanter. 2017. Gradual Refinement Types. In *ACM Symposium on Principles of Programming Languages (POPL)*.

Paul Blain Levy. 2003. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Springer.

QingMing Ma and John C. Reynolds. 1991. Types, Abstractions, and Parametric Polymorphism, Part 2. In *Mathematical Foundations of Programming Semantics, 7th International Conference, Pittsburgh, PA, USA*.

John C. Mitchell and Gordon D. Plotkin. 1985. Abstract types have existential type. In *ACM Symposium on Principles of Programming Languages (POPL), New Orleans, Louisiana*.

Georg Neis, Derek Dreyer, and Andreas Rossberg. 2009. Non-Parametric Parametricity. In *International Conference on Functional Programming (ICFP)*. 135–148.

Max S. New and Amal Ahmed. 2018. Graduality from Embedding-Projection Pairs. In *International Conference on Functional Programming (ICFP), St. Louis, Missouri*.

Max S. New, Dustin Jamner, and Amal Ahmed. 2020. Technical Appendix to Graduality and Parametricity: Together Again for the First Time. http://www.ccs.neu.edu/home/amal/papers/gradparam-tr.pdf

Max S. New, Daniel R. Licata, and Amal Ahmed. 2019. Gradual Type Theory *(POPL '19)*.

Gordon Plotkin and Martín Abadi. 1993. A logic for parametric polymorphism. *Typed Lambda Calculi and Applications* (1993), 361–375.

John C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France*.

Amr Sabry and Matthias Felleisen. 1992. Reasoning about Programs in Continuation-Passing Style. In *Conf. on LISP and functional programming, LFP '92*.

Ilya Sergey and Dave Clarke. 2012. Gradual Ownership Types. In *ESOP (Lecture Notes in Computer Science)*, Vol. 7211. Springer, 579–599.

Jeremy Siek, Micahel Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined Criteria for Gradual Typing. In *1st Summit on Advances in Programming Languages*.

Jeremy G. Siek and Walid Taha. 2006. Gradual Typing for Functional Languages. In *Scheme and Functional Programming Workshop (Scheme)*. 81–92.

Jeremy G. Siek and Walid Taha. 2007. Gradual Typing for Objects. In *European Conference on Object-Oriented Programming (ECOOP)*.

Eijiro Sumii and Benjamin C. Pierce. 2004. A Bisimulation for Dynamic Sealing. In *ACM Symposium on Principles of Programming Languages (POPL), Venice, Italy*.

Asumu Takikawa, T. Stephen Strickland, Christos Dimoulas, Sam Tobin-Hochstadt, and Matthias Felleisen. 2012. Gradual typing for first-class classes *(ACM Symposium on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA))*.

Sam Tobin-Hochstadt and Matthias Felleisen. 2006. Interlanguage Migration: From Scripts to Programs. In *Dynamic Languages Symposium (DLS)*. 964–974.

Sam Tobin-Hochstadt and Matthias Felleisen. 2008. The Design and Implementation of Typed Scheme. In *ACM Symposium on Principles of Programming Languages (POPL), San Francisco, California*.

Sam Tobin-Hochstadt, Vincent St-Amour, Eric Dobson, and Asumu Takikawa. [n. d.]. Typed Racket Reference. https://docs.racket-lang.org/ts-reference/Typed_Units.html Accessed: 2019-10-30.

Matías Toro, Ronald Garcia, and Éric Tanter. 2018. Type-Driven Gradual Security with References. *ACM Transactions on Programming Languages and Systems* 40, 4 (Dec. 2018). http://doi.acm.org/10.1145/3229061

Matías Toro, Elizabeth Labrada, and Éric Tanter. 2019. Gradual Parametricity, Revisited. *Proc. ACM Program. Lang.* 3, POPL, Article 17 (Jan. 2019), 30 pages. https://doi.org/10.1145/3290330

Roger Wolff, Ronald Garcia, Éric Tanter, and Jonathan Aldrich. 2011. Gradual Typestate. In *Proceedings of the 25th European Conference on Object-oriented Programming (ECOOP'11)*.

Ningning Xie, Xuan Bi, and Bruno C. d. S. Oliveira. 2018. Consistent Subtyping for All. In *Programming Languages and Systems*, Amal Ahmed (Ed.). Springer International Publishing, Cham, 3–30.

$$\vdash \cdot \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, x : A} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, X} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, X \cong A}$$

$$\Gamma \vdash ? \qquad \frac{X \in \Gamma}{\Gamma \vdash X} \qquad \Gamma \vdash \mathbb{B} \qquad \frac{\Gamma \vdash A_1 \qquad \Gamma \vdash A_2}{\Gamma \vdash A_1 \times A_2} \qquad \frac{\Gamma \vdash A_1 \qquad \Gamma \vdash A_2}{\Gamma \vdash A_1 \to A_2} \qquad \frac{\Gamma, X \vdash A}{\Gamma \vdash \exists^\nu X.A}$$

$$\frac{\Gamma, X \vdash A}{\Gamma \vdash \forall^\nu X.A}$$

Fig. 19. Well-formedness of Environments, Types

$$\Gamma^\sqsubseteq \vdash ? : ? \sqsubseteq ? \qquad \Gamma^\sqsubseteq \vdash \mathbb{B} : \mathbb{B} \sqsubseteq \mathbb{B} \qquad \frac{X \in \Gamma^\sqsubseteq}{\Gamma^\sqsubseteq \vdash X : X \sqsubseteq X} \qquad \frac{\Gamma^\sqsubseteq \vdash A^\sqsubseteq : A \sqsubseteq G}{\Gamma^\sqsubseteq \vdash \mathrm{tag}_G(A^\sqsubseteq) : A \sqsubseteq ?}$$

$$\frac{\Gamma^\sqsubseteq, X \vdash A^\sqsubseteq : A_l \sqsubseteq A_r}{\Gamma^\sqsubseteq \vdash \forall^\nu X.A^\sqsubseteq : \forall^\nu X.A_l \sqsubseteq \forall^\nu X.A_r} \qquad \frac{\Gamma^\sqsubseteq, X \vdash A^\sqsubseteq : A_l \sqsubseteq A_r}{\Gamma^\sqsubseteq \vdash \exists^\nu X.A^\sqsubseteq : \exists^\nu X.A_l \sqsubseteq \exists^\nu X.A_r}$$

$$\frac{\Gamma^\sqsubseteq \vdash A^\sqsubseteq : A_l \sqsubseteq A_r \qquad \Gamma^\sqsubseteq \vdash B^\sqsubseteq : B_l \sqsubseteq B_r}{\Gamma^\sqsubseteq \vdash A^\sqsubseteq \to B^\sqsubseteq : A_l \to B_l \sqsubseteq A_r \to B_r} \qquad \frac{\Gamma^\sqsubseteq \vdash A^\sqsubseteq : A_l \sqsubseteq A_r \qquad \Gamma^\sqsubseteq \vdash B^\sqsubseteq : B_l \sqsubseteq B_r}{\Gamma^\sqsubseteq \vdash A^\sqsubseteq \times B^\sqsubseteq : A_l \times B_l \sqsubseteq A_r \times B_r}$$

$$\begin{aligned} \cdot|_{\Gamma'} &= \cdot \\ (X \cong A^\sqsubseteq, \Gamma^\sqsubseteq)|_{\Gamma'} &= X \cong A^\sqsubseteq, (\Gamma^\sqsubseteq|_{\Gamma')} && (\mathrm{FV}(A^\sqsubseteq) \cap \Gamma' = \emptyset) \\ (X \cong A^\sqsubseteq, \Gamma^\sqsubseteq)|_{\Gamma'} &= \Gamma^\sqsubseteq|_{\Gamma',X} && (\mathrm{FV}(A^\sqsubseteq) \cap \Gamma' \neq \emptyset) \end{aligned}$$

Fig. 20. Type Precision in a Precision Context, Restriction of a Precision Context

A SURFACE LANGUAGE

Well-formedness of environments and types is mutually defined:

B TYPE PRECISION

We present the definition of type precision with respect to a precision context in Figure 20.

With this presentation it is easy to see that

(1) There is at most one derivation of $\Gamma^\sqsubseteq \vdash A \sqsubseteq A'$
(2) ? is the most imprecise type.
(3) Precision is reflexive.
(4) Precision is transitive.

The latter 3 statements are actually ambiguous because we haven't said what $\Gamma^\sqsubseteq$ is in each situation. We don't use these operations in their full generality, but ok.

Lemma B.1. *If* $\Gamma^\sqsubseteq \vdash A^\sqsubseteq : A \sqsubseteq A'$ *and* $\Gamma^\sqsubseteq B^\sqsubseteq : A \sqsubseteq A'$ *then* $A^\sqsubseteq = B^\sqsubseteq$

$$
\begin{aligned}
A \sqcap ? &= A \\
? \sqcap B &= B \\
X \sqcap X &= X \\
\mathbb{B} \sqcap \mathbb{B} &= \mathbb{B} \\
(A_1 \times A_2) \sqcap (B_1 \times B_2) &= (A_1 \sqcap B_1) \times (A_2 \sqcap B_2) \\
(A_i \to A_o) \sqcap (B_i \to B_o) &= (A_i \sqcap B_i) \to (A_o \sqcap B_o) \\
(\exists^\nu X.A) \sqcap (\exists^\nu X.B) &= \exists^\nu X.(A \sqcap B) \\
(\forall^\nu X.A) \sqcap (\forall^\nu X.B) &= \forall^\nu X.(A \sqcap B)
\end{aligned}
$$

Fig. 21. Gradual Meets

Proof. By induction on derivations. If $A' \neq ?$, then only one rule applies and the proof follows by inductive hypotheses. If $A^{\sqsubseteq} = ? : ? \sqsubseteq ?$, the only other rule that could apply is $\mathrm{tag}_G(A^{\sqsubseteq})$, which cannot apply because it is easy to see that $? \sqsubseteq G$ does not hold for any G. Finally, we need to show that if $A^{\sqsubseteq} = \mathrm{tag}_G(A^{\sqsubseteq})$ and $B^{\sqsubseteq} = \mathrm{tag}_{G'}(A^{\sqsubseteq})$ then $G = G'$ because it is easy to see if $A \sqsubseteq G$ and $A \sqsubseteq G'$ then $G = G'$. □

This is "identity expansion", the derivations are used in the parametricity theorem.

Lemma B.2. *If $\Gamma \vdash A$ is a well-formed type then $\Gamma \vdash A : A \sqsubseteq A$.*

Proof. By induction over A. Every type constructor is punned with its type precision constructor. □

This is "cut elimination":

Lemma B.3. *If $\Gamma^{\sqsubseteq} \vdash AB^{\sqsubseteq} : A \sqsubseteq B$ and $\Gamma^{\sqsubseteq} \vdash BC^{\sqsubseteq} : B \sqsubseteq C$ then we can construct a proof $\Gamma^{\sqsubseteq} \vdash AC^{\sqsubseteq} : A \sqsubseteq C$.*

Proof. By induction on $BC^{\sqsubseteq}$.

(1) If $BC^{\sqsubseteq} = ? : ? \sqsubseteq ?$, then the proof is just $AB^{\sqsubseteq}$
(2) If $BC^{\sqsubseteq} = \mathrm{tag}_G(BG^{\sqsubseteq})$, then $BG^{\sqsubseteq} : B \sqsubseteq G$ so by inductive hypothesis, there is a proof $AG^{\sqsubseteq} : A \sqsubseteq G$ and the proof we need is $\mathrm{tag}_G(AG^{\sqsubseteq})$.
(3) If $BC^{\sqsubseteq} = BC^{\sqsubseteq}_1 \times BC^{\sqsubseteq}_2$, then it must also be the case that $AB^{\sqsubseteq} = AB^{\sqsubseteq}_1 \times AB^{\sqsubseteq}_2$, and then our result is $AC^{\sqsubseteq}_1 \times AC^{\sqsubseteq}_2$ where $AC^{\sqsubseteq}_1, AC^{\sqsubseteq}_2$ come from the inductive hypothesis.
(4) All other cases are analogous to the product.

□

Next, we (partially) define gradual meets $A \sqcap B$ in Figure 21. The meet is undefined if the case is missing.

Lemma B.4. *For every $\Gamma \vdash A, B$, $\Gamma \vdash A \sqcap B$ and there are precision derivations*

(1) $\Gamma \vdash A\sqcap^{\sqsubseteq} : A \sqcap B \sqsubseteq A$
(2) $\Gamma \vdash B\sqcap^{\sqsubseteq} : A \sqcap B \sqsubseteq B$

Such that for any $\Gamma \vdash C$ with $\Gamma \vdash CA^{\sqsubseteq} : C \sqsubseteq A$ and $\Gamma \vdash CB^{\sqsubseteq} : C \sqsubseteq B$, there exists a derivation

$$\Gamma \vdash C\sqcap^{\sqsubseteq} : C \sqsubseteq A \sqcap B$$

$$
\begin{array}{lll}
E[\mho] & \mapsto & \mho \text{ where } E \neq [\,] \\
E[\mathtt{unseal}_\sigma(\mathtt{seal}_\sigma V)] & \mapsto & E[V] \\
E[\mathtt{is}(G)?\ (\mathtt{inj}_G\ V)] & \mapsto & E[\mathtt{true}] \\
E[\mathtt{is}(G)?\ (\mathtt{inj}_H\ V)] & \mapsto & E[\mathtt{false}] \text{ where } G \neq H \\
\Sigma \triangleright E[\mathsf{hide}\ X \cong A; M] & \mapsto & \Sigma, \sigma : A \triangleright E[M[\sigma/X]] \\
\Sigma \triangleright E\left[\begin{array}{l}\mathsf{unpack}\ (X, x) = \mathsf{pack}^\nu(X \cong A', [\overline{A^\sqsubseteq \updownarrow}], M); \\ N\end{array}\right] & \mapsto & \Sigma, \sigma : A' \triangleright E\left[\begin{array}{l}\mathtt{let}\ x = \overline{\langle A^\sqsubseteq[\sigma/X]\rangle\updownarrow}\, M[\sigma/X]; \\ N[\sigma/X]\end{array}\right] \\
E\,[\mathsf{pack}^\nu(X \cong A, M)] & \mapsto & E[\mathsf{pack}^\nu(X \cong A', [\,], M)] \\
E[(\Lambda^\nu X.M)\{\sigma \cong A\}] & \mapsto & E[M[\sigma/X]] \\
E[(\lambda(x : A).M)\ V] & \mapsto & E[M[V/x]] \\
E[\mathtt{if\ true}\,\mathsf{then}\ M_1\,\mathsf{else}\ M_2] & \mapsto & E[M_1] \\
E[\mathtt{if\ false}\,\mathsf{then}\ M_1\,\mathsf{else}\ M_2] & \mapsto & E[M_2] \\
E[\mathtt{let}\ (x, y)\ =\ (V_1, V_2); M] & \mapsto & E[M[V_1/x][V_2/y]] \\
E[\mathtt{let}\ x = V; M] & \mapsto & E[M[V/x]] \\
E[\langle A^\sqsubseteq\rangle\updownarrow V] & \mapsto & E[V] \text{ where } A^\sqsubseteq \in \{\mathbb{B}, \sigma, ?\} \\
E[\langle A_1^\sqsubseteq \times A_2^\sqsubseteq\rangle\updownarrow (V_1, V_2)] & \mapsto & E[(\langle A_1^\sqsubseteq\rangle\updownarrow V_1, \langle A_2^\sqsubseteq\rangle\updownarrow V_2)] \\
E[(\langle A_1^\sqsubseteq \to A_2^\sqsubseteq\rangle\updownarrow V_1)\ V_2] & \mapsto & E[\langle A_2^\sqsubseteq\rangle\updownarrow (V_1\ \langle A_1^\sqsubseteq\rangle\updownarrow^-\ V_2)] \\
E[\langle \exists^\nu X.A^\sqsubseteq\rangle\updownarrow \mathsf{pack}^\nu(X \cong A', [A'^\sqsubseteq \updownarrow', \ldots], M)] & \mapsto & E[\mathsf{pack}^\nu(X \cong A', [A^\sqsubseteq \updownarrow, A'^\sqsubseteq \updownarrow', \ldots], M)] \\
E[(\langle \forall^\nu X.A^\sqsubseteq\rangle\updownarrow V)\{\sigma \cong A\}] & \mapsto & E[\langle A^\sqsubseteq[\sigma/X]\rangle\updownarrow (V\{\sigma \cong A\})] \\
E[\langle \mathrm{tag}_G(A^\sqsubseteq)\rangle\uparrow V] & \mapsto & E[\mathtt{inj}_G\ \langle A^\sqsubseteq\rangle\uparrow V] \\
E[\langle \mathrm{tag}_G(A^\sqsubseteq)\rangle\downarrow \mathtt{inj}_G\ V] & \mapsto & E[\langle A^\sqsubseteq\rangle\downarrow V] \\
E[\langle \mathrm{tag}_G(A^\sqsubseteq)\rangle\downarrow \mathtt{inj}_H\ V] & \mapsto & \mho \text{ where } H \neq G
\end{array}
$$

Fig. 22. PolyC$^\nu$ Dynamic Semantics (full)

C CAST CALCULUS

We present the full cast calculus operational semantics in Figure 22.

Some of the semantics in Figure 7 involve terms with σs in places we would expect Xs, in particular instantiations, seals, and unseals. These forms come about when we evaluate a `hide` and substitute σ for X. We also have our aforementioned intermediate form for pack casts. Figure 23 gives the static typing rules for runtime terms. Note that the typing of runtime terms depends on a given Σ. To reason about well-typed terms at runtime, we also thread a store through the rules in Figure 5.

We prove a few standard operational lemmas.

Lemma C.1 (Unique Decomposition). *If $\Sigma_1; \cdot \vdash M_1 : A$, then there exist unique E, M_2 such that $M_1 = E[M_2]$.*

Proof. By induction on M_1. □

Lemma C.2 (Cast calculus dynamic semantics are deterministic). *If $\Sigma \triangleright M \mapsto \Sigma_1 \triangleright M_1$ and $\Sigma \triangleright M \mapsto \Sigma_2 \triangleright M_2$ then $\Sigma_1 = \Sigma_2$ and $M_1 = M_2$.*

Proof. By unique decomposition and definition of the dynamic semantics. □

Lemma C.3 (Progress). *If $\Sigma_1; \cdot \vdash M_1 : A$ then either $\Sigma_1 \triangleright M_1 \mapsto \Sigma_2 \triangleright M_2$, $M_1 = \mho$, or $M_1 = V$ for some V.*

$$\frac{x : A \in \Gamma}{\Sigma; \Gamma \vdash x : A; \cdot} \qquad \frac{\Sigma; \Gamma \vdash M : A_l; \Gamma_M \qquad \Sigma; \Gamma \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Sigma; \Gamma \vdash \langle A^{\sqsubseteq} \rangle\!\uparrow M : A_r; \Gamma_M} \qquad \frac{\Sigma; \Gamma \vdash M : A_r; \Gamma_M \qquad \Sigma; \Gamma \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Sigma; \Gamma \vdash \langle A^{\sqsubseteq} \rangle\!\downarrow M : A_l; \Gamma_M}$$

$$\frac{\Sigma; \Gamma \vdash M : A; \Gamma_M \qquad \Sigma; \Gamma, \Gamma_M, x : A \vdash N : B; \Gamma_N}{\Sigma; \Gamma \vdash \texttt{let } x = M; N : B; \Gamma_M, \Gamma_N} \qquad \frac{\Sigma; \Gamma \vdash M : A; \Gamma_o \qquad X \cong A \in \Gamma, \Gamma_o}{\Sigma; \Gamma \vdash \texttt{seal}_X M : X; \Gamma_o}$$

$$\frac{\Sigma; \Gamma \vdash M : X; \Gamma_o \qquad X \cong A \in \Gamma, \Gamma_o}{\Sigma; \Gamma \vdash \texttt{unseal}_X M : A; \Gamma_o} \qquad \frac{\Sigma; \Gamma \vdash M : ?; \Gamma_o \qquad \Sigma; \Gamma \vdash G}{\Sigma; \Gamma \vdash \texttt{is}(G)?\ M : \mathbb{B}; \Gamma_o} \qquad \Sigma; \Gamma \vdash \texttt{true} : \mathbb{B}; \cdot$$

$$\Sigma; \Gamma \vdash \texttt{false} : \mathbb{B}; \cdot \qquad \frac{\Sigma; \Gamma \vdash M : \mathbb{B}; \Gamma_M \qquad \Sigma; \Gamma, \Gamma_M \vdash N_t : B; \Gamma_N \qquad \Sigma; \Gamma, \Gamma_M \vdash N_f : B; \Gamma_N}{\Sigma; \Gamma \vdash \texttt{if } M \texttt{ then } N_t \texttt{ else } N_f : B; \Gamma_M, \Gamma_N}$$

$$\frac{\Sigma; \Gamma \vdash M_1 : A_1; \Gamma_1 \qquad \Sigma; \Gamma, \Gamma_1 \vdash M_2 : A_2; \Gamma_2}{\Sigma; \Gamma \vdash (M_1, M_2) : A_1 \times A_2; \Gamma_1, \Gamma_2}$$

$$\frac{\Sigma; \Gamma \vdash M : A_1 \times A_2; \Gamma_M \qquad \Sigma; \Gamma, \Gamma_M, x : A_1, y : A_2 \vdash N : B; \Gamma_N}{\Sigma; \Gamma \vdash \texttt{let } (x, y) = M; N : B; \Gamma_M, \Gamma_N} \qquad \frac{\Sigma; \Gamma \vdash A \qquad \Sigma; \Gamma, x : A \vdash M : B; \cdot}{\Sigma; \Gamma \vdash \lambda x : A.M : A \to B; \cdot}$$

$$\frac{\Sigma; \Gamma \vdash M : A \to B; \Gamma_M \qquad \Sigma; \Gamma, \Gamma_M \vdash N : A; \Gamma_N}{\Sigma; \Gamma \vdash M\ N : B; \Gamma_M, \Gamma_N} \qquad \frac{\Sigma; \Gamma, X \cong A \vdash M : B; \cdot}{\Sigma; \Gamma \vdash \texttt{pack}^\nu(X \cong A, M) : \exists^\nu X.B; \cdot}$$

$$\frac{\Sigma; \Gamma \vdash M : \exists^\nu X.A; \Gamma_M \qquad \Sigma; \Gamma, \Gamma_M, X, x : A \vdash N : B; \Gamma_N \qquad \Sigma; \Gamma, \Gamma_M \vdash \Gamma_N \qquad \Sigma; \Gamma, \Gamma_M, \Gamma_N \vdash B}{\Sigma; \Gamma \vdash \texttt{unpack } (X, x) = M; N : B; \Gamma_M, \Gamma_N}$$

$$\frac{\Sigma; \Gamma, X \vdash M : A; \cdot}{\Sigma; \Gamma \vdash \Lambda^\nu X.M : \forall^\nu X.A; \cdot} \qquad \frac{\Sigma; \Gamma \vdash M : \forall^\nu X.A; \Gamma_M \qquad \Sigma; \Gamma, \Gamma_M \vdash B}{\Sigma; \Gamma \vdash M\{X \cong B\} : A; \Gamma_M, X \cong B}$$

$$\frac{\Sigma; \Gamma \vdash M : \Gamma_M, X \cong A, \Gamma'_M \qquad \Sigma; \Gamma, \Gamma_M \vdash \Gamma'_M}{\Sigma; \Gamma \vdash \texttt{hide } X \cong A; M; \Gamma_M, \Gamma'_M} \qquad \frac{\Sigma; \Gamma \vdash M : A; \Gamma_o \qquad \sigma : A \in \Sigma}{\Sigma; \Gamma \vdash \texttt{seal}_\sigma M : \sigma; \Gamma_o}$$

$$\frac{\Sigma; \Gamma \vdash M : \sigma; \Gamma_o \qquad \sigma : A \in \Sigma}{\Sigma; \Gamma \vdash \texttt{unseal}_\sigma M : A; \Gamma_o} \qquad \frac{\Sigma; \Gamma \vdash M : \forall^\nu X.A; \Gamma_M \qquad \sigma : A \in \Sigma}{\Sigma; \Gamma \vdash M\{\sigma \cong B\} : A; \Gamma_M}$$

$$\frac{\Sigma; \Gamma, X \cong A' \vdash \langle A^{\sqsubseteq} \rangle\!\updownarrow \ldots M : B; \cdot}{\Sigma; \Gamma \vdash \texttt{pack}^\nu(X \cong A', [A^{\sqsubseteq} \updownarrow \ldots], M) : \exists^\nu X.B; \cdot}$$

Fig. 23. PolyC$^\nu$ Typing for Runtime Forms

Proof. If $M_1 = \mho$ or M_1 is a value, we conclude. Otherwise, by Lemma C.1 (unique decomposition), we have unique E, M_3 such that $M_1 = E[M_3]$. We proceed by cases on M_3.

- $M_3 = \mho$ Since $M_1 \neq \mho$, we have $E \neq [\,]$, so $M_1 \mapsto \mho$ and we conclude.
- $M_3 = \texttt{hide } X \cong B; M_4$ Then $\Sigma_1 \triangleright M_3 \mapsto \Sigma_1, \sigma : B \triangleright M_4[\sigma/X]$, so we conclude since $M_2 \mapsto E[M_4[\sigma/X]]$.
- $M_3 = \texttt{if } V \texttt{ then } M_4 \texttt{ else } M_5$ Since M_1 is well-typed, $V : \mathbb{B}$ and so, by Lemma F.9 (canonical forms), which we may use thanks to Lemma F.2, either $V = \texttt{true}$ or $V = \texttt{false}$. In the first case, $M_3 \mapsto M_4$ and in the second $M_3 \mapsto M_5$, so we conclude.

- $M_3 = \texttt{let}\ x = V; M_4$ Then $M_3 \mapsto M_4[V/x]$, so we conclude.
- $M_3 = \texttt{let}\ (x, y) = V; M_4$ Since M_1 is well-typed, $V : A_1 \times A_2$ for some A_1, A_2 and so, by Lemma F.9 (canonical forms), $V = (V_1, V_2)$ for some V_1, V_2. Then $M_3 \mapsto M_4[V_1/x][V_2/y]$, so we conclude.
- $M_3 = \texttt{pack}^\nu(X \cong B, M_4)$ Then $M_3 \mapsto \texttt{pack}^\nu(X \cong B, [], M_4)$, so we conclude.
- $M_3 = \texttt{unpack}\ (X, x) = V; M_4$ Since M_1 is well-typed, $V : \exists^\nu X B$ for some B and so, by Lemma F.9 (canonical forms), $V = \texttt{pack}^\nu(X \cong B, [], M_5)$ for some M_5. Then $\Sigma \triangleright M_3 \mapsto \Sigma, \sigma : B \triangleright \texttt{let}\ x = M_5[\sigma/X]; M_4[\sigma/X]$, so we conclude.
- $M_3 = V\{\sigma \cong B\}$ Since M_1 is well-typed, $V : \forall^\nu X A_1$ for some A_1 and so, by Lemma F.9 (canonical forms), $V = \Lambda^\nu X.M_4$ for some M_4 or $V = \langle \forall^\nu X.A^{\sqsubseteq} \rangle\updownarrow V_1$ for some $A^{\sqsubseteq}, V_1$. In the first case, $M_3 \mapsto M_4[\sigma/X]$ and in the second, $M_3 \mapsto \langle A^{\sqsubseteq}[\sigma/X] \rangle\updownarrow (V_1\{\sigma \cong B\})$.
- $M_3 = V_1\ V_2$ Since M_1 is well-typed, $V : A_1 \to A_2$ for some A_1, A_2 and so, by Lemma F.9 (canonical forms), $V_1 = \lambda x : A_1.M_4$ for some M_4 or $V = \langle A_1^{\sqsubseteq} \to A_2^{\sqsubseteq} \rangle\updownarrow V_1$ for some $A_1^{\sqsubseteq}, A_2^{\sqsubseteq}, V_1$. In the first case, $M_3 \mapsto M_4[V_2/x]$ and in the second, $M_3 \mapsto \langle A_2^{\sqsubseteq} \rangle\updownarrow (V_1\ \langle A_1^{\sqsubseteq} \rangle\updownarrow^-\ V_2)$.
- $M_3 = \texttt{unseal}_\sigma V$ Since M_1 is well-typed, $V : \sigma$ and so, by Lemma F.9 (canonical forms), $V = \texttt{seal}_\sigma V_1$ for some V_1. Then $M_3 \mapsto V_1$, so we conclude.
- $M_3 = \texttt{is}(G)?\ V$ Since M_1 is well-typed, $V : ?$ and so, by Lemma F.9 (canonical forms), $V = \texttt{inj}_H\ V_1$ for some H, V_1. If $H = G$, then $M_3 \mapsto \texttt{true}$, otherwise $M_3 \mapsto \texttt{false}$.
- $M_3 = \langle \exists^\nu X.A^{\sqsubseteq} \rangle\updownarrow V$ Since M_1 is well-typed, $V : \exists^\nu X.A_1$ for some A_1 and so, by Lemma F.9 (canonical forms), $V = \texttt{pack}^\nu(X \cong B, [A_2^{\sqsubseteq} \updownarrow_2 \ldots], M_4)$ for some $B, A_2^{\sqsubseteq} \updownarrow_2 \ldots, M_4$. Then $M_3 \mapsto \texttt{pack}^\nu(X \cong A_2, [A^{\sqsubseteq} \updownarrow, A_2^{\sqsubseteq} \updownarrow_2 \ldots], M_4)$.
- $M_3 = \langle A_1^{\sqsubseteq} \times A_2^{\sqsubseteq} \rangle\updownarrow V$ Since M_1 is well-typed, $V : B_1 \times B_2$ for some B_1, B_2 and so, by Lemma F.9 (canonical forms), $V = (V_1, V_2)$ for some V_1, V_2. Then $M_3 \mapsto (\langle A_1^{\sqsubseteq} \rangle\updownarrow V_1, \langle A_2^{\sqsubseteq} \rangle\updownarrow V_2)$.
- $M_3 = \langle A^{\sqsubseteq} \rangle\updownarrow V$ where $A^{\sqsubseteq} \in \{\mathbb{B}, \sigma, ?\}$ Then $M_3 \mapsto V$.
- $M_3 = \langle \mathrm{tag}_G(A^{\sqsubseteq}) \rangle\uparrow V$ Then $M_3 \mapsto \texttt{inj}_G\ \langle A^{\sqsubseteq} \rangle\uparrow V$.
- $M_3 = \langle \mathrm{tag}_G(A^{\sqsubseteq}) \rangle\downarrow V$ Since M_1 is well-typed, $V : ?$ and so, by Lemma F.9 (canonical forms), $V = \texttt{inj}_H\ V_1$ for some H, V_1. If $H = G$, then $M_3 \mapsto \langle A^{\sqsubseteq} \rangle\downarrow V$. Otherwise, $M_3 \mapsto \mho$.

□

D CBPV

Definition D.1 (Preamble store, substitution). We name the store generated by the preamble Σ_p, defined as

$$\begin{array}{rcl} \Sigma_p & = & (4, f) \\ f(0) & = & \mathbb{B} \\ f(1) & = & U(\mathrm{OSum} \to F\mathrm{OSum}) \\ f(2) & = & \mathrm{OSum} \times \mathrm{OSum} \\ f(3) & = & \exists X.U(\mathrm{Case}\ X \to F\mathrm{OSum}) \\ f(4) & = & U(\forall X.\mathrm{Case}\ X \to F\mathrm{OSum}) \end{array}$$

We define γ_p to be a substitution that closes terms with respect to Γ_p using the store Σ_p:

$$\begin{array}{rcl} \gamma_p(c_{\mathrm{Bool}}) & = & 0 \\ \gamma_p(c_{\mathrm{Fun}}) & = & 1 \\ \gamma_p(c_{\mathrm{Times}}) & = & 2 \\ \gamma_p(c_{\mathrm{Ex}}) & = & 3 \\ \gamma_p(c_{\mathrm{All}}) & = & 4 \end{array}$$

$$\frac{\Sigma;\Delta;\Gamma \vdash V_T : \text{Case } A \qquad \Sigma;\Delta;\Gamma \vdash V : \text{OSum} \qquad \Sigma;\Delta;\Gamma, x : A \mid \cdot \vdash M : B \qquad \Sigma;\Delta;\Gamma \mid \cdot \vdash N : B}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \texttt{match } V_T \texttt{ with } V\{\texttt{inj } x.M \mid N\} : B}$$

$$\frac{\sigma : A \in \Sigma \qquad \Delta \vdash A}{\Sigma;\Delta;\Gamma \vdash \sigma : \text{Case } A} \qquad \frac{\Delta \vdash A \qquad \Sigma;\Delta;\Gamma, x : \text{Case } A \mid \cdot \vdash M : B}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \texttt{newcase}_A\ x; M : B}$$

$$\frac{\Sigma;\Delta;\Gamma \vdash V_T : \text{Case } A \qquad \Sigma;\Delta;\Gamma \vdash V : A}{\Sigma;\Delta;\Gamma \vdash \texttt{inj}_{V_T}\ V : \text{OSum}} \qquad \frac{\Sigma;\Delta, X;\Gamma \mid \cdot \vdash M : B}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \Lambda X.M : \forall X.B}$$

$$\frac{\Sigma;\Delta;\Gamma \mid \Theta \vdash M : \forall X.B \qquad \Delta \vdash A}{\Sigma;\Delta;\Gamma \mid \Theta \vdash M[A] : B[A/X]} \qquad \frac{\Sigma;\Delta;\Gamma \vdash V : A[A'/X]}{\Sigma;\Delta;\Gamma \vdash \texttt{pack}(A', V) \texttt{ as } \exists X.A : \exists X.A}$$

$$\frac{\Sigma;\Delta;\Gamma \vdash V : \exists X.A \qquad \Delta \vdash B \qquad \Sigma;\Delta, X;\Gamma, x : A \mid \cdot \vdash M : B}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \texttt{unpack } (X, x) = V; M : B} \qquad \frac{}{\Sigma;\Delta;\Gamma, x : A, \Gamma' \vdash x : A}$$

$$\frac{}{\Sigma;\Delta;\Gamma \mid \bullet : B \vdash \bullet : B} \qquad \frac{\Sigma;\Delta;\Gamma \vdash V : \mathbb{B} \qquad \Sigma;\Delta;\Gamma \mid \cdot \vdash M_1 : B \qquad \Sigma;\Delta;\Gamma \mid \cdot \vdash M_2 : B}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \texttt{if } V \texttt{ then } M_1 \texttt{ else } M_2 : B}$$

$$\frac{}{\Sigma;\Delta;\Gamma \vdash \texttt{true} : \mathbb{B}} \qquad \frac{}{\Sigma;\Delta;\Gamma \vdash \texttt{false} : \mathbb{B}} \qquad \frac{\Sigma;\Delta;\Gamma \vdash V_1 : A_1 \qquad \Sigma;\Delta;\Gamma \vdash V_2 : A_2}{\Sigma;\Delta;\Gamma \vdash (V_1, V_2) : A_1 \times A_2}$$

$$\frac{\Sigma;\Delta;\Gamma \vdash V : A_1 \times A_2 \qquad \Sigma;\Delta;\Gamma, x : A_1, y : A_2 \mid \cdot \vdash M : B}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \texttt{let } (x, y) = V; M : B} \qquad \frac{\Sigma;\Delta;\Gamma \mid \cdot \vdash M : B}{\Sigma;\Delta;\Gamma \vdash \texttt{thunk } M : UB}$$

$$\frac{\Sigma;\Delta;\Gamma \vdash V : UB}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \texttt{force } V : B} \qquad \frac{\Sigma;\Delta;\Gamma \vdash V : A}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \texttt{ret } V : FA}$$

$$\frac{\Sigma;\Delta;\Gamma \mid \Theta \vdash M : FA \qquad \Sigma;\Delta;\Gamma \mid \cdot \vdash N : B}{\Sigma;\Delta;\Gamma \mid \Theta \vdash x \leftarrow M; N : B} \qquad \frac{\Sigma;\Delta;\Gamma \mid \Theta \vdash M : A \to B \qquad \Sigma;\Delta;\Gamma \vdash V : A}{\Sigma;\Delta;\Gamma \mid \Theta \vdash M\ V : B}$$

$$\frac{\Delta \vdash A \qquad \Sigma;\Delta;\Gamma, x : A \mid \cdot \vdash V : B}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \lambda x : A.V : A \to B} \qquad \frac{}{\Sigma;\Delta;\Gamma \mid \cdot \vdash \mho : B}$$

Fig. 24. $\text{CBPV}_{\text{OSum}}$ Type System

We give the full translation from PolyC$^\nu$ to $\text{CBPV}_{\text{OSum}}$ in Figure 26. Added to the cases from the paper are the translation for types $\mathbb{B}$ and $A \times B$.

E TERM PRECISION

Figure 29 shows the full definition of term precision for PolyG$^\nu$. Figures 30 and 31 shows the full definition of term precision for PolyC$^\nu$.

Lemma E.1 (Casts are Monotone). *If $A^{\sqsubseteq} : A_l \sqsubseteq A_r$ and $AB_l^{\sqsubseteq} : A_l \sqsubseteq B_l$ and $AB_r^{\sqsubseteq} : A_r \sqsubseteq B_r$ and $B^{\sqsubseteq} : B_l \sqsubseteq B_r$, then*

$$
\begin{aligned}
S[\mho] &\mapsto \mho \\
\Sigma \triangleright S[\mathtt{newcase}_A\ x; M] &\mapsto \Sigma, \sigma : A \triangleright S[M[\sigma/x]] \\
S[\mathtt{match}\ \mathtt{inj}_\sigma\ V\ \mathtt{with}\ \sigma\{\mathtt{inj}\ x.M \mid N\}] &\mapsto S[M[V/x]] \\
S[\mathtt{match}\ \mathtt{inj}_{\sigma_1}\ V\ \mathtt{with}\ \sigma_2\{\mathtt{inj}\ x.M \mid N\}] &\mapsto S[N] \qquad \text{(where } \sigma_1 \neq \sigma_2\text{)} \\
S[\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ M\ \mathtt{else}\ N] &\mapsto S[M] \\
S[\mathtt{if}\ \mathtt{false}\ \mathtt{then}\ M\ \mathtt{else}\ N] &\mapsto S[N] \\
S[\mathtt{let}\ (x, y) = (V_1, V_2); M] &\mapsto S[M[V_1/x, V_2/y]] \\
S[\mathtt{force}\ (\mathtt{thunk}\ M)] &\mapsto S[M] \\
S[\mathtt{unpack}\ (X, x) = \mathtt{pack}(A, V); M] &\mapsto S[M[A/X, V/x]] \\
S[(\Lambda X.M)[A]] &\mapsto S[M[A/X]] \\
S[(\lambda(x : A).M)\ V] &\mapsto S[M[V/x]] \\
S[x \leftarrow \mathtt{ret}\ V; N] &\mapsto S[N[V/x]]
\end{aligned}
$$

Fig. 25. $\mathrm{CBPV}_{\mathrm{OSum}}$ Operational Semantics

$$
\begin{array}{lcl}
[\![x]\!] & = & \mathtt{ret}\ x \\
[\![\mathtt{let}\ x = M; N]\!] & = & x \leftarrow [\![M]\!]; [\![N]\!] \\
[\![\mho_A]\!] & = & \mho \\
[\![\mathtt{seal}_\alpha M]\!] & = & [\![M]\!] \\
[\![\mathtt{unseal}_\alpha M]\!] & = & [\![M]\!] \\
[\![\mathtt{inj}_G\ M]\!] & = & r \leftarrow [\![M]\!]; \mathtt{ret}\ \mathtt{inj}_{\mathrm{case}(G)}\ r \\
[\![\mathtt{is}(G)?\ M]\!] & = & r \leftarrow [\![M]\!]; \\
 & & \mathtt{match}\ r\ \mathtt{with}\ \mathrm{case}(G)\{\mathtt{inj}\ y.\mathtt{ret}\ \mathtt{true} \mid \mathtt{ret}\ \mathtt{false}\} \\
[\![\mathtt{hide}\ X \cong A; M]\!] & = & \mathtt{newcase}_{[\![A]\!]}\ c_X; [\![M]\!] \\
[\![\langle A^{\sqsubseteq}\rangle\updownarrow M]\!] & = & [\![A^{\sqsubseteq}]\!]\updownarrow[[\![M]\!]] \\
[\![\mathtt{true}]\!] & = & \mathtt{ret}\ \mathtt{true} \\
[\![\mathtt{false}]\!] & = & \mathtt{ret}\ \mathtt{false} \\
[\![\mathtt{if}\ M_1\ \mathtt{then}\ M_2\ \mathtt{else}\ M_3]\!] & = & r \leftarrow [\![M_1]\!]; \mathtt{if}\ r\ \mathtt{then}\ [\![M_2]\!]\ \mathtt{else}\ [\![M_3]\!] \\
[\![(M_1, M_2)]\!] & = & x_1 \leftarrow [\![M_1]\!]; x_2 \leftarrow [\![M_2]\!]; \mathtt{ret}\ (x_1, x_2) \\
[\![\mathtt{let}\ (x, y) = M; N]\!] & = & r \leftarrow [\![M]\!]; \mathtt{let}\ (x, y) = r; [\![N]\!] \\
[\![\mathtt{pack}^\nu(X \cong A, M)]\!] & = & \mathtt{ret}\ \mathtt{pack}(A, \mathtt{thunk}\ (\lambda c_X : \mathrm{Case}\ A.[\![M]\!])) \\
[\![\mathtt{pack}^\nu(X \cong A', [A_n^{\sqsubseteq} \updownarrow_n \dots A_1^{\sqsubseteq} \updownarrow_1], M)]\!] & = & \mathtt{ret}\ \mathtt{pack}(A, \mathtt{thunk}\ (\lambda c_X : \mathrm{Case}\ A.M_n')) \\
 & & \text{where } M_0' = [\![M]\!] \\
 & & \text{and } M_{i+1}' = [\![A_{i+1}^{\sqsubseteq}]\!]\updownarrow_{i+1}[\mathtt{force}\ (\mathtt{thunk}\ (\lambda c_X : \mathrm{Case}\ A'.[\![M_i]\!]))\ c_X] \\
[\![\mathtt{unpack}\ (X, x) = M; N]\!] & = & r \leftarrow [\![M]\!]; \mathtt{unpack}\ (X, f) = r; \\
 & & \mathtt{newcase}_X\ c_X; x \leftarrow (\mathtt{force}\ f)\ c_X; [\![N]\!] \\
[\![\Lambda^\nu X.M]\!] & = & \mathtt{ret}\ (\mathtt{thunk}\ (\Lambda X.\lambda(c_X : \mathrm{Case}\ X).[\![M]\!])) \\
[\![M\{\alpha \cong A\}]\!] & = & f \leftarrow [\![M]\!]; (\mathtt{force}\ f)[A]\ (\mathrm{case}(\alpha)) \\
[\![\lambda(x : A).M]\!] & = & \mathtt{ret}\ \mathtt{thunk}\ \lambda(x : [\![A]\!]).[\![M]\!] \\
[\![M\ N]\!] & = & f \leftarrow [\![M]\!]; a \leftarrow [\![N]\!]; (\mathtt{force}\ f)\ a
\end{array}
$$

Fig. 26. PolyC^ν term translation

$$\cdot : \cdot \sqsubseteq \cdot \qquad \frac{\Gamma^{\sqsubseteq} : \Gamma_l \sqsubseteq \Gamma_r \qquad \Gamma^{\sqsubseteq} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{(\Gamma^{\sqsubseteq}, x : A^{\sqsubseteq}) : \Gamma_l, x : A_l \sqsubseteq \Gamma_r, x : A_r} \qquad \frac{\Gamma^{\sqsubseteq} : \Gamma_l \sqsubseteq \Gamma_r \qquad \Gamma^{\sqsubseteq} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{(\Gamma^{\sqsubseteq}, X \cong A^{\sqsubseteq}) : \Gamma_l, X \cong A_l \sqsubseteq \Gamma_r, X \cong A_r}$$

$$\frac{\Gamma^{\sqsubseteq} : \Gamma_l \sqsubseteq \Gamma_r}{(\Gamma^{\sqsubseteq}, X) : \Gamma_l, X \sqsubseteq \Gamma_r, X}$$

Fig. 27. Type Precision Contexts

$$\begin{aligned}
\mathrm{cod}(A^{\sqsubseteq} \to B^{\sqsubseteq}) &= B^{\sqsubseteq}\\
\mathrm{cod}(?) &= ?\\
\mathrm{cod}(\mathrm{tag}_G(A^{\sqsubseteq} \to B^{\sqsubseteq})) &= B^{\sqsubseteq}\\
\pi_i(A_0^{\sqsubseteq} \times A_1^{\sqsubseteq}) &= A_i^{\sqsubseteq}\\
\pi_i(?) &= ?\\
\pi_i(\mathrm{tag}_{?\times ?}(A_0^{\sqsubseteq} \times A_1^{\sqsubseteq})) &= A_i^{\sqsubseteq}\\
\\
\mathrm{un}\forall^{\nu}(\forall^{\nu}X.A^{\sqsubseteq}) &= A^{\sqsubseteq}\\
\mathrm{un}\forall^{\nu}(?) &= ?\\
\mathrm{un}\forall^{\nu}(\mathrm{tag}_{\forall^{\nu}X.?}(\forall^{\nu}X.A^{\sqsubseteq})) &= A^{\sqsubseteq}\\
\mathrm{un}\exists^{\nu}(\exists^{\nu}X.A^{\sqsubseteq}) &= A^{\sqsubseteq}\\
\mathrm{un}\exists^{\nu}(?) &= ?\\
\mathrm{un}\exists^{\nu}(\mathrm{tag}_{\exists^{\nu}X.?}(\exists^{\nu}X.A^{\sqsubseteq})) &= A^{\sqsubseteq}
\end{aligned}$$

Fig. 28. Metafunctions extended to type precision derivations

(1) *If* $\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}$, *then* $\Gamma^{\sqsubseteq} \vdash \langle AB_l^{\sqsubseteq} \rangle\!\!\upharpoonleft M_l \sqsubseteq \langle AB_r^{\sqsubseteq} \rangle\!\!\upharpoonleft M_r : B^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}$
(2) *If* $\Gamma^{\sqsubseteq} \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}$, *then* $\Gamma^{\sqsubseteq} \vdash \langle c \rangle\!\!\downharpoonleft insertsthat AB_l^{\sqsubseteq} N_l \sqsubseteq \langle AB_r^{\sqsubseteq} \rangle\!\!\downharpoonleft N_r : A^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}$

Proof. (1) By the following derivation

$$\dfrac{\dfrac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}}{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq \langle AB_r^{\sqsubseteq} \rangle\!\!\upharpoonleft M_r : AB_{lr}^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}}}{\Gamma^{\sqsubseteq} \vdash \langle AB_l^{\sqsubseteq} \rangle\!\!\upharpoonleft M_l \sqsubseteq \langle AB_r^{\sqsubseteq} \rangle\!\!\upharpoonleft M_r : B^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}}$$

Where $AB_{lr}^{\sqsubseteq} : A_l \sqsubseteq B_r$, which exists by transitivity lemma B.3.

(2) By the following derivation

$$\dfrac{\dfrac{\Gamma^{\sqsubseteq} \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}}{\Gamma^{\sqsubseteq} \vdash \langle AB_l^{\sqsubseteq} \rangle\!\!\downharpoonleft N_l \sqsubseteq N_r : AB_{lr}^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}}}{\Gamma^{\sqsubseteq} \vdash \langle AB_l^{\sqsubseteq} \rangle\!\!\downharpoonleft N_l \sqsubseteq \langle AB_r^{\sqsubseteq} \rangle\!\!\downharpoonleft N_r : A^{\sqsubseteq}; \Gamma_o^{\sqsubseteq}}$$

Where $AB_{lr}^{\sqsubseteq} : A_l \sqsubseteq B_r$, which exists by transitivity lemma B.3.

□

Lemma E.2 (Hide Monotonicity). *If* $\Gamma_1^{\sqsubseteq} : \Gamma_{l1} \sqsubseteq \Gamma_{r1}$ *and* $\Gamma_2^{\sqsubseteq} : \Gamma_{l2} \sqsubseteq \Gamma_{r2}$ *and* $\Gamma_{l1} \subseteq \Gamma_{l2}$ *and* $\Gamma_{r1} \subseteq \Gamma_{r2}$ *and* $M_l \sqsubseteq M_r$, *then*

$$\texttt{hide}\ \Gamma_{l1} \subseteq \Gamma_{l2}; M_l \sqsubseteq \texttt{hide}\ \Gamma_{r1} \subseteq \Gamma_{r2}; M_r$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vdash (M_l :: B_l) \sqsubseteq (M_r :: B_r) : B^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}} \qquad \frac{x : A^{\sqsubseteq} \in \Gamma^{\sqsubseteq}}{\Gamma^{\sqsubseteq} \vdash x \sqsubseteq x : A^{\sqsubseteq}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, x : A^{\sqsubseteq} \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash \texttt{let } x = M_l; N_l \sqsubseteq \texttt{let } x = M_r; N_r : B; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq} \qquad X \cong B^{\sqsubseteq} \in \Gamma^{\sqsubseteq}}{\Gamma^{\sqsubseteq} \vdash \texttt{seal}_X M_l \sqsubseteq \texttt{seal}_X M_r : X} \qquad \frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_o \qquad X \cong B^{\sqsubseteq} \in \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_o}{\Gamma^{\sqsubseteq} \vdash \texttt{unseal}_X M_l \sqsubseteq \texttt{unseal}_X M_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_o}$$

$$\Gamma^{\sqsubseteq} \vdash \texttt{true} \sqsubseteq \texttt{true} : \mathbb{B}; \cdot \qquad \Gamma^{\sqsubseteq} \vdash \texttt{false} \sqsubseteq \texttt{false} : \mathbb{B}; \cdot$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash N_{lt} \sqsubseteq N_{rt} : B^{\sqsubseteq}_t; \Gamma^{\sqsubseteq}_t \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash N_{lf} \sqsubseteq N_{rf} : B^{\sqsubseteq}_f; \Gamma^{\sqsubseteq}_f}{\Gamma^{\sqsubseteq} \vdash \texttt{if } M_l \texttt{ then } N_{lt} \texttt{ else } N_{lf} \sqsubseteq \texttt{if } M_r \texttt{ then } N_{rt} \texttt{ else } N_{rf} : B^{\sqsubseteq}_t \sqcup B^{\sqsubseteq}_f; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_t \cap \Gamma^{\sqsubseteq}_f}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_{l1} \sqsubseteq M_{r1} : A^{\sqsubseteq}_1; \Gamma^{\sqsubseteq}_1 \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_1 \vdash M_{l2} \sqsubseteq M_{r2} : A^{\sqsubseteq}_2; \Gamma^{\sqsubseteq}_2}{\Gamma^{\sqsubseteq} \vdash (M_{l1}, M_{l2}) \sqsubseteq (M_{r1}, M_{r2}) : A^{\sqsubseteq}_1 \times A^{\sqsubseteq}_2; \Gamma^{\sqsubseteq}_1, \Gamma^{\sqsubseteq}_2}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M, x : \pi_0(A^{\sqsubseteq}), y : \pi_1(A^{\sqsubseteq}_1) \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash \texttt{let } (x, y) = M_l; N_l \sqsubseteq \texttt{let } (x, y) = M_r; N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

$$\frac{\Gamma^{\sqsubseteq}, x : A^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma^{\sqsubseteq} \vdash \lambda x : A_l.M_l \sqsubseteq \lambda x : A_r.M_r : A^{\sqsubseteq} \to B^{\sqsubseteq}; \cdot}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash M_l\ N_l \sqsubseteq M_r\ N_r : \mathrm{cod}(A^{\sqsubseteq}); \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

$$\frac{\Gamma^{\sqsubseteq}, X \cong B^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_o \qquad \Gamma^{\sqsubseteq} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vdash \mathsf{pack}^{\nu}(X \cong B_l, M_l) \sqsubseteq \mathsf{pack}^{\nu}(X \cong B_r, M_r) : \exists^{\nu} X.A^{\sqsubseteq}; \cdot}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M, X, x : \mathrm{un}\exists^{\nu}(A^{\sqsubseteq}) \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash \mathsf{unpack}\ (X, x) = M_l; N_l \sqsubseteq \mathsf{unpack}\ (X, x) = M_r; N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N|_X}$$

$$\frac{\Gamma^{\sqsubseteq}, X \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_o}{\Gamma^{\sqsubseteq} \vdash \Lambda^{\nu} X.M_l \sqsubseteq \Lambda^{\nu} X.M_r : \forall^{\nu} X.A^{\sqsubseteq}; \cdot}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vdash M_l\{X \cong B_l\} \sqsubseteq M_r\{X \cong B_r\} : \mathrm{un}\forall^{\nu}(A^{\sqsubseteq}); \Gamma^{\sqsubseteq}_M, X \cong B^{\sqsubseteq}}$$

Fig. 29. PolyG$^{\nu}$ Term Precision

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : AC^{\sqsubseteq};\Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma,\Gamma' \vdash AB^{\sqsubseteq} : A \sqsubseteq B \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash BC^{\sqsubseteq} : B \sqsubseteq C}{\Gamma^{\sqsubseteq} \vdash \langle AB^{\sqsubseteq}\rangle\!\uparrow M_l \sqsubseteq M_r : BC^{\sqsubseteq};\Gamma^{\sqsubseteq\prime}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : BC^{\sqsubseteq};\Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma,\Gamma' \vdash AB^{\sqsubseteq} : A \sqsubseteq B \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash BC^{\sqsubseteq} : B \sqsubseteq C}{\Gamma^{\sqsubseteq} \vdash \langle AB^{\sqsubseteq}\rangle\!\downarrow M_l \sqsubseteq M_r : AC^{\sqsubseteq};\Gamma^{\sqsubseteq\prime}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : AB^{\sqsubseteq};\Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma,\Gamma' \vdash BC^{\sqsubseteq} : B \sqsubseteq C \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash AB^{\sqsubseteq} : A \sqsubseteq B}{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq \langle BC^{\sqsubseteq}\rangle\!\uparrow M_r : AC^{\sqsubseteq};\Gamma^{\sqsubseteq\prime}}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : AC^{\sqsubseteq};\Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma,\Gamma' \vdash BC^{\sqsubseteq} : B \sqsubseteq C \qquad \Gamma^{\sqsubseteq},\Gamma^{\sqsubseteq\prime} \vdash AB^{\sqsubseteq} : A \sqsubseteq B}{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq \langle BC^{\sqsubseteq}\rangle\!\downarrow M_r : AB^{\sqsubseteq};\Gamma^{\sqsubseteq\prime}}$$

Fig. 30. PolyC$^\nu$ Term Precision Part 1

Proof. By induction over $\Gamma_2^{\sqsubseteq}$, each case is either congruence or trivial application of inductive hypothesis. □

Lemma E.3. *If* $\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq};\Gamma^{\sqsubseteq\prime}$ *in the surface language, then* $\Gamma^{\sqsubseteq} \vdash M_l^+ \sqsubseteq M_r^+ : A^{\sqsubseteq};\Gamma^{\sqsubseteq\prime}$

Proof. By induction on term precision derivations.

(1) Cast. By applying lemma E.1 twice.
(2) Var: Immediate
(3) Let: immediate
(4) seal: by the argument for the ascription case.
(5) unseal: There are three cases for $A^{\sqsubseteq}$: X, ? and $\text{tag}_X(X)$. The first case is immediate. If $A^{\sqsubseteq} = ?$, we need to show

$$\texttt{unseal}_X\langle \text{tag}_X(X)\rangle\!\downarrow M_l^+ \sqsubseteq \texttt{unseal}_X\langle \text{tag}_X(X)\rangle\!\downarrow M_r^+$$

Which follows by $\texttt{unseal}_X$ congruence and lemma E.1. For the final case we need to show

$$\texttt{unseal}_X M_l^+ \sqsubseteq \texttt{unseal}_X\langle \text{tag}_X(X)\rangle\!\downarrow M_r^+$$

which follows by congruence for $\texttt{unseal}_X$ and the downcast-right rule.
(6) tag-check $\texttt{is}(G)?\ M_l \sqsubseteq \texttt{is}(G)?\ M_r$: follows by congruence and lemma E.1.
(7) tru: Immediate
(8) fls: Immediate
(9) if: By if congruence, we need to show the condition and the two branches of the if are ordered.
   - For the condition there are three subcases $M_l \sqsubseteq M_r : A^{\sqsubseteq}$: either $\mathbb{B}$, ? or $\text{tag}_{\mathbb{B}}(\mathbb{B})$. The ordering follows by the same argument as the $\texttt{unseal}_X$ case.
   - The two branches follow by the same argument. We describe the true branch. We have by inductive hypothesis that $N_{tl}^+ \sqsubseteq N_{tr}^+$ and we need to show

$$\langle B_{tl}^{\sqsubseteq}\rangle\!\downarrow \texttt{hide}\ \Gamma_{tl} \subseteq \Gamma_{tl} \cap \Gamma_{fl}; N_{tl}^+ \sqsubseteq \langle B_{tr}^{\sqsubseteq}\rangle\!\downarrow \texttt{hide}\ \Gamma_{tr} \subseteq \Gamma_{tr} \cap \Gamma_{fr}; N_{tr}^+$$

   Which follows by lemmas E.1 and E.2.
(10) Pair intro: Immediate by inductive hypothesis.
(11) Pair elim: By similar argument to $\texttt{unseal}_X$
(12) By congruence and lemma E.2.
(13) Function application: similar argument to $\texttt{unseal}_X$
(14) $\exists^\nu$ introduction: by congruence and lemma E.2.
(15) $\forall^\nu$ elimination: by similar argument to $\texttt{unseal}_X$, and lemma E.2.
(16) $\forall^\nu$ introduciton: by congruence and lemma E.2.
(17) $\forall^\nu$ elimination: by similar argument to $\texttt{unseal}_X$ case.

□

$$\frac{x : A^{\sqsubseteq} \in \Gamma^{\sqsubseteq}}{\Gamma^{\sqsubseteq} \vdash x \sqsubseteq x : A^{\sqsubseteq}} \qquad \frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma_M \qquad \Gamma^{\sqsubseteq}, \Gamma_M, x : A^{\sqsubseteq} \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma_N}{\Gamma^{\sqsubseteq} \vdash \mathtt{let}\ x = M_l; N_l \sqsubseteq \mathtt{let}\ x = M_r; N_r : B^{\sqsubseteq}; \Gamma_M, \Gamma_N}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}, X \cong A^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma^{\sqsubseteq} \vdash \mathsf{hide}\ X \cong A_l; M_l \sqsubseteq \mathsf{hide}\ X \cong A_r; M_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}, \Gamma^{\sqsubseteq\prime\prime}}$$

$$\frac{(X \cong A^{\sqsubseteq}) \in \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}{\Gamma^{\sqsubseteq} \vdash \mathsf{seal}_X M_l \sqsubseteq \mathsf{seal}_X M_r : X; \Gamma^{\sqsubseteq\prime}}$$

$$\frac{(X \cong A^{\sqsubseteq}) \in \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : X; \Gamma^{\sqsubseteq\prime}}{\Gamma^{\sqsubseteq} \vdash \mathsf{unseal}_X M_l \sqsubseteq \mathsf{unseal}_X M_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}} \qquad \Gamma^{\sqsubseteq} \vdash \mathtt{true} \sqsubseteq \mathtt{true} : \mathbb{B}$$

$$\Gamma^{\sqsubseteq} \vdash \mathtt{false} \sqsubseteq \mathtt{false} : \mathbb{B}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : \mathbb{B}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash N_{lt} \sqsubseteq N_{rt} : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash N_{lf} \sqsubseteq N_{rf} : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash \mathtt{if}\ M_l\ \mathsf{then}\ N_{lt}\ \mathsf{else}\ N_{lf} \sqsubseteq \mathtt{if}\ M_r\ \mathsf{then}\ N_{rt}\ \mathsf{else}\ N_{rf} : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_{l1} \sqsubseteq M_{r1} : A^{\sqsubseteq}_1; \Gamma^{\sqsubseteq}_1 \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_1 \vdash M_{l2} \sqsubseteq M_{r2} : A^{\sqsubseteq}_2; \Gamma^{\sqsubseteq}_2}{\Gamma^{\sqsubseteq} \vdash (M_{l1}, M_{l2}) \sqsubseteq (M_{r1}, M_{r2}) : A^{\sqsubseteq}_1 \times A^{\sqsubseteq}_2; \Gamma^{\sqsubseteq}_1, \Gamma^{\sqsubseteq}_2}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}_1 \times A^{\sqsubseteq}_2; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M, x : A^{\sqsubseteq}_1, y : A^{\sqsubseteq}_2 \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash \mathtt{let}\ (x, y) = M_l; N_l \sqsubseteq \mathtt{let}\ (x, y) = M_r; N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

$$\frac{\Gamma^{\sqsubseteq}, x : A^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : B^{\sqsubseteq}; \cdot \qquad \Gamma^{\sqsubseteq} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma^{\sqsubseteq} \vdash \lambda x : A_l.M_l \sqsubseteq \lambda x : A_l.M_l : A^{\sqsubseteq} \to B^{\sqsubseteq}; \cdot}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq} \to B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash N_l \sqsubseteq N_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash M_l\ N_l \sqsubseteq M_r\ N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

$$\frac{\Gamma^{\sqsubseteq}, X \cong B^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \cdot}{\Gamma^{\sqsubseteq} \vdash \mathsf{pack}^{\nu}(X \cong B_l, M_l) \sqsubseteq \mathsf{pack}^{\nu}(X \cong B_r, M_r) : \exists^{\nu} X.A^{\sqsubseteq}; \cdot}$$

$$\frac{\begin{array}{c}\Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash \Gamma^{\sqsubseteq}_N \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N \vdash B^{\sqsubseteq} \\ \Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : \exists^{\nu} X.A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M, X, x : A^{\sqsubseteq} \vdash N_l \sqsubseteq N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N\end{array}}{\Gamma^{\sqsubseteq} \vdash \mathsf{unpack}\ (X, x) = M_l; N_l \sqsubseteq \mathsf{unpack}\ (X, x) = M_r; N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : \forall^{\nu} X.A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vdash M_l\{X \cong B_l\} \sqsubseteq M_r\{X \cong B_r\} : A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}, X \cong B^{\sqsubseteq}} \qquad \frac{\Gamma^{\sqsubseteq}, X \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq}; \cdot}{\Gamma^{\sqsubseteq} \vdash \Lambda^{\nu} X.M_l \sqsubseteq \Lambda^{\nu} X.M_r : \forall^{\nu} X.A^{\sqsubseteq}; \cdot}$$

Fig. 31. PolyC$^{\nu}$ Term Precision Part 2

F SIMULATION

We provide the full proof of simulation here, including all supporting lemmas. As outlined in the paper, this proof revolves around a translation relation that generalizes our translation function.

Lemma F.1 (target language dynamic semantics are deterministic). *If $\Sigma' \triangleright M' \mapsto \Sigma'_1 \triangleright M'_1$ and $\Sigma' \triangleright M' \mapsto \Sigma'_2 \triangleright M'_2$ then $\Sigma'_1 = \Sigma'_2$ and $M'_1 = M'_2$.*

Lemma F.2. *If $\Sigma; \Gamma \vdash M : A; \Gamma'$, then $M \rightsquigarrow^{CT} [\![\Sigma; \Gamma \vdash M]\!]$.*

Proof. By induction on M. □

The proof of simulation requires knowledge about how substitutions translate. Value substitutions are straightforward due to the definition of our relation. Type substitutions behave differently depending on whether the type variable to be replaced is known.

Lemma F.3. *If $\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : B; \Gamma'$ and $X \in \Gamma$ and $\sigma : A \in \Sigma$, then $M[\sigma/X] \rightsquigarrow^{CT} M'[[\![\Sigma; \Gamma \vdash A]\!]/X][\sigma/c_X]$.*

Proof. By induction on the derivation of $M \rightsquigarrow^{CT} M'$. □

Lemma F.4. *If $\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : B; \Gamma'$ and $X \cong A \in \Gamma, \Gamma'$ and $\sigma : A \in \Sigma$, then $M[\sigma/X] \rightsquigarrow^{CT} M'[\sigma/c_X]$.*

Proof. By induction on the derivation of $M \rightsquigarrow^{CT} M'$. □

Lemma F.5. *If $\Sigma; \Gamma, x : A', \Gamma' \vdash M \rightsquigarrow^{CT} M' : A; \Gamma''$ and $\Sigma; \Gamma \vdash V \rightsquigarrow^{CT} \texttt{ret}\ V' : A'; \cdot$, then $\Sigma; \Gamma, \Gamma' \vdash M[V/x] \rightsquigarrow^{CT} M'[V'/x] : A; \Gamma''$.*

Proof. By induction on the derivation of $M \rightsquigarrow^{CT} M'$. □

We use a restricted form of reduction that only eliminates bind steps in our proof of simulation. This reduction gives us exactly the leeway we need to deal with the possibility of administrative redexes in our translation.

Definition F.6 (bind reduction). We define $\mapsto_b$ to be the least relation on closed terms such that $S[x \leftarrow \texttt{ret}\ V'; M'] \mapsto_b S[M'[V'/x]]$. Note that $\mapsto_b$ is a strict subset of $\mapsto$. We use $\mapsto_b^*$ to mean its reflexive, transitive closure.

Lemma F.7 (bind reduction normalization). *Let $\Sigma; \cdot \vdash M_1 : A$. There exists a unique M_2 such that $M_1 \mapsto_b^* M_2$ and M_2 does not take a bind reduction.*

Proof. By induction on the number of binds in M_1 not under thunks. If $M_1 = S[x \leftarrow \texttt{ret}\ V; N_1]$ then $M_1 \mapsto_b N_1[V/x]$ and we conclude by the inductive hypothesis for $N_1[V/x]$. Otherwise, M_1 must not take a bind reduction, so $M_2 = M_1$ and we conclude by reflexivity. □

Lemma F.8 (bind reduction confluence). *Let $\Sigma; \cdot \vdash M_1 : A$, $\Sigma; \cdot \vdash M_2 : A$, and $\Sigma; \cdot \vdash M_3 : A$. If $M_1 \mapsto_b^* M_2$, $M_1 \mapsto_b^* M_3$, and M_3 does not take a bind reduction, then $M_2 \mapsto_b^* M_3$.*

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} x \leftarrow \mathtt{ret}\ V'; M' : A; \Gamma_1}{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M'[V'/x] : A; \Gamma_1}$$

$$\frac{\Sigma; \Gamma \vdash M_1 \rightsquigarrow^{CT} M_1' : A_1; \Gamma_1 \qquad \Sigma; \Gamma, \Gamma_1 \vdash M_2 \rightsquigarrow^{CT} M_2' : A_2; \Gamma_2}{\Sigma; \Gamma \vdash (M_1, M_2) \rightsquigarrow^{CT} x_1 \leftarrow M_1'; x_2 \leftarrow M_2'; \mathtt{ret}\ (x_1, x_2) : A_1 \times A_2; \Gamma_1, \Gamma_2}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : \mathbb{B}; \Gamma' \qquad \Sigma; \Gamma, \Gamma' \vdash M_1 \rightsquigarrow^{CT} M_1' : A; \Gamma'' \qquad \Sigma; \Gamma, \Gamma' \vdash M_2 \rightsquigarrow^{CT} M_2' : A; \Gamma''}{\Sigma; \Gamma \vdash \mathtt{if}\ M\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 \rightsquigarrow^{CT} r \leftarrow M'; \mathtt{if}\ r\ \mathtt{then}\ M_1'\ \mathtt{else}\ M_2' : A; \Gamma', \Gamma''}$$

$$\frac{}{\Sigma; \Gamma \vdash \mathtt{true} \rightsquigarrow^{CT} \mathtt{ret\ true} : \mathbb{B}; \cdot} \qquad \frac{}{\Sigma; \Gamma \vdash \mathtt{false} \rightsquigarrow^{CT} \mathtt{ret\ false} : \mathbb{B}; \cdot} \qquad \frac{}{\Sigma; \Gamma \vdash \mho \rightsquigarrow^{CT} \mho : A; \cdot}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : A; \Gamma' \qquad \Sigma; \Gamma, \Gamma', x : A \vdash N \rightsquigarrow^{CT} N' : B; \Gamma''}{\Sigma; \Gamma \vdash \mathtt{let}\ x = M; N \rightsquigarrow^{CT} x \leftarrow M'; N' : B; \Gamma', \Gamma''}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : A_1 \times A_2; \Gamma' \qquad \Sigma; \Gamma, \Gamma', x : A_1, y : A_2 \vdash N \rightsquigarrow^{CT} N' : B; \Gamma''}{\Sigma; \Gamma \vdash \mathtt{let}\ (x, y) = M; N \rightsquigarrow^{CT} r \leftarrow M'; \mathtt{let}\ (x, y) = r; N' : B; \Gamma', \Gamma''}$$

$$\frac{\Sigma; \Gamma, X \cong A \vdash M \rightsquigarrow^{CT} M' : B; \cdot \qquad A' = [\![\Sigma; \Delta \vdash A]\!]}{\Sigma; \Gamma \vdash \mathsf{pack}^{\nu}(X \cong A, M) \rightsquigarrow^{CT} \mathtt{ret}\ \mathsf{pack}(A', \mathsf{thunk}\ \lambda(c_X : \mathrm{Case}\ A').M')\ \mathsf{as}\ [\![\Sigma; \Gamma \vdash \exists^{\nu} X.B]\!] : \exists^{\nu} X.B; \cdot}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : \exists^{\nu} X.A; \Gamma' \qquad \Sigma; \Gamma \vdash N \rightsquigarrow^{CT} N' : B; \Gamma'' \qquad \Gamma, \Gamma', \Gamma'' \vdash B}{\Sigma; \Gamma \vdash \mathsf{unpack}\ (X, x) = M; N \rightsquigarrow^{CT} \begin{array}{l} r \leftarrow M'; \mathsf{unpack}\ (X, f) = r; \\ \mathsf{newcase}_X\ c_X; x \leftarrow (\mathtt{force}\ f)\ c_X; N' \end{array} : B; \Gamma', \Gamma''}$$

$$\frac{\Sigma; \Gamma, X \vdash M \rightsquigarrow^{CT} M' : A; \cdot}{\Sigma; \Gamma \vdash \Lambda X.M \rightsquigarrow^{CT} \mathtt{ret\ thunk}\ \Lambda X.\lambda(c_X : \mathrm{Case}\ X).M' : \forall^{\nu} X.A; \cdot}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : \forall^{\nu} X.B; \Gamma'}{\Sigma; \Gamma \vdash M\{X \cong A\} \rightsquigarrow^{CT} f \leftarrow M'; (\mathtt{force}\ f)[A]\ c_X : B; \Gamma', X \cong A}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : \forall^{\nu} X.B; \Gamma'}{\Sigma; \Gamma \vdash M\{\sigma \cong A\} \rightsquigarrow^{CT} f \leftarrow M'; (\mathtt{force}\ f)[A]\ \sigma : B[\sigma/X]; \Gamma'}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : A \to B; \Gamma' \qquad \Sigma; \Gamma, \Gamma' \vdash N \rightsquigarrow^{CT} N' : A; \Gamma''}{\Sigma; \Gamma \vdash M\ N \rightsquigarrow^{CT} f \leftarrow M'; a \leftarrow N'; (\mathtt{force}\ f)\ a : B; \Gamma', \Gamma''}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : B; \cdot}{\Sigma; \Gamma \vdash \lambda x : A.M \rightsquigarrow^{CT} \mathtt{ret\ thunk}\ \lambda x : [\![\Sigma; \Gamma \vdash A]\!].M' : A \to B; \cdot}$$

$$\frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : A; \Gamma' \qquad X \cong A \in \Gamma, \Gamma'}{\Sigma; \Gamma \vdash \mathsf{seal}_X M \rightsquigarrow^{CT} M' : X; \Gamma'} \qquad \frac{\Sigma; \Gamma \vdash M \rightsquigarrow^{CT} M' : X; \Gamma' \qquad X \cong A \in \Gamma, \Gamma'}{\Sigma; \Gamma \vdash \mathsf{unseal}_X M \rightsquigarrow^{CT} M' : A; \Gamma'}$$

Fig. 32. PolyC$^{\nu}$ translation relation

$$\frac{\Sigma;\Gamma \vdash M \rightsquigarrow^{CT} M' : G;\Gamma'}{\Sigma;\Gamma \vdash \mathtt{inj}_G\ M \rightsquigarrow^{CT} x \leftarrow M';\mathtt{ret}\ \mathtt{inj}_{\mathrm{case}(G)}\ x : ?;\Gamma'}$$

$$\frac{\Sigma;\Gamma, X \cong A \vdash M \rightsquigarrow^{CT} M' : B;\cdot \qquad A' = [\![\Sigma;\Delta \vdash A]\!]}{\Sigma;\Gamma \vdash \mathtt{pack}^{\nu}(X \cong A, [], M) \rightsquigarrow^{CT} \mathtt{ret}\ \mathtt{pack}(A', \mathtt{thunk}\ \lambda(c_X : \mathtt{Case}\ A').M')\ \mathtt{as}\ [\![\Sigma;\Gamma \vdash \exists^{\nu}X.B]\!] : \exists^{\nu}X.B;\cdot}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{pack}^{\nu}(X \cong A, [A_2 \updownarrow_2 \ldots], M) \rightsquigarrow^{CT} \mathtt{ret}\ \mathtt{pack}(A', \mathtt{thunk}\ \lambda(c_X : \mathtt{Case}\ A').M')\ \mathtt{as}\ \exists X.A'_1 : \exists^{\nu}X.A^{\sqsubseteq}_{1l};\cdot}{\begin{array}{c}\Sigma;\Gamma \vdash \mathtt{pack}^{\nu}(X \cong A, [A_1 \upharpoonleft, A_2 \updownarrow_2 \ldots], M) \\ \rightsquigarrow^{CT} \left(\begin{array}{ll}\mathtt{ret}\ \mathtt{pack}(A', & \mathtt{thunk}\ \lambda c_X : \mathtt{Case}\ A'. \\ & [\![\Sigma;\Gamma, X \cong A' \vdash A^{\sqsubseteq}]\!]\upharpoonleft \\ & \quad [(\mathtt{force}\ \mathtt{thunk}\ \lambda c_X : \mathtt{Case}\ A'.M')\ c_X])\ \mathtt{as}\ [\![\exists^{\nu}X.A^{\sqsubseteq}_{1r}]\!]\end{array}\right) : \exists^{\nu}X.A^{\sqsubseteq}_{1r};\cdot\end{array}}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{pack}^{\nu}(X \cong A, [A_2 \updownarrow_2 \ldots], M) \rightsquigarrow^{CT} \mathtt{ret}\ \mathtt{pack}(A', \mathtt{thunk}\ \lambda(c_X : \mathtt{Case}\ A').M')\ \mathtt{as}\ \exists X.A'_1 : \exists^{\nu}X.A^{\sqsubseteq}_{1r};\cdot}{\begin{array}{c}\Sigma;\Gamma \vdash \mathtt{pack}^{\nu}(X \cong A, [A_1 \downharpoonleft, A_2 \updownarrow_2 \ldots], M) \\ \rightsquigarrow^{CT} \left(\begin{array}{ll}\mathtt{ret}\ \mathtt{pack}(A', & \mathtt{thunk}\ \lambda c_X : \mathtt{Case}\ A'. \\ & [\![\Sigma;\Gamma, X \cong A' \vdash A^{\sqsubseteq}]\!]\downharpoonleft \\ & \quad [(\mathtt{force}\ \mathtt{thunk}\ \lambda c_X : \mathtt{Case}\ A'.M')\ c_X])\ \mathtt{as}\ [\![\exists^{\nu}X.A^{\sqsubseteq}_{1l}]\!]\end{array}\right) : \exists^{\nu}X.A^{\sqsubseteq}_{1l};\cdot\end{array}}$$

$$\frac{\Sigma;\Gamma \vdash M \rightsquigarrow^{CT} M' : A^{\sqsubseteq}_l;\Gamma' \qquad \Sigma;\Gamma \vdash A^{\sqsubseteq}}{\Sigma;\Gamma \vdash \langle A^{\sqsubseteq}\rangle\upharpoonleft M \rightsquigarrow^{CT} [\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\upharpoonleft [M'] : A^{\sqsubseteq}_r;\Gamma'} \qquad \frac{\Sigma;\Gamma \vdash M \rightsquigarrow^{CT} M' : A^{\sqsubseteq}_r;\Gamma' \qquad \Sigma;\Gamma \vdash A^{\sqsubseteq}}{\Sigma;\Gamma \vdash \langle A^{\sqsubseteq}\rangle\downharpoonleft M \rightsquigarrow^{CT} [\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\downharpoonleft [M'] : A^{\sqsubseteq}_l;\Gamma'}$$

$$\frac{\Sigma;\Gamma \vdash M \rightsquigarrow^{CT} M' : ?;\Gamma'}{\Sigma;\Gamma \vdash \mathtt{is}(G)?\ M \rightsquigarrow^{CT} \left(\begin{array}{l} r \leftarrow M'; \\ \mathtt{match}\ \mathrm{case}(G)\ \mathtt{with}\ r\{\mathtt{inj}\ y.\mathtt{ret}\ \mathtt{true} \mid \mathtt{ret}\ \mathtt{false}\}\end{array}\right) : \mathbb{B};\Gamma'}$$

Fig. 33. PolyC$^{\nu}$ translation relation (Continued)

Proof. By induction on $M_1 \mapsto_b^* M_2$. If $M_1 = M_2$, then we conclude since $M_1 \mapsto_b^* M_3$. Otherwise, by the definition of $\mapsto_b^*$, we have $M_1 \mapsto_b N_1 \mapsto_b^* M_2$. Since M_1 takes a bind reduction, $M_1 \neq M_3$ and so we must have $M_1 \mapsto_b N_2 \mapsto_b^* M_3$. Furthermore, by Lemma C.2 (deterministic semantics), we have $N_1 = N_2$. We then conclude by the inductive hypothesis for N_2, M_2, M_3. □

At the core of our simulation argument is the guarantee that, up to some bind reductions, any related terms have a related structure that follows the form of the translation function.

Lemma F.9 (canonical forms of the translation relation). *If $\Sigma;\cdot \vdash M \rightsquigarrow^{CT} M' : B$, then either*

- $M = \Lambda X.N_1$ *and* $M' = \mathtt{ret}\ \mathtt{thunk}\ \Lambda X.\lambda(c_X : Case\ X).N'_1$
- $M = N_1\{\sigma \cong A\}$ *and* $(f \leftarrow N'_1; (\mathtt{force}\ f)[A]\ \sigma)[\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \mathtt{hide}\ X \cong A; N_1$ *and* $M' = \mathtt{newcase}_A\ c_X; N'_1$
- $M = (N_1, N_2)$ *and* $(x_1 \leftarrow N'_1; x_2 \leftarrow N'_2; \mathtt{ret}\ (x_1, x_2))[\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \mathtt{let}\ (x, y) = N_1; N_2$ *and* $(r \leftarrow N'_1; \mathtt{let}\ (x, y) = r; N'_2)[\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \mathtt{true}$ *and* $M' = \mathtt{ret}\ \mathtt{true}$
- $M = \mathtt{false}$ *and* $M' = \mathtt{ret}\ \mathtt{false}$
- $M = \mathtt{if}\ N_1\ \mathtt{then}\ N_2\ \mathtt{else}\ N_3$ *and* $(r \leftarrow N'_1; \mathtt{if}\ r\ \mathtt{then}\ N'_2\ \mathtt{else}\ N'_3)[\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \mathtt{let}\ x = N_1; N_2$ *and* $(x \leftarrow N'_1; N'_2)[\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \mho_A$ *and* $M' = \mho$

- $M = \texttt{inj}_G\ N_1$ *and* $(x \leftarrow N_1'; \texttt{inj}_{case(G)}\ x)[\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \langle A^{\sqsubseteq}\rangle\updownarrow N_1$ *and* $[\![\Sigma; \cdot \vdash A^{\sqsubseteq}]\!]\updownarrow[N_1'][\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \texttt{is}(G)?\ N_1$ *and* $(r \leftarrow M'; \texttt{match}\ case(G)\ \texttt{with}\ r\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\})[\gamma_p] \mapsto_b^* M'[\gamma_p]$
- $M = \texttt{seal}_\sigma N_1$ *and* $N_1' \mapsto_b^* M'$
- $M = \texttt{unseal}_\sigma N_1$ *and* $N_1' \mapsto_b^* M'$
- $M = \texttt{pack}^\nu(X \cong A, N_1)$ *and* $M' = \texttt{ret pack}(A, \texttt{thunk}\ \lambda(c_X : Case\ A).N_1')\ \texttt{as}\ [\![B]\!]$
- $M = \texttt{pack}^\nu(X \cong A', [A^{\sqsubseteq}\updownarrow ...], N_1)$ *and* $M' = \texttt{ret pack}(A, \texttt{thunk}\ \lambda(c_X : Case\ A).M_n')\ \texttt{as}\ [\![B]\!]$ *where*
$$\begin{array}{rcl} M_0' &=& N_1' \\ M_{i+1}' &=& [\![\Sigma;\Delta \vdash A_{i+1}^{\sqsubseteq}]\!]\updownarrow_{i+1}[\texttt{force}\ (\texttt{thunk}\ (\lambda c_X : Case\ A'.[\![M_i]\!]))\ c_X] \end{array}$$
- $M = \texttt{unpack}\ (X, x) = N_1; N_2$ *and* $M_{unp} \mapsto_b^* M'$ *where*
$$\begin{array}{rl} M_{unp} = & r \leftarrow N_1'; \texttt{unpack}\ (X, f) = r; \\ & \texttt{newcase}_X\ c_X; x \leftarrow (\texttt{force}\ f)\ c_X; N_2' \end{array}$$

for some $N_1, ..., N_n$ *and* $N_1', ..., N_n'$ *where* $N_i \rightsquigarrow^{CT} N_i'$.

Proof. By induction on the derivation of $M \rightsquigarrow^{CT} M'$. If the derivation ends in a congruence from the translation, the conclusion is immediate by reflexivity. We then have only the bind-reduction rule to consider. In this case, we have $M' = M''[V'/x]$ where $M \rightsquigarrow^{CT} x \leftarrow \texttt{ret}\ V'; M''$. By the inductive hypothesis, we have the desired property for M and $x \leftarrow \texttt{ret}\ V'; M''$. Then, since $(x \leftarrow \texttt{ret}\ V'; M'')[\gamma_p] \mapsto_b M''[V'/x][\gamma_p] = M'[\gamma_p]$, we have what we need to show by transitivity. □

Lemma F.10. *Let* $\Sigma; \cdot \vdash V : A$. *If* $V \rightsquigarrow^{CT} M'$, *then for any closing* γ $M'[\gamma] \mapsto_b^* \texttt{ret}\ V'[\gamma]$ *and* $V \rightsquigarrow^{CT} \texttt{ret}\ V'$ *for some* V'.

Proof. By induction on the derivation of $V \rightsquigarrow^{CT} M'$. All cases are immediate since $M' = \texttt{ret}\ V'$ except the following:

- $(V_1, V_2) \rightsquigarrow^{CT} x_1 \leftarrow N_1'; x_2 \leftarrow N_2'; \texttt{ret}\ (x_1, x_2)$ where $V_1 \rightsquigarrow^{CT} N_1'$ and $V_2 \rightsquigarrow^{CT} N_2'$: By the inductive hypothesis for each of these sub-derivations, we have $N_i'[\gamma] \mapsto_b^* \texttt{ret}\ V_i'[\gamma]$ and $V_i \rightsquigarrow^{CT} \texttt{ret}\ V_i'$. Then, by definition, we have $M'[\gamma] \mapsto_b^* \texttt{ret}\ (V_1, V_2)[\gamma]$ and $(V_1, V_2) \rightsquigarrow^{CT} \texttt{ret}\ (V_1, V_2)$ as we were required to show.

□

Since our dynamic semantics for PolyC$^\nu$ uses evaluation contexts, we need to be able to describe their transformation into $\text{CBPV}_{\text{OSum}}$.

Lemma F.11 (Translation context decomposition). *If* $\Sigma; \cdot \vdash M_1 : A$ *and* $\Sigma; \cdot \vdash E[M_1] \rightsquigarrow^{CT} M' : B$, *then for any closing* γ, $S[M_1'][\gamma] \mapsto_b^* M'[\gamma]$ *for some* S *and* M_1' *such that* $\Sigma; \cdot \vdash M_1 \rightsquigarrow^{CT} M_1' : A$ *and* $E \rightsquigarrow^{CT} S$.

Proof. By induction on the derivation of $E[M_1] \rightsquigarrow^{CT} M'$. All cases are straightforward except the bind rule. That case proceeds as follows: we have $E[M_1] \rightsquigarrow^{CT} x \leftarrow \texttt{ret}\ V_1'; M''$ and, from the inductive hypothesis, some S, M_1' such that $S[M_1'][\gamma] \mapsto_b^* (x \leftarrow \texttt{ret}\ V_1'; M'')[\gamma]$, $\Sigma; \cdot \vdash M_1 \rightsquigarrow^{CT} M_1' : A$, and $E \rightsquigarrow^{CT} S$. It then suffices to show that $S[M_1'][\gamma] \mapsto_b^* M''[V_1'/x][\gamma]$, which we have by definition. □

$$\frac{}{\Sigma;\Gamma \mid \bullet : A \vdash [] \rightsquigarrow^{CT} \bullet : A; \cdot}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A \vdash E_1 \rightsquigarrow^{CT} S_1' : A_1;\Gamma_1 \qquad \Sigma;\Gamma,\Gamma_1 \vdash M_2 \rightsquigarrow^{CT} M_2' : A_2;\Gamma_2}{\Sigma;\Gamma \mid \bullet : A \vdash (E_1, M_2) \rightsquigarrow^{CT} x_1 \leftarrow S_1'; x_2 \leftarrow M_2'; \texttt{ret}\ (x_1, x_2) : A_1 \times A_2;\Gamma_1,\Gamma_2}$$

$$\frac{\Sigma;\Gamma \vdash V_1 \rightsquigarrow^{CT} \texttt{ret}\ V_1' : A_1;\Gamma_1 \qquad \Sigma;\Gamma,\Gamma_1 \mid \bullet : A \vdash E_2 \rightsquigarrow^{CT} S_2' : A_2;\Gamma_2}{\Sigma;\Gamma \mid \bullet : A \vdash (V_1, E_2) \rightsquigarrow^{CT} x_2 \leftarrow S_2'; \texttt{ret}\ (V_1', x_2) : A_1 \times A_2;\Gamma_1,\Gamma_2}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : \mathbb{B};\Gamma' \qquad \Sigma;\Gamma,\Gamma' \vdash M_1 \rightsquigarrow^{CT} M_1' : A;\Gamma'' \qquad \Sigma;\Gamma,\Gamma' \vdash M_2 \rightsquigarrow^{CT} M_2' : A;\Gamma''}{\Sigma;\Gamma \mid \bullet : A' \vdash \texttt{if}\ E\ \texttt{then}\ M_1\ \texttt{else}\ M_2 \rightsquigarrow^{CT} r \leftarrow S'; \texttt{if}\ r\ \texttt{then}\ M_1'\ \texttt{else}\ M_2' : A;\Gamma',\Gamma''}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : A;\Gamma' \qquad \Sigma;\Gamma,\Gamma', x : A \vdash N \rightsquigarrow^{CT} N' : B;\Gamma''}{\Sigma;\Gamma \mid \bullet : A' \vdash \texttt{let}\ x = E; N \rightsquigarrow^{CT} x \leftarrow S'; N' : B;\Gamma',\Gamma''}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A \vdash E \rightsquigarrow^{CT} S' : A_1 \times A_2;\Gamma' \qquad \Sigma;\Gamma,\Gamma', x : A_1, y : A_2 \vdash N \rightsquigarrow^{CT} N' : B;\Gamma''}{\Sigma;\Gamma \mid \bullet : A \vdash \texttt{let}\ (x,y)\ =\ E; N \rightsquigarrow^{CT} r \leftarrow S'; \texttt{let}\ (x,y)\ =\ r; N' : B;\Gamma',\Gamma''}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A \vdash E \rightsquigarrow^{CT} S' : \exists^{\nu} X.A;\Gamma' \qquad \Sigma;\Gamma \vdash N \rightsquigarrow^{CT} N' : B;\Gamma'' \qquad \Gamma,\Gamma',\Gamma'' \vdash B}{\Sigma;\Gamma \mid \bullet : A \vdash \mathsf{unpack}\ (X,x) = E; N \rightsquigarrow^{CT} \left(\begin{array}{l} r \leftarrow S'; \\ \mathsf{unpack}\ (X,f) = r; \\ \mathsf{newcase}_X\ c_X; \\ x \leftarrow (\mathsf{force}\ f)\ c_X; \\ N' \end{array}\right) : B;\Gamma',\Gamma''}$$

Fig. 34. PolyC$^{\nu}$ evaluation context translation relation

Lemma F.12 (Translation context plug). *If* $\Sigma;\cdot \vdash M_1 \rightsquigarrow^{CT} M_1' : A$ *and* $\Sigma;\cdot \mid \bullet : A \vdash E \rightsquigarrow^{CT} S : B$*, then* $\Sigma;\cdot \vdash E[M_1] \rightsquigarrow^{CT} S[M_1'] : B$.

Proof. By induction on the derivation of $E \rightsquigarrow^{CT} S$. □

Lemma F.13 (Translation stack decomposition). *If* $\Sigma;\cdot \vdash M \rightsquigarrow^{CT} S[M_1'] : A$ *then* $M = E[M_1]$ *for some* E *and* M_1 *such that* $E \rightsquigarrow^{CT} S$ *and* $M_1 \rightsquigarrow^{CT} M_1'$.

Proof. By induction on the derivation of $M \rightsquigarrow^{CT} S[M_1']$. □

Lemma F.14 (Bind reduction preserves translation). *If* $\Sigma;\cdot \vdash M \rightsquigarrow^{CT} M_1' : A$ *and* $M_1'[\gamma_p] \mapsto_b M_2'[\gamma_p]$ *then* $\Sigma;\cdot \vdash M \rightsquigarrow^{CT} M_2' : A$

Proof. By the definition of $\mapsto_b$, we have that $M_1' = S[x \leftarrow \texttt{ret}\ V'; M_3']$ and $M_2' = S[M_3'[V'/x]]$ for some S, V', M_3'. By Lemma F.13 (stack decomposition), we have $M = E[M_3]$, $E \rightsquigarrow^{CT} S$ and $M_3 \rightsquigarrow^{CT} x \leftarrow \texttt{ret}\ V'; M_3'$. We then apply the bind reduction rule, so we have $M_3 \rightsquigarrow^{CT} M_3'[V'/x]$ and conclude by Lemma F.12 (translation context plug). □

We would like all PolyC$^{\nu}$ terms to make progress whenever their corresponding CBPV$_{\text{OSum}}$ translation evaluates. However, this is not always the case since some PolyC$^{\nu}$ terms step to terms

$$\frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : \forall^\nu X.B;\Gamma'}{\Sigma;\Gamma \mid \bullet : A' \vdash E\{X \cong A\} \rightsquigarrow^{CT} f \leftarrow S'; (\texttt{force}\ f)[A]\ c_X : B;\Gamma', X \cong A}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : \forall^\nu X.B;\Gamma'}{\Sigma;\Gamma \mid \bullet : A' \vdash E\{\sigma \cong A\} \rightsquigarrow^{CT} f \leftarrow S'; (\texttt{force}\ f)[A]\ \sigma : B[\sigma/X];\Gamma'}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : A \to B;\Gamma' \qquad \Sigma;\Gamma,\Gamma' \vdash N \rightsquigarrow^{CT} N' : A;\Gamma''}{\Sigma;\Gamma \mid \bullet : A' \vdash E\ N \rightsquigarrow^{CT} f \leftarrow S'; a \leftarrow N'; (\texttt{force}\ f)\ a : B;\Gamma',\Gamma''}$$

$$\frac{\Sigma;\Gamma \vdash V \rightsquigarrow^{CT} \texttt{ret}\ V' : A \to B;\Gamma' \qquad \Sigma;\Gamma,\Gamma' \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : A;\Gamma''}{\Sigma;\Gamma \mid \bullet : A' \vdash V\ E \rightsquigarrow^{CT} a \leftarrow S'; (\texttt{force}\ V')\ a : B;\Gamma',\Gamma''}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : A;\Gamma' \qquad X \cong A \in \Gamma,\Gamma'}{\Sigma;\Gamma \mid \bullet : A' \vdash \texttt{seal}_X E \rightsquigarrow^{CT} S' : X;\Gamma'} \qquad \frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : X;\Gamma' \qquad X \cong A \in \Gamma,\Gamma'}{\Sigma;\Gamma \mid \bullet : A' \vdash \texttt{unseal}_X E \rightsquigarrow^{CT} S' : A;\Gamma'}$$

$$\frac{\Sigma;\Gamma \mid \bullet : A' \vdash E \rightsquigarrow^{CT} S' : G;\Gamma'}{\Sigma;\Gamma \mid \bullet : A' \vdash \texttt{inj}_G\ E \rightsquigarrow^{CT} x \leftarrow S'; \texttt{inj}_{\mathrm{case}(G)}\ x : ?;\Gamma'}$$

$$\frac{\Sigma;\Gamma \mid \bullet : B \vdash E \rightsquigarrow^{CT} S' : A^{\sqsubseteq}_l;\Gamma' \qquad \Sigma;\Gamma \vdash A^{\sqsubseteq}}{\Sigma;\Gamma \mid \bullet : B \vdash \langle A^{\sqsubseteq}\rangle\!\upharpoonleft E \rightsquigarrow^{CT} [\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\!\upharpoonleft [S'] : A^{\sqsubseteq}_r;\Gamma'}$$

$$\frac{\Sigma;\Gamma \mid \bullet : B \vdash E \rightsquigarrow^{CT} S' : A^{\sqsubseteq}_r;\Gamma' \qquad \Sigma;\Gamma \vdash A^{\sqsubseteq}}{\Sigma;\Gamma \mid \bullet : B \vdash \langle A^{\sqsubseteq}\rangle\!\downharpoonleft E \rightsquigarrow^{CT} [\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\!\downharpoonleft [S'] : A^{\sqsubseteq}_l;\Gamma'}$$

$$\frac{\Sigma;\Gamma \mid \bullet : B \vdash E \rightsquigarrow^{CT} S' : ?;\Gamma'}{\Sigma;\Gamma \mid \bullet : B \vdash \texttt{is}(G)?\ E \rightsquigarrow^{CT} \left(\begin{array}{l} r \leftarrow S'; \\ \texttt{match case}(G)\ \texttt{with}\ r\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\} \end{array}\right) : \mathbb{B};\Gamma'}$$

Fig. 35. PolyC$^\nu$ calculus evaluation context translation relation (Continued)

with identical translations. However, the number of such steps is limited by the syntactic size of the term, defined below.

Definition F.15 (Type/Term size).

$$\begin{array}{lcl}
|\texttt{let}\ x = M; N| & = & 1 + |N[M/x]| \\
|\mathrm{tag}_G(A^{\sqsubseteq})| & = & 2 + |A^{\sqsubseteq}| \\
|\texttt{pack}^\nu(X \cong A', M)| & = & 2 + |M| \\
|C(\vec{A^{\sqsubseteq}}, \vec{M})| & = & 1 + \sum_{M \in \vec{M}} |M| + \sum_{A^{\sqsubseteq} \in \vec{A^{\sqsubseteq}}} |A^{\sqsubseteq}| \text{ otherwise}
\end{array}$$

where C denotes any syntactic term or precision judgment constructor.

Theorem F.16 (Simulation). *If $\Sigma; \cdot \vdash M_1 : A; \cdot$ and $\Sigma; \cdot \vdash M_1 \rightsquigarrow^{CT} M_1' : A; \cdot$ and $\Sigma \triangleright M_1 \mapsto \Sigma' \triangleright M_2$, then $\Sigma'; \cdot \vdash M_2 \rightsquigarrow^{CT} M_2' : A; \cdot$ and either $\Sigma_p, [\![\Sigma]\!] \triangleright M_1'[\gamma_p] \mapsto^{+} \Sigma_p, [\![\Sigma']\!] \triangleright M_2'[\gamma_p]$ or $\Sigma_p, [\![\Sigma]\!] \triangleright M_1'[\gamma_p] \mapsto^{*} \Sigma_p, [\![\Sigma']\!] \triangleright M_2'[\gamma_p]$ and $|M_1| > |M_2|$ for some M_2'.*

Proof. Note that, by unique decomposition, we have some context E and redex M_3 such that $M_1 = E[M_3]$ and either $M_3 = \mho_B$ or $M_3 \mapsto M_4$ for some M_4. By Lemma F.11 (translation context decomposition), we have some S, M_3' such that $S[M_3'][\gamma_p] \mapsto_b^* M_1'[\gamma_p]$ where $E \rightsquigarrow^{CT} S$ and $M_3 \rightsquigarrow^{CT} M_3'$. We proceed by cases on $\Sigma \triangleright M_1 \mapsto \Sigma' \triangleright M_2$. In each case, we first use Lemma F.9 (canonical forms of the translation) to determine the form of M_3'.

- Error:

$$E[\mho_B] \mapsto \mho_{type(\Sigma;\cdot\vdash E[\mho_B])} \text{ where } E \neq []$$

We have $M_3' = \mho$ and so $S[M_3'][\gamma_p] \mapsto \mho$. Since $\Sigma; \cdot \vdash \mho_{type(\Sigma;\cdot\vdash M_1)} \rightsquigarrow^{CT} \mho : type(\Sigma; \cdot \vdash M_1)$, we have what we were required to show.

- Instantiation:

$$E[(\Lambda X.N_1)\{\sigma \cong B\}] \mapsto E[N_1[\sigma/X]]$$

Let $B' = [\![\Sigma; \cdot \vdash B]\!]$. We have

$$(f \leftarrow \texttt{ret thunk } \Lambda X.\lambda(c_X : \mathsf{Case}\ X).N_1'; (\texttt{force } f)[B']\ \sigma)[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$$

for some N_1' such that $N_1 \rightsquigarrow^{CT} N_1'$. Then, by Lemma F.8 (bind confluence), we have $M_3'[\gamma_p] \mapsto_b^* ((\texttt{force thunk } \Lambda X.\lambda(c_X : \mathsf{Case}\ X).N_1')[B']\ \sigma)[\gamma_p]$ and so by the dynamic semantics, we have the following:

$$\begin{array}{ll} & M_1'[\gamma_p] \\ \mapsto_b^* & S[(\texttt{force thunk } \Lambda X.\lambda(c_X : \mathsf{Case}\ X).N_1')[B']\ \sigma][\gamma_p] \\ \mapsto & S[(\Lambda X.\lambda(c_X : \mathsf{Case}\ X).N_1')[B']\ \sigma][\gamma_p] \\ \mapsto & S[(\lambda(c_X : \mathsf{Case}\ B').N_1'[B'/X])\ \sigma][\gamma_p] \\ \mapsto & S[N_1'[B'/X][\sigma/c_X]][\gamma_p] \end{array}$$

Furthermore, by Lemma F.3, since $\sigma : B \in \Sigma$ and $\Sigma; X \vdash N_1 \rightsquigarrow^{CT} N_1' : A_1$, we have that $N_1[\sigma/X] \rightsquigarrow^{CT} N_1'[B'/X][\sigma/c_X]$. Then, by Lemma F.12 (translation context plug), we have what we were required to show.

- Hiding:

$$\Sigma \triangleright E[\texttt{hide } X \cong B; N] \mapsto \Sigma, \sigma : \mathsf{Case}\ B \triangleright E[N[\sigma/X]]$$

We have $M_3' = \texttt{newcase}_B\ c_X; N'$ for some N' such that $N \rightsquigarrow^{CT} N'$. We then have the following by the operational semantics:

$$\begin{array}{rrl} & [\![\Sigma]\!] \triangleright & S[\texttt{newcase}_B\ c_X; N'][\gamma_p] \\ \mapsto & [\![\Sigma]\!], \sigma : \mathsf{Case}\ B \triangleright & S[N'[\sigma/c_X]][\gamma_p] \end{array}$$

Then by Lemma F.4, since $\Sigma; \cdot \vdash N \rightsquigarrow^{CT} N' : A, X \cong B$, we have $N[\sigma/X] \rightsquigarrow^{CT} N'[\sigma/c_X]$, so we conclude by Lemma F.12 (translation context plug).

- Pack:

$$E[\mathsf{pack}^\nu(X \cong A_1, N_1)] \mapsto E[\mathsf{pack}^\nu(X \cong A_1, [], N_1)]$$

Note that $\mathsf{pack}^\nu(X \cong A_1, N_1) \rightsquigarrow^{CT} M_3'$ iff $\mathsf{pack}^\nu(X \cong A_1, [], N_1) \rightsquigarrow^{CT} M_3'$, so by Lemma F.12 (translation context plug), we may choose $M_2' = M_1'$. Then, it suffices to show that $|M_1| > |M_2|$. This holds since $|\mathsf{pack}^\nu(X \cong A_1, N_1)| = 2 + |N_1| > 1 + |N_1| = |\mathsf{pack}^\nu(X \cong A_1, [], N_1)|$.

- Unpack:

$$\begin{array}{rrl} & \Sigma \triangleright & E[\texttt{unpack } (X, x) = \mathsf{pack}^\nu(X \cong A_1, [A^{\sqsubseteq} \updownarrow ...], N_1); N_2] \\ \mapsto & \Sigma, \sigma : \mathsf{Case}\ A_1 \triangleright & E[\texttt{let } x = \langle A^{\sqsubseteq}[\sigma/X]\rangle\updownarrow ...\ N_1[\sigma/X]; N_2[\sigma/X]] \end{array}$$

Let $A' = [\![\Sigma; \cdot \vdash \exists^\nu X.B]\!]$ and $A'_1 = [\![\Sigma; \cdot \vdash A_1]\!]$. We have

$$\begin{array}{ll} & r \leftarrow \texttt{ret pack}(A'_1, \texttt{thunk}\ \lambda(c_X : \text{Case } A'_1).N'_{cst}[\gamma_p])\ \texttt{as}\ A'; \\ & \texttt{unpack}\ (X, f) = r; \\ & \texttt{newcase}_X\ c_X; \\ & x \leftarrow (\texttt{force}\ f)\ c_X; \\ & N'_2[\gamma_p] \\ \mapsto_b^* & M'_3[\gamma_p] \end{array}$$

where $N'_{cst} = [\![\Sigma; X \cong A_1 \vdash A^{\sqsubseteq}]\!] \updownarrow [\texttt{force}\ (\texttt{thunk}\ (\lambda c_X : \text{Case } A'.(\ldots\ N'_1\ \ldots)))\ c_X]$ and $N_i \leadsto^{CT} N'_i$. We then have the following by Lemma F.8 (bind confluence) and the operational semantics:

$$\begin{array}{lrl} & & M'_3[\gamma_p] \\ \mapsto_b^* & & \texttt{unpack}\ (X, f) = \texttt{pack}(A'_1, \texttt{thunk}\ \lambda(c_X : \text{Case } A'_1).N'_{cst}[\gamma_p])\ \texttt{as}\ A'; \\ & & \texttt{newcase}_X\ c_X; \\ & & x \leftarrow (\texttt{force}\ f)\ c_X; \\ & & N'_2[\gamma_p] \\ \mapsto & [\![\Sigma]\!] \triangleright & \texttt{newcase}_{A'_1}\ c_X; \\ & & x \leftarrow (\texttt{force thunk}\ \lambda(c_X : \text{Case } A'_1).N'_{cst}[\gamma_p])\ c_X; \\ & & N'_2[\gamma_p][A'_1/X][\texttt{thunk}\ \lambda(c_X : \text{Case } A'_1).N'_{cst}[\gamma_p]/f] \\ = & [\![\Sigma]\!] \triangleright & \texttt{newcase}_{A'_1}\ c_X; \\ & & x \leftarrow (\texttt{force thunk}\ \lambda(c_X : \text{Case } A'_1).N'_{cst}[\gamma_p])\ c_X; \\ & & N'_2[\gamma_p][A'_1/X] \\ \mapsto^* & [\![\Sigma]\!], \sigma : A'_1 \triangleright & x \leftarrow N'_{cst}[\gamma_p][\sigma/c_X]; \\ & & N'_2[\gamma_p][A'_1/X][\sigma/c_X] \\ \mapsto^* & [\![\Sigma]\!], \sigma : A'_1 \triangleright & x \leftarrow [\![\Sigma; X \cong A_1 \vdash A^{\sqsubseteq}]\!]\updownarrow[\ldots\ N'_1\ \ldots][\gamma_p][\sigma/c_X]; \\ & & N'_2[\gamma_p][A'_1/X][\sigma/c_X] \end{array}$$

Note that, since $\Sigma; X \cong A'_1 \vdash N_1 \leadsto^{CT} N'_1$, we have $\Sigma; X \cong A'_1 \vdash \langle A^{\sqsubseteq}[\sigma/X]\rangle\updownarrow \ldots N_1 \leadsto^{CT} [\![\Sigma; X \cong A_1 \vdash A^{\sqsubseteq}]\!]\updownarrow[\ldots\ N'_1\ \ldots]$ by the definition of the translation relation for casts. Furthermore, by Lemma F.4, since $\sigma : A_1 \in \Sigma$, we have $\langle A^{\sqsubseteq}[\sigma/X]\rangle\updownarrow \ldots N_1[\sigma/X] \leadsto^{CT} [\![\Sigma; X \cong A_1 \vdash A^{\sqsubseteq}]\!]\updownarrow[\ldots\ N'_1\ \ldots][\sigma/c_X]$ and by Lemma F.3, since $\sigma : A_1 \in \Sigma$ and $\Sigma; X \vdash N_2 \leadsto^{CT} N'_2 : A_2$, we have $N_2[\sigma/X] \leadsto^{CT} N'_2[A'_1/X][\sigma/c_X]$. Thus, we have

$$\texttt{let}\ x = \langle A^{\sqsubseteq}[\sigma/X]\rangle\updownarrow \ldots N_1[\sigma/X]; N_2[\sigma/X] \leadsto^{CT} x \leftarrow [\![\Sigma; X \cong A_1 \vdash A^{\sqsubseteq}]\!]\updownarrow[\ldots\ N'_1\ \ldots][\sigma/c_X]; N'_2[A'_1/X][\sigma/c_X]$$

so we conclude by Lemma F.12 (translation context plug).

- function application

$$E[(\lambda x : A.N_1)\ V] \mapsto E[N_1[V/x]]$$

We have $(f \leftarrow \texttt{ret thunk}\ \lambda x : [\![\Sigma; \cdot \vdash A]\!].N'_1; a \leftarrow N'_2; \texttt{force}\ f\ a)[\gamma_p] \mapsto_b^* M'_3[\gamma_p]$ where $N_i \leadsto^{CT} N'_i$. By Lemma F.10 (value translation), we have some V' such that $N'_2[\gamma_p] \mapsto_b^* \texttt{ret}\ V'[\gamma_p]$ and $V \leadsto^{CT} \texttt{ret}\ V'$. Then, by Lemma F.8 (bind confluence), we have $M'_3[\gamma_p] \mapsto_b^* (\texttt{force}\ (\texttt{thunk}\ \lambda x : [\![\Sigma; \cdot \vdash A]\!].N'_1)\ V')[\gamma_p]$. Then $M'_3[\gamma_p] \mapsto^+ N'_1[V'/x][\gamma_p]$ and by Lemma F.5 (value substitution translation), we have $\Sigma; \cdot \vdash N_1[V/x] \leadsto^{CT} N'_1[V'/x] : B$. Thus, we conclude by Lemma F.12 (translation context plug).

- If true:

$$E[\texttt{if true then}\ N_1\ \texttt{else}\ N_2] \mapsto E[N_1]$$

We have $(r \leftarrow \texttt{ret true}; \texttt{if}\ r\ \texttt{then}\ N_1'\ \texttt{else}\ N_2')[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$. where $N_i \rightsquigarrow^{CT} N_i'$. We then have the following by Lemma F.7 (bind normalization) and the operational semantics:

$$\begin{array}{ll} & M_1'[\gamma_p] \\ \mapsto_b^* & S[\texttt{if true then}\ N_1'\ \texttt{else}\ N_2'][\gamma_p] \\ \mapsto & S[N_1'][\gamma_p] \end{array}$$

We then conclude by Lemma F.12 (translation context plug) since $N_1 \rightsquigarrow^{CT} N_1'$.

- If false:

$$\texttt{if false then}\ N_1\ \texttt{else}\ N_2 \mapsto M_2$$

We have $(r \leftarrow \texttt{ret false}; \texttt{if}\ r\ \texttt{then}\ N_1'\ \texttt{else}\ N_2')[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$. where $N_i \rightsquigarrow^{CT} N_i'$. We then have the following by Lemma F.7 (bind normalization) and the operational semantics:

$$\begin{array}{ll} & M_1'[\gamma_p] \\ \mapsto_b^* & S[\texttt{if false then}\ N_1'\ \texttt{else}\ N_2'][\gamma_p] \\ \mapsto & S[N_2'][\gamma_p] \end{array}$$

We then conclude by Lemma F.12 (translation context plug) since $N_2 \rightsquigarrow^{CT} N_2'$.

- Pair elimination:

$$\texttt{let}\ (x, y) = (V_1, V_2); N \mapsto N[V_1/x][V_2/y]$$

We have $(r \leftarrow M''; \texttt{let}\ (x, y) = r; N')[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$ where $(V_1, V_2) \rightsquigarrow^{CT} M''$ and $N \rightsquigarrow^{CT} N'$. Thus, by Lemma F.10 (value translation), we further have $M''[\gamma_p] \mapsto_b^* \texttt{ret}\ V'[\gamma_p]$ and $(V_1, V_2) \rightsquigarrow^{CT} \texttt{ret}\ V'$ for some V'. Note that by the type of V', we have $V' = (V_1', V_2')$ for some V_1', V_2'.

We then have the following by Lemma F.8 (bind confluence) and the operational semantics:

$$\begin{array}{ll} & M_1'[\gamma_p] \\ \mapsto_b^* & S[\texttt{let}\ (x, y) = (V_1', V_2'); N'][\gamma_p] \\ \mapsto & S[N'[V_1'/x][V_2'/y]][\gamma_p] \end{array}$$

By Lemma F.5, since $V_i \rightsquigarrow^{CT} \texttt{ret}\ V_i'$, we have $N[V_1/x][V_2/y] \rightsquigarrow^{CT} N'[V_1'/x][V_2'/y]$. We then conclude by Lemma F.12 (translation context plug).

- Let:

$$\texttt{let}\ x = V; N_2 \mapsto N_2[V/x]$$

We have $(x \leftarrow N_1'; N_2')[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$ where $V \rightsquigarrow^{CT} N_1'$ and $N_2 \rightsquigarrow^{CT} N_2'$. Then, by Lemma F.10 (value translation reduction), we have $N_1'[\gamma_p] \mapsto_b^* \texttt{ret}\ V'[\gamma_p]$ for some V' such that $V \rightsquigarrow^{CT} \texttt{ret}\ V'$. Thus, we have the following reduction:

$$S[x \leftarrow N_1'; N_2'][\gamma_p] \mapsto_b^* S[N_2'[V'/x]][\gamma_p]$$

By Lemma F.7 (bind normalization) there exists a unique M_4' such that

$$S[x \leftarrow N_1'; N_2'][\gamma_p] \mapsto_b^* M_4'[\gamma_p]$$

and $M_4'[\gamma_p]$ does not take a bind reduction. Then, by Lemma F.8 (bind reduction confluence), we have $M_1'[\gamma_p] \mapsto_b^* M_4'[\gamma_p]$ and $S[N_2'[V'/x]][\gamma_p] \mapsto_b^* M_4'[\gamma_p]$.

Note that, by Lemma F.5 (value substitution translation), we have $N_2[V/x] \rightsquigarrow^{CT} N_2'[V'/x]$ and thus by Lemma F.12 (translation context plug), we have $E[N_2[V/x]] \rightsquigarrow^{CT} S[N_2'[V'/x]]$. Then, by Lemma F.14 (bind reduction preserves translation), we have $E[N_2[V/x]] \rightsquigarrow^{CT} M_4'$. It finally suffices to show that $|M_1| > |M_2|$, which we have since by definition $|M_3| = 1 + |N_2[V/x]|$.

- Unseal:

$$E[\texttt{unseal}_\sigma \texttt{seal}_\sigma V] \mapsto E[V]$$

We have $N'[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$ where $V \rightsquigarrow^{CT} N'$ and so $S[N'][\gamma_p] \mapsto_b^* S[M_3'][\gamma_p]$. By Lemma F.7 (bind normalization) there exists a unique M_4' such that $S[N'][\gamma_p] \mapsto_b^* M_4'[\gamma_p]$ and $M_4'[\gamma_p]$ does not take a bind reduction. Then, by Lemma F.8 (bind confluence), we have $S[M_3'][\gamma_p] \mapsto_b^* M_4'[\gamma_p]$. Since, by Lemmas F.12 (translation context plug) and F.14 (bind reduction preserves translation), we have $E[V] \rightsquigarrow^{CT} M_4'$, it suffices to show that $|M_1| > |M_2|$. This holds since by definition $|M_3| = 1 + 1 + |V|$.

- Atomic cast:

$$E[\langle A^\sqsubseteq \rangle\!\updownarrow V] \mapsto E[V] \text{ where } A \in \{\mathbb{B}, \alpha, ?\}$$

The reasoning for this case is analogous to the prior case.

- Pair cast:

$$E[\langle A_1^\sqsubseteq \times A_2^\sqsubseteq \rangle\!\updownarrow (V_1, V_2)] \mapsto E[(\langle A_1^\sqsubseteq \rangle\!\updownarrow V_1, \langle A_2^\sqsubseteq \rangle\!\updownarrow V_2)]$$

We have $[\![\Sigma; \cdot \vdash A_1^\sqsubseteq \times A_2^\sqsubseteq]\!]\!\updownarrow [x_1 \leftarrow N_1'; x_2 \leftarrow N_2'; \texttt{ret}\ (x_1, x_2)][\gamma_p] \mapsto_b^* M_3'[\gamma_p]$ where $V_i \rightsquigarrow^{CT} N_i'$. Then, by Lemma F.10 (value translation reduction), we have $N_i'[\gamma_p] \mapsto_b^* \texttt{ret}\ V_i'[\gamma_p]$ for some V_1', V_2' such that $V_i \rightsquigarrow^{CT} \texttt{ret}\ V_i'$. Thus, we have the following reductions:

$$\begin{array}{ll}
& [\![\Sigma; \cdot \vdash A_1^\sqsubseteq \times A_2^\sqsubseteq]\!]\!\updownarrow [x_1 \leftarrow N_1'; x_2 \leftarrow N_2'; \texttt{ret}\ (x_1, x_2)][\gamma_p] \\
\mapsto_b^* & [\![\Sigma; \cdot \vdash A_1^\sqsubseteq \times A_2^\sqsubseteq]\!]\!\updownarrow [\texttt{ret}\ (V_1', V_2')][\gamma_p] \\
\mapsto_b^* & \texttt{let}\ (x_1, x_2)\ =\ (V_1', V_2')[\gamma_p]; \\
& x_1' \leftarrow [\![\Sigma; \Gamma \vdash A_1^\sqsubseteq]\!]\!\updownarrow [\texttt{ret}\ x_1][\gamma_p]; \\
& x_2' \leftarrow [\![\Sigma; \Gamma \vdash A_2^\sqsubseteq]\!]\!\updownarrow [\texttt{ret}\ x_2][\gamma_p]; \\
& \texttt{ret}\ (x_1', x_2')
\end{array}$$

Let $M_4'[\gamma_p]$ be the latter term. By Lemma F.8 (bind confluence), we have $M_3'[\gamma_p] \mapsto_b^* M_4'[\gamma_p]$. Thus, we have this last reduction:

$$\begin{array}{ll}
& S[M_3'][\gamma_p] \\
\mapsto_b^* & S[M_4'][\gamma_p] \\
\mapsto & S[x_1' \leftarrow [\![\Sigma; \Gamma \vdash A_1^\sqsubseteq]\!]\!\updownarrow [\texttt{ret}\ V_1']; \\
& \quad x_2' \leftarrow [\![\Sigma; \Gamma \vdash A_2^\sqsubseteq]\!]\!\updownarrow [\texttt{ret}\ V_2']; \\
& \quad \texttt{ret}\ (x_1', x_2')][\gamma_p]
\end{array}$$

And, since $V_i \rightsquigarrow^{CT} \texttt{ret}\ V_i'$, we have that this result is related to M_2 by Lemma F.12 (translation context plug), so we conclude.

- Function cast:

$$E[(\langle A_1^\sqsubseteq \to A_2^\sqsubseteq \rangle\!\updownarrow V_1)\ V_2] \mapsto E[\langle A_2^\sqsubseteq \rangle\!\updownarrow (V_1\ \langle A_1^\sqsubseteq \rangle\!\updownarrow^-\ V_2)]$$

We have $(f \leftarrow [\![\Sigma; \cdot \vdash A_1^\sqsubseteq \to A_2^\sqsubseteq]\!]\!\updownarrow [N_1']; a \leftarrow N_2'; \texttt{force}\ f\ a)[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$ where $V_i \rightsquigarrow^{CT} N_i'$. Then, by Lemma F.10 (value translation reduction), we have $N_i'[\gamma_p] \mapsto_b^* \texttt{ret}\ V_i'[\gamma_p]$ for some V_1', V_2' such that $V_i \rightsquigarrow^{CT} \texttt{ret}\ V_i'$. Thus, we have the following reduction:

$$\begin{array}{ll}
& (f \leftarrow [\![\Sigma; \cdot \vdash A_1^\sqsubseteq \to A_2^\sqsubseteq]\!]\!\updownarrow [N_1']; a \leftarrow N_2'; \texttt{force}\ f\ a)[\gamma_p] \\
\mapsto_b^* & (\texttt{force}\ (\texttt{thunk}\ \lambda y : [\![\Sigma; \Gamma \vdash A_{1l}^\sqsubseteq]\!]. \\
& \quad a \leftarrow [\![\Sigma; \Gamma \vdash A_1^\sqsubseteq]\!]\!\updownarrow^-\ [\texttt{ret}\ y]; [\![\Sigma; \Gamma \vdash A_2^\sqsubseteq]\!]\!\updownarrow [\texttt{force}\ V_1'\ a])\ V_2')[\gamma_p] \\
\mapsto^+ & (a \leftarrow [\![\Sigma; \Gamma \vdash A_1^\sqsubseteq]\!]\!\updownarrow^-\ [\texttt{ret}\ V_2']; [\![\Sigma; \Gamma \vdash A_2^\sqsubseteq]\!]\!\updownarrow [\texttt{force}\ V_1'\ a])[\gamma_p]
\end{array}$$

Let $M_4'[\gamma_p]$ be the latter term. Then we have $\langle A_2^\sqsubseteq \rangle\!\updownarrow (V_1\ \langle A_1^\sqsubseteq \rangle\!\updownarrow^-\ V_2 \rightsquigarrow^{CT} M_4'$ since $V_i \rightsquigarrow^{CT} \texttt{ret}\ V_i'$, applying the bind reduction rule to eliminate the first bind. We then conclude by Lemma F.12 (translation context plug).

- Universal cast:

$$E[(\langle\forall^\nu X.A^\sqsubseteq\rangle\updownarrow \Lambda X.N)\{\sigma \cong A_1\}] \mapsto E[\langle A^\sqsubseteq[\sigma/X]\rangle\updownarrow N[\sigma/X]]$$

Let $A_1' = [\![\Sigma;\cdot \vdash A_1]\!]$. We have

$$(x \leftarrow [\![\Sigma;\Gamma \vdash \forall^\nu X.A^\sqsubseteq]\!]\updownarrow[\texttt{ret thunk } \Lambda X.\lambda c_X : \text{Case } X.N'];\texttt{force } x\ [A_1']\ \sigma)[\gamma_p] \mapsto_b^* M_3'[\gamma_p]$$

where $N \rightsquigarrow^{CT} N'$. Let $V' = \texttt{thunk } \Lambda X.\lambda c_X : \text{Case } X.N'$. We then have the following reduction:

$$\begin{array}{rl} & (x \leftarrow [\![\Sigma;\cdot \vdash \forall^\nu X.A^\sqsubseteq]\!]\updownarrow[\texttt{ret } V'];\texttt{force } x\ [A_1']\ \sigma)[\gamma_p] \\ \mapsto_b^* & (\texttt{force (thunk } \Lambda X.\lambda c_X : \text{Case } X.[\![\Sigma;X \vdash A^\sqsubseteq]\!]\updownarrow[\texttt{force } V'\ [X]\ c_X])\ [A_1']\ \sigma)[\gamma_p] \end{array}$$

Then, by Lemma F.8 (bind confluence) and the operational semantics, we have

$$\begin{array}{rl} & S[M_3'][\gamma_p] \\ \mapsto_b^* & S[\texttt{force (thunk } \Lambda X.\lambda c_X : \text{Case } X.[\![\Sigma;X \vdash A^\sqsubseteq]\!]\updownarrow[\texttt{force } V'\ [X]\ c_X])\ [A_1']\ \sigma][\gamma_p] \\ \mapsto^+ & S[([\![\Sigma;X \vdash A^\sqsubseteq]\!]\updownarrow[A_1'/X][\sigma/c_X])[\texttt{force } V'\ [A_1']\ \sigma]][\gamma_p] \\ \mapsto^+ & S[([\![\Sigma;X \vdash A^\sqsubseteq]\!]\updownarrow[A_1'/X][\sigma/c_X])[N'[A_1'/X][\sigma/c_X]]][\gamma_p] \\ = & S[[\![\Sigma;X \vdash A^\sqsubseteq]\!]\updownarrow[N'][A_1'/X][\sigma/c_X]][\gamma_p] \end{array}$$

Let $M_2'[\gamma_p]$ be the latter term. Then, by Lemma F.12 (translation context plug), and since $\langle A^\sqsubseteq[\sigma/X]\rangle\updownarrow N[\sigma/X] \rightsquigarrow^{CT} [\![\Sigma;X \vdash A^\sqsubseteq]\!]\updownarrow[N'][A_1'/X][\sigma/c_X]$, which we have by Lemma F.3 (type variable translation substitution) because $\Sigma;X \vdash N \rightsquigarrow^{CT} N' : B$ and $\sigma : A_1 \in \Sigma$, we may conclude.

- tag upcast:

$$E[\langle \text{tag}_G(A^\sqsubseteq)\rangle\uparrow V] \mapsto E[\texttt{inj}_G\ \langle A^\sqsubseteq\rangle\uparrow V]$$

We have $[\![\Sigma;\cdot \vdash \text{tag}_G(A^\sqsubseteq)]\!]\uparrow[N'][\gamma_p] \mapsto_b^* M_3'[\gamma_p]$ where $V \rightsquigarrow^{CT} N'$. Then, by Lemma F.10 (value translation), we have some V' such that $N'[\gamma_p] \mapsto_b^* \texttt{ret } V'[\gamma_p]$ and $V \rightsquigarrow^{CT} \texttt{ret } V'$. Thus, we have the following reduction:

$$\begin{array}{rl} & [\![\Sigma;\cdot \vdash \text{tag}_G(A^\sqsubseteq)]\!]\uparrow[N'][\gamma_p] \\ \mapsto_b^* & [\![\Sigma;\cdot \vdash \text{tag}_G(A^\sqsubseteq)]\!]\uparrow[\texttt{ret } V'][\gamma_p] \\ = & (r \leftarrow [\![\Sigma;\cdot \vdash A^\sqsubseteq]\!]\uparrow[\texttt{ret } V'];\texttt{inj}_{\text{case}(G)}\ r)[\gamma_p] \end{array}$$

Let the latter term be $M_4'[\gamma_p]$. By Lemma F.7 (bind normalization) there must exist some M_5' such that $S[M_4'][\gamma_p] \mapsto_b^* M_5'[\gamma_p]$ and $M_5'[\gamma_p]$ does not take a bind reduction. Then, by Lemma F.8 (bind confluence), we have that $M_1'[\gamma_p] \mapsto_b^* M_5'[\gamma_p]$. Note that $E[\texttt{inj}_G\ \langle A^\sqsubseteq\rangle\uparrow V] \rightsquigarrow^{CT} S[M_4']$ by Lemma F.12 (translation context plug). Then, by Lemma F.14 (bind reduction preserves translation), we have $E[\texttt{inj}_G\ \langle A^\sqsubseteq\rangle\uparrow V] \rightsquigarrow^{CT} M_5'$. Thus, it suffices to show that $|\langle\text{tag}_G(A^\sqsubseteq)\rangle\uparrow V| > |\texttt{inj}_G\ \langle A^\sqsubseteq\rangle\uparrow V|$, which we have since $|\langle\text{tag}_G(A^\sqsubseteq)\rangle\uparrow V| = 1+2+|A^\sqsubseteq|+|V|$.

- tag downcast:

$$E[\langle \text{tag}_G(A^\sqsubseteq)\rangle\downarrow \texttt{inj}_G\ V] \mapsto E[\langle A^\sqsubseteq\rangle\downarrow V]$$

We have $[\![\Sigma;\Delta \vdash \text{tag}_G(A^\sqsubseteq)]\!]\downarrow[r \leftarrow N';\texttt{ret inj}_{\text{case}(G)}\ r][\gamma_p] \mapsto_b^* M_3'[\gamma_p]$ where $V \rightsquigarrow^{CT} N'$. Then, by Lemma F.10 (value translation), we have some V' such that $N'[\gamma_p] \mapsto_b^* \texttt{ret } V'[\gamma_p]$ and $V \rightsquigarrow^{CT} \texttt{ret } V'$. Thus, we have the following reduction:

$$\begin{array}{rl} & [\![\Sigma;\Delta \vdash \text{tag}_G(A^\sqsubseteq)]\!]\downarrow[r \leftarrow N';\texttt{ret inj}_{\text{case}(G)}\ r][\gamma_p] \\ \mapsto_b^* & [\![\Sigma;\Delta \vdash \text{tag}_G(A^\sqsubseteq)]\!]\downarrow[\texttt{ret inj}_{\text{case}(G)}\ V'][\gamma_p] \\ \mapsto_b^* & \texttt{match case}(G)\ \texttt{with}\ (\texttt{inj}_{\text{case}(G)}\ V')\{\texttt{inj}\ y.[\![\Sigma;\Gamma \vdash A^\sqsubseteq]\!]\downarrow[\texttt{ret } y] \mid \mho\}[\gamma_p] \end{array}$$

Let $M'_4[\gamma_p]$ be the latter term. Then, by Lemma F.8 (bind confluence) and the operational semantics, we have the following reductions:

$$\begin{array}{ll} & M'_1[\gamma_p] \\ \mapsto^*_b & S[\texttt{match case}(G)\,\texttt{with}\ (\texttt{inj}_{\text{case}(G)}\ V')\{\texttt{inj}\ y.[\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\!\downarrow[\texttt{ret}\ y] \mid \mho\}][\gamma_p] \\ \mapsto & S[[\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\!\downarrow[\texttt{ret}\ V']][\gamma_p] \end{array}$$

Then, since $V \rightsquigarrow^{CT} \texttt{ret}\ V'$, we have $\langle A^{\sqsubseteq}\rangle\!\downarrow V \rightsquigarrow^{CT} [\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\!\downarrow[\texttt{ret}\ V'][\gamma_p]$ and we may conclude by Lemma F.12 (translation context plug).

- tag downcast error:

$$E[\langle \text{tag}_G(A^{\sqsubseteq})\rangle\!\downarrow \texttt{inj}_H\ V] \mapsto E[\mho_B] \text{ where } G \neq H$$

We have $[\![\Sigma;\Delta \vdash \text{tag}_G(A^{\sqsubseteq})]\!]\!\downarrow[r \leftarrow N';\texttt{ret inj}_{\text{case}(H)}\ r][\gamma_p] \mapsto^*_b M'_3[\gamma_p]$ where $V \rightsquigarrow^{CT} N'$. Then, by Lemma F.10 (value translation), we have some V' such that $N'[\gamma_p] \mapsto^*_b \texttt{ret}\ V'[\gamma_p]$ and $V \rightsquigarrow^{CT} \texttt{ret}\ V'$. Thus, we have the following reduction:

$$\begin{array}{ll} & [\![\Sigma;\Delta \vdash \text{tag}_G(A^{\sqsubseteq})]\!]\!\downarrow[r \leftarrow N';\texttt{ret inj}_{\text{case}(H)}\ r][\gamma_p] \\ \mapsto^*_b & [\![\Sigma;\Delta \vdash \text{tag}_G(A^{\sqsubseteq})]\!]\!\downarrow[\texttt{ret inj}_{\text{case}(H)}\ V'][\gamma_p] \\ \mapsto^*_b & \texttt{match case}(G)\,\texttt{with}\ (\texttt{inj}_{\text{case}(H)}\ V')\{\texttt{inj}\ y.[\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\!\downarrow[\texttt{ret}\ y] \mid \mho\}[\gamma_p] \end{array}$$

Let $M'_4[\gamma_p]$ be the latter term. Then, by Lemma F.8 (bind confluence) and the operational semantics, we have the following reductions:

$$\begin{array}{ll} & M'_1[\gamma_p] \\ \mapsto^*_b & S[\texttt{match case}(G)\,\texttt{with}\ (\texttt{inj}_{\text{case}(H)}\ V')\{\texttt{inj}\ y.[\![\Sigma;\Gamma \vdash A^{\sqsubseteq}]\!]\!\downarrow[\texttt{ret}\ y] \mid \mho\}][\gamma_p] \\ \mapsto & S[\mho][\gamma_p] \end{array}$$

Then, since $\mho_B \rightsquigarrow^{CT} \mho$, we may conclude by Lemma F.12 (translation context plug).

- Existential upcast:

$$E[\langle \exists^\nu X.A_1^{\sqsubseteq}\rangle\!\uparrow \texttt{pack}^\nu(X \cong A, [A_2^{\sqsubseteq} \updownarrow ...], N_1)] \mapsto E[\texttt{pack}^\nu(X \cong A, [A_1^{\sqsubseteq} \uparrow, A_2^{\sqsubseteq} \updownarrow ...], N_1)]$$

For some N'_1 such that $N_1 \rightsquigarrow^{CT} N'_1$, we have

$$[\![\Sigma;\cdot \vdash \exists^\nu X.A_1^{\sqsubseteq}]\!]\!\uparrow[\texttt{ret pack}(A', \texttt{thunk}\ \lambda c_X : \text{Case}\ A'.N'_{cst})\ \texttt{as}\ [\![\Sigma;\cdot \vdash \exists^\nu X.A_{1r}^{\sqsubseteq}]\!]][\gamma_p] \mapsto^*_b M'_3[\gamma_p]$$

where $N_{cst} = [\![\Sigma;X \cong A' \vdash A_2^{\sqsubseteq}]\!]\updownarrow[\texttt{force}\ (\texttt{thunk}\ (\lambda c_X : \text{Case}\ A'.(...\ N'_1\ ...)))\ c_X]$ and $A' = [\![\Sigma;\cdot \vdash A]\!]$. By Lemma F.8 (bind confluence), we then have the following reductions:

$$\begin{array}{ll} & M'_3[\gamma_p] \\ \mapsto^*_b & \texttt{unpack}\ (Y,f) = \texttt{pack}(A', \texttt{thunk}\ \lambda c_X : \text{Case}\ A'.N'_{cst})\ \texttt{as}\ [\![\exists^\nu X.A_{1l}^{\sqsubseteq}]\!]; \\ & \texttt{ret pack}(Y, \texttt{thunk}\ \lambda c_X : \text{Case}\ Y.[\![\Sigma;Y,X \cong Y \vdash A_1^{\sqsubseteq}]\!]\!\uparrow[\texttt{force}\ f\ c_X])\ \texttt{as}\ [\![\exists^\nu X.A_{1r}^{\sqsubseteq}]\!][\gamma_p] \\ \mapsto^+ & \texttt{ret pack}(A', \texttt{thunk}\ \lambda c_X : \text{Case}\ A'. \\ & \quad [\![\Sigma;X \cong A' \vdash A_1^{\sqsubseteq}]\!]\!\uparrow[\texttt{force}\ (\texttt{thunk}\ \lambda c_X : \text{Case}\ A'.N'_{cst})\ c_X])\ \texttt{as}\ [\![\exists^\nu X.A_{1r}^{\sqsubseteq}]\!][\gamma_p] \end{array}$$

Let the above term be $M'_4[\gamma_p]$. We know that $\texttt{pack}^\nu(X \cong A', [A_1^{\sqsubseteq} \uparrow, A_2^{\sqsubseteq} \updownarrow ...], N_1) \rightsquigarrow^{CT} M'_4$ since $N_1 \rightsquigarrow^{CT} N'_1$, so we conclude by Lemma F.12 (translation context plug).

- Existential downcast:

$$E[\langle \exists^\nu X.A_1^{\sqsubseteq}\rangle\!\downarrow \texttt{pack}^\nu(X \cong A, [A_2^{\sqsubseteq} \updownarrow ...], N_1)] \mapsto E[\texttt{pack}^\nu(X \cong A, [A_1^{\sqsubseteq} \downarrow, A_2^{\sqsubseteq} \updownarrow ...], N_1)]$$

This case proceeds analogously, replacing $A_{1r}^{\sqsubseteq}$ with $A_{1l}^{\sqsubseteq}$ and vice versa.

- Tag check true:

$$E[\texttt{is}(G)?\ \texttt{inj}_G\ V] \mapsto E[\texttt{true}]$$

We have

$$\begin{array}{rl} & (r \leftarrow x \leftarrow N'; \texttt{ret inj}_{\mathrm{case}(G)}\ x; \\ & \texttt{match case}(G)\ \texttt{with}\ r\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\})[\gamma_p] \\ \mapsto_b^* & M_3'[\gamma_p] \end{array}$$

where $V \rightsquigarrow^{CT} N'$. Then, by Lemma F.10 (value translation), we have some V' such that $N'[\gamma_p] \mapsto_b^* \texttt{ret}\ V'[\gamma_p]$ and $V \rightsquigarrow^{CT} \texttt{ret}\ V'$. Thus, we have the following reduction:

$$\begin{array}{rl} & (r \leftarrow x \leftarrow N'; \texttt{ret inj}_{\mathrm{case}(G)}\ x; \\ & \texttt{match case}(G)\ \texttt{with}\ r\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\})[\gamma_p] \\ \mapsto_b^* & (r \leftarrow \texttt{ret inj}_{\mathrm{case}(G)}\ V'; \\ & \texttt{match case}(G)\ \texttt{with}\ r\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\})[\gamma_p] \\ \mapsto_b^* & (\texttt{match case}(G)\ \texttt{with}\ (\texttt{inj}_{\mathrm{case}(G)}\ V')\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\})[\gamma_p] \end{array}$$

Then, by Lemma F.8 (bind confluence), we have

$$\begin{array}{rl} & M_3' \\ \mapsto_b^* & (\texttt{match case}(G)\ \texttt{with}\ (\texttt{inj}_{\mathrm{case}(G)}\ V')\{\texttt{inj}\ y.\texttt{ret true} \mid \texttt{ret false}\})[\gamma_p] \\ \mapsto & \texttt{ret true} \end{array}$$

Since $\texttt{true} \rightsquigarrow^{CT} \texttt{ret true}$, we conclude by Lemma F.12 (translation context plug).

- Tag check false:

$$E[\texttt{is}(G)?\ \texttt{inj}_H\ V] \mapsto E[\texttt{false}] \text{ where } G \neq H$$

This case is analogous to the former except that since $G \neq H$, the translation produces $\texttt{false}$ and since $\texttt{false} \rightsquigarrow^{CT} \texttt{false}$, we conclude.

□

Lemma F.17 (Multi-step simulation). *If $\Sigma; \cdot \vdash M \rightsquigarrow^{CT} M' : A$ and $\Sigma \triangleright M \mapsto^* \Sigma_1 \triangleright M_1$, then $M_1 \rightsquigarrow^{CT} M_1'$ and $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \mapsto^* \Sigma_p, [\![\Sigma_1]\!] \triangleright M_1'[\gamma_p]$ for some M_1'.*

Proof. We proceed by induction on the number of steps n in $\Sigma \triangleright M \mapsto^* \Sigma_1 \triangleright M_1$.

- $n = 0$: Then $M_1 = M$, so we have $M_1 \rightsquigarrow^{CT} M'$ and $M'[\gamma_p] \mapsto^* M'[\gamma_p]$ by reflexivity.
- $n = n' + 1$: Then there exists some Σ_2, M_2 such that $\Sigma \triangleright M \mapsto^* \Sigma_2 \triangleright M_2 \mapsto \Sigma_1 \triangleright M_1$. By the inductive hypothesis for n', we then have some M_2' such that $M_2 \rightsquigarrow^{CT} M_2'$ and

$$\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \mapsto^* \Sigma_p, [\![\Sigma_2]\!] \triangleright M_2'[\gamma_p]$$

Finally, by Theorem F.16 (simulation), since $M_2 \rightsquigarrow^{CT} M_2'$ and $\Sigma_2 \triangleright M_2 \mapsto \Sigma_1 \triangleright M_1$, there exists M_1' such that $M_1 \rightsquigarrow^{CT} M_1'$ and $\Sigma_p, [\![\Sigma_2]\!] \triangleright M_2'[\gamma_p] \mapsto^* \Sigma_p, [\![\Sigma_1]\!] \triangleright M_1'[\gamma_p]$. Therefore, $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \mapsto^* \Sigma_p, [\![\Sigma_1]\!] \triangleright M_1'[\gamma_p]$ as we were required to show.

□

F.1 Adequacy

Now that we have established the multi-step simulation theorem, we can prove our desired adequacy theorems that say that we can tell if a PolyC$^\nu$ term terminates, errors or diverges by looking at its translation to CBPV$_{\mathrm{OSum}}$. They follow by our simulation theorem and progress for PolyC$^\nu$.

Corollary F.18. *If $\Sigma; \cdot \vdash M \rightsquigarrow^{CT} M' : A$, and $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \Uparrow$, then $\Sigma \triangleright M \Uparrow$.*

Proof. Assume the opposite, that $M \mapsto^* M_1$ and M_1 does not take a step. By Lemma F.17 (multi-step simulation), we have some M_1' such that $M_1 \rightsquigarrow^{CT} M_1'$ and $M'[\gamma_p] \mapsto^* M_1'[\gamma_p]$. Then, by Lemma C.3 (progress), $M_1 = \mho$ or $M_1 = V$ for some value V so we have two cases to consider:

- $M_1 = \mho$: By Lemma F.9 (canonical forms), $M_1' = \mho$ and we have $M'[\gamma_p] \mapsto^* \mho$, but $M'[\gamma_p] \Uparrow$, so we have a contradiction.
- $M_1 = V$: By Lemma F.10, we have $M'[\gamma_p] \mapsto^* M_1'[\gamma_p] \mapsto^* \texttt{ret}\ V'$ for some value V', but $M'[\gamma_p] \Uparrow$, so again we have a contradiction.

□

Theorem F.19. *If $\Sigma; \cdot \vdash M \rightsquigarrow^{CT} M' : A$, and $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \Downarrow$, then $\Sigma \triangleright M \Downarrow$.*

Proof. We proceed by induction first on the number of steps $M'[\gamma_p]$ takes and then on $|M|$. Assume that $\Sigma \triangleright M \mapsto \Sigma_1 \triangleright M_1$ since otherwise, we may conclude. By Theorem F.16 (simulation), we have some M_1' such that $M_1 \rightsquigarrow^{CT} M_1'$ and either $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \mapsto^+ \Sigma_p, [\![\Sigma_1]\!] \triangleright M_1'[\gamma_p]$ or $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \mapsto^* \Sigma_p, [\![\Sigma_1]\!] \triangleright M_1'[\gamma_p]$ and $|M| > |M_1|$. Note that since $M'[\gamma_p] \Downarrow$, by Lemma F.1 (target semantics deterministic) we have $M_1'[\gamma_p] \Downarrow$. We then have two cases to consider. If $M' \mapsto^+ M_1'$, then we conclude by the inductive hypothesis for $M_1'[\gamma_p] \Downarrow$. Otherwise, we have that $|M| > |M_1|$, so we conclude by the inductive hypothesis for $|M_1|$.

□

Lemma F.20. *If $\Sigma; \cdot \vdash M \rightsquigarrow^{CT} M' : A$, and $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \Downarrow \mho$, then $\Sigma \triangleright M \Downarrow \mho_A$.*

Proof. By Theorem **??** (target termination implies source termination), we have $\Sigma \triangleright M \Downarrow M_R$ for some M_R. If $M_R = \mho_A$, we conclude. Otherwise, $M_R = \texttt{ret}\ V$ for some V. We prove this case by contradiction. By Lemma F.17 (multi-step simulation), We have some Σ_R, M_R' such that $M_R \rightsquigarrow^{CT} M_R'$ and $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \mapsto^* \Sigma_p, [\![\Sigma_R]\!] \triangleright M_R'[\gamma_p]$. Then, by Lemma F.10 (value translation), we then have $M_R'[\gamma_p] \mapsto_b^* \texttt{ret}\ V'[\gamma_p]$. However, by Lemma F.1 (target language deterministic), we have $M_R'[\gamma_p] \Downarrow \mho$ and we have a contradiction. □

Lemma F.21. *If $\Sigma; \cdot \vdash M \rightsquigarrow^{CT} M' : A$, and $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \Downarrow \texttt{ret}\ V'$, then $\Sigma \triangleright M \Downarrow V$.*

Proof. By Theorem **??** (target termination implies source termination), we have $\Sigma \triangleright M \Downarrow M_R$ for some M_R. If $M_R = V$ for some V, we conclude. Otherwise, $M_R = \mho_A$. We prove this case by contradiction. By Lemma F.17 (multi-step simulation), We have some Σ_R, M_R' such that $M_R \rightsquigarrow^{CT} M_R'$ and $\Sigma_p, [\![\Sigma]\!] \triangleright M'[\gamma_p] \mapsto^* \Sigma_p, [\![\Sigma_R]\!] \triangleright M_R'[\gamma_p]$. Then, by Lemma F.9 (canonical forms), we then have $M_R' = \mho$. However, by Lemma F.1 (target language deterministic), we have $M_R'[\gamma_p] \Downarrow \texttt{ret}\ V'$ and we have a contradiction. □

G GRADUALITY AND PARAMETRICITY

First, we show the full graduality/parametricity logical relation in Figure 36

Definition G.1. We say γ, δ are valid instantiations of $\Gamma^{\sqsubseteq}$ in CBPV$_{\text{OSum}}$, written $(\gamma, \delta) \vDash \Gamma^{\sqsubseteq}$. when

- For each $i \in \{l, r\}$, there exists Σ_i such that for each $(x : A^{\sqsubseteq}) \in \Gamma^{\sqsubseteq}$, $\Sigma_i \mid \cdot \vdash \gamma_i(x) : [\![A_i]\!]$ when $\Gamma^{\sqsubseteq} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r$ and for each $X \in \Gamma^{\sqsubseteq}$, $\cdot \vdash \delta_i(X)$ and $\Sigma_i \mid \cdot \vdash \gamma_i(c_X) : \text{Case}\ \delta_i(X)$.
- For each $X \in \Gamma^{\sqsubseteq}$, $\delta_R(X) \in \text{Rel}_{\omega}[\delta_l(X), \delta_r(X)]$.

$$
\begin{array}{rcl}
\mathcal{V}_n^{\sim}[\![X]\!]\gamma\delta & = & \lfloor \delta(X) \rfloor_n \\
\mathcal{V}_n^{\sim}[\![?]\!]\gamma\delta & = & \{(w, \mathtt{inj}_{\sigma_l}\ V_l, \mathtt{inj}_{\sigma_r}\ V_r) \in \mathrm{Atom}_n[?]\delta \mid w.\eta \vDash (\sigma_l, \sigma_r, R) \land (w, V_l, V_r) \in \triangleright R\} \\
\mathcal{V}_n^{\prec}[\![\mathrm{tag}_G(A^{\sqsubseteq})]\!]\gamma\delta & = & \{(w, V_l, \mathtt{inj}_{\gamma_r(\mathrm{case}(G))}\ V_r) \in \mathrm{Atom}_n[\mathrm{tag}_G(A^{\sqsubseteq})]\delta \mid (w, V_l, V_r) \in \mathcal{V}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta\} \\
\mathcal{V}_n^{\succ}[\![\mathrm{tag}_G(A^{\sqsubseteq})]\!]\gamma\delta & = & \{(w, V_l, \mathtt{inj}_{\gamma_r(\mathrm{case}(G))}\ V_r) \in \mathrm{Atom}_n[\mathrm{tag}_G(A^{\sqsubseteq})]\delta \mid (w, V_l, V_r) \in (\triangleright\mathcal{V}^{\succ}[\![A^{\sqsubseteq}]\!]\gamma\delta)\} \\
\mathcal{V}_n^{\sim}[\![\mathbb{B}]\!]\gamma\delta & = & \{(w, \mathtt{true}, \mathtt{true}) \in \mathrm{Atom}_n[\mathbb{B}]\delta\} \cup \{(w, \mathtt{false}, \mathtt{false}) \in \mathrm{Atom}_n[\mathbb{B}]\delta\} \\
\mathcal{V}_n^{\sim}[\![A_1^{\sqsubseteq} \times A_2^{\sqsubseteq}]\!]\gamma\delta & = & \{((V_{l1}, V_{l2}), (V_{r1}, V_{r2})) \in \mathrm{Atom}_n[A_1^{\sqsubseteq} \times A_2^{\sqsubseteq}]\delta \\
 & & \mid (w, V_{l1}, V_{r1}) \in \mathcal{V}_n^{\sim}[\![A_1^{\sqsubseteq}]\!]\gamma\delta \land (w, V_{l2}, V_{r2}) \in \mathcal{V}_n^{\sim}[\![A_2^{\sqsubseteq}]\!]\gamma\delta\} \\
\mathcal{V}_n^{\sim}[\![A^{\sqsubseteq} \to B^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, V_l, V_r) \in \mathrm{Atom}_n[A^{\sqsubseteq} \to B^{\sqsubseteq}]\delta \mid \forall w' \sqsupseteq w.(w', V_l', V_r') \in \mathcal{V}_n^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta. \\
 & & \quad (w', \mathtt{force}\ V_l\ V_l', \mathtt{force}\ V_r\ V_r') \in \mathcal{E}_n^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta\} \\
\mathcal{V}_n^{\sim}[\![\forall^\nu X.B^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, V_l, V_r) \in \mathrm{Atom}_n[\forall^\nu X.A^{\sqsubseteq}]\delta \mid \\
 & & \quad \forall R \in \mathrm{Rel}_\omega[A_l, A_r].\forall w' \sqsupseteq w.\forall \sigma_l, \sigma_r.w'.\eta \vDash (\sigma_l, \sigma_r, \lfloor R \rfloor_{w'.j}) \implies \\
 & & \quad (w', \mathtt{force}\ V_l\ [A_l]\ \sigma_l, \mathtt{force}\ V_r\ [A_r]\ \sigma_r) \in \mathcal{E}_n^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta' \\
 & & \quad (\text{where } \gamma' = \gamma, c_X \mapsto (\sigma_l, \sigma_r), \text{ and } \delta' = \delta, X \mapsto (A_l, A_r, R))\} \\
\mathcal{V}_n^{\sim}[\![\exists^\nu X.B^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, \mathtt{pack}\ (A_l, V_l), \mathtt{pack}\ (A_r, V_r)) \in \mathrm{Atom}_n[\exists^\nu X.B^{\sqsubseteq}]\delta \mid \\
 & & \quad \exists R \in \mathrm{Rel}_\omega[A_l, A_r].\forall w' \sqsupseteq w.\forall \sigma_l, \sigma_r.w'.\eta \vDash (\sigma_l, \sigma_r, \lfloor R \rfloor_{w'.j}) \implies \\
 & & \quad (\mathtt{force}\ V_l\ \sigma_l, \mathtt{force}\ V_r\ \sigma_r) \in \mathcal{E}_n^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta' \\
 & & \quad (\text{where } \gamma' = \gamma, c_X \mapsto (\sigma_l, \sigma_r), \text{ and } \delta' = \delta, X \mapsto (A_l, A_r, R))\} \\
\mathcal{E}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, M_l, M_r) \in \mathrm{CAtom}_n[A^{\sqsubseteq}]\delta \mid (w.\Sigma_l, M_l) \mapsto^{w.j} \lor ((w.\Sigma_l, M_l) \mapsto^{<w.j} (\Sigma_l', \mho)) \\
 & & \lor(\exists w' \sqsupseteq w.(w', V_l, V_r) \in \mathcal{V}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta. \\
 & & \quad (w.\Sigma_l, M_l) \mapsto^{w'.j - w.j} (w'.\Sigma_l, \mathtt{ret}\ V_l) \land (w.\Sigma_r, M_r) \mapsto^{*} (w'.\Sigma_r, \mathtt{ret}\ V_r))\} \\
\mathcal{E}_n^{\succ}[\![A^{\sqsubseteq}]\!]\gamma\delta & = & \{(w, M_l, M_r) \in \mathrm{CAtom}_n[A^{\sqsubseteq}]\delta \mid (w.\Sigma_r, M_r) \mapsto^{w.j} \lor ((w.\Sigma_l, M_l) \mapsto^{*} (\Sigma_l', \mho)) \\
 & & \lor \exists w' \sqsupseteq w.(w', V_l, V_r) \in \mathcal{V}_n^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta. \\
 & & \quad (w.\Sigma_l, M_l) \mapsto^{*} (w'.\Sigma_l, \mathtt{ret}\ V_l) \land (w.\Sigma_r, M_r) \mapsto^{w'.j - w.j} (w'.\Sigma_r, \mathtt{ret}\ V_r)\}
\end{array}
$$

$$
\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta = \bigcup_{n \in \mathbb{N}} \mathcal{V}_n^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta \qquad \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta = \bigcup_{n \in \mathbb{N}} \mathcal{E}_n^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta
$$

$$
\begin{array}{rcl}
\mathcal{G}^{\sim}[\![\cdot]\!] & = & \{(w, \emptyset, \emptyset) \mid \exists n.w \in \mathrm{World}_n\} \\
\mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}, x : A^{\sqsubseteq}]\!] & = & \{(w, (\gamma, x \mapsto (V_l, V_r)), \delta) \mid (w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!] \land (w, V_l, V_r) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta\} \\
\mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}, X]\!] & = & \{(w, (\gamma, c_X \mapsto (\sigma_l, \sigma_r)), \delta, X \mapsto (A_l, A_r, R)) \mid (w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!] \\
 & & \quad \land R \in \mathrm{Rel}_\omega[A_l, A_r] \land (\sigma_l, \sigma_r, \lfloor R \rfloor_{w.j}) \in w\} \\
\mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}, X \cong A^{\sqsubseteq}]\!] & = & \{(w, (\gamma, c_X \mapsto (\sigma_l, \sigma_r)), \delta, X \mapsto (A_l, A_r, \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta)) \mid (w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!] \\
 & & \quad w \vDash (\sigma_l, \sigma_r, \mathcal{V}_{w.j}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta)\}
\end{array}
$$

$$
\begin{array}{rcl}
\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} & = & \Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq\prec M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \land \Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq\succ M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \\
\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq\sim M_r \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} & = & \forall (w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime}]\!]. \\
 & & \quad (w, [\![M_l]\!][\gamma_l][\delta_l], [\![M_r]\!][\gamma_l][\delta_l]) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta
\end{array}
$$

Fig. 36. Graduality/Parametricity Logical Relation

Definition G.2. We define the extension of an interpretation η with a new association between seals as

$$\eta \boxplus (\sigma_l, \sigma_r, R) = (\eta.size + 1, (f, \eta.size \mapsto (\sigma_l, \sigma_r)), (\rho, \eta.size \mapsto R))$$

Logical Lemmas

Lemma G.3. *If $R \in Rel_\omega[A_l, A_r]$, then $\lfloor R \rfloor_n \in Rel_n[A_l, A_r]$*

Proof. Direct by definition. ◻

Lemma G.4. $\lfloor \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta \rfloor_n = \mathcal{V}^{\sim}_n[\![A^{\sqsubseteq}]\!]\gamma\delta$

Proof. Direct by definition. □

Lemma G.5. *If $R \in Rel_n[A_l, A_r]$, then $\triangleright R \in Rel_n[A_l, A_r]$*

Proof. If $(w, V_l, V_r) \in (\triangleright R)$ and $w' \sqsupseteq w$, then to show $(w', V_l, V_r) \in \triangleright R$, we need to show that for any $w'' \sqsupset w'$, that $(w'', V_l, V_r) \in R$, but this follows because $(w, V_l, V_r) \in \triangleright R$ and $w'' \sqsupset w' \sqsupseteq w$. □

Lemma G.6. *Let $\Gamma^{\sqsubseteq}$ be a well-formed context, with $\Gamma^{\sqsubseteq} \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r$. If $(\gamma, \delta) \vDash \Gamma^{\sqsubseteq}$, then $\mathcal{V}^{\sim}_n[\![A^{\sqsubseteq}]\!]\gamma\delta \in Rel_n[A_l, A_r]$.*

Proof. By induction on $A^{\sqsubseteq}$.

(1) X: by lemma G.3.
(2) ?: If $(w, \mathtt{inj}_{\sigma_l}\ V_l, \mathtt{inj}_{\sigma_r}\ V_r) \in \mathcal{V}^{\sim}_n[\![?]\!]\gamma\delta$ and $w' \sqsupseteq w$, then there exists $R \in \mathrm{Rel}_n[A_l, A_r]$ with $w.\eta \vDash (\sigma_l, \sigma_r, R)$ and $(w, V_l, V_r) \in \triangleright R$. By definition of $\sqsupseteq$, we have that $w' \vDash (\sigma_l, \sigma_r, \lfloor R \rfloor_{w'.j})$ so it is sufficient to show $(w, V_l, V_r) \in \triangleright \lfloor R \rfloor_{w'.j}$, which follows by lemma G.5 that later preserves monotonicity.
(3) $\mathrm{tag}_G(A^{\sqsubseteq})$: by inductive hypothesis, using lemma G.5 in the $>$ case.
(4) $\mathbb{B}$: immediate
(5) $\times$: immediate by inductive hypothesis.
(6) $\to$: If $(w, V_l, V_r) \in \mathcal{V}^{\sim}_n[\![A^{\sqsubseteq} \to B^{\sqsubseteq}]\!]\gamma\delta$ and $w' \sqsupseteq w$. Then given $w'' \sqsupseteq w'$ and $(w'', V_l', V_r') \in \mathcal{V}^{\sim}_n[\![A^{\sqsubseteq}]\!]\gamma\delta$, we need to show $(w'', \mathtt{force}\ \ V_l\, V_l', \mathtt{force}\ \ V_r\, V_r')\mathcal{E}^{\sim}_n[\![B^{\sqsubseteq}]\!]\gamma\delta$, but this holds by relatedness of V_l, V_r because $w'' \sqsupseteq w$ by transitivity of world extension.
(7) $\forall^{\nu}$, $\exists^{\nu}$: similar to the $\to$ case.

□

Lemma G.7. *If $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!]$ and $w' \sqsupseteq w$, then $(w', \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!]$.*

Proof. By induction on $\Gamma^{\sqsubseteq}$, uses monotonicity of $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]$ □

Corollary G.8. $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta \in Rel_{\omega}[A_l, A_r]$

Lemma G.9 (Anti-Reduction). (1) *If $w' \sqsupseteq w$ and $(w.\Sigma_l, M_l) \mapsto^{w.j - w'.j} (w'.\Sigma_l, M_l')$ and $(w.\Sigma_r, M_r) \mapsto^{*} (w'.\Sigma_r, M_r')$ and $(w', M_l', M_r') \in \mathcal{E}^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta$, then $(w, M_l, M_r) \in \mathcal{E}^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta$.*
(2) *If $w' \sqsupseteq w$ and $(w.\Sigma_l, M_l) \mapsto^{w.j - w'.j} (w'.\Sigma_l, M_l')$ and $(w.\Sigma_r, M_r) \mapsto^{*} (w'.\Sigma_r, M_r')$ and $(w', M_l', M_r') \in \mathcal{E}^{\succ}[\![A^{\sqsubseteq}]\!]\gamma\delta$, then $(w, M_l, M_r) \in \mathcal{E}^{\succ}[\![A^{\sqsubseteq}]\!]\gamma\delta$.*

Proof. We do the $\prec$ case, the other is symmetric. By case analysis on $(w', M_l', M_r') \in \mathcal{E}^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta$.

(1) If $w'.\Sigma_l, M_l' \mapsto^{w'.j+1}$, then $(w.\Sigma_l, M_l) \mapsto^{w.j - w'.j + w'.j + 1}$ and $w.j - w'.j + w'.j + 1 = w.j + 1$.
(2) If $w'.\Sigma_l, M_l' \mapsto^{j} \Sigma_l', \mho$, with $j \le w.j$, then $w.\Sigma_l, M_l \mapsto^{w.j - w'.j + j} \Sigma_l', \mho$ and $w.j - w'.j + j \le w.j$ since $j - w'.j \le 0$.
(3) Finally, if there is some $w'' \sqsupseteq w'$ and $(w'', V_l, V_r) \in \mathcal{V}^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta$ with $w'.\Sigma_l, M_l' \mapsto^{w'j. - w''.j} w''.\Sigma_l, \mathtt{ret}\ \ V_l$ and $w'.\Sigma_r, M_r' \mapsto^{*} w'', \Sigma_r, \mathtt{ret}\ \ V_r$, then $w.\Sigma_l, M_l \mapsto^{w.j - w'.j + w'.j - w''.j} w''.\Sigma_l, \mathtt{ret}\ \ V_l''$ and $w.\Sigma_r, M_r \mapsto^{*} w'', \Sigma_r, \mathtt{ret}\ \ V_r$ and $w.j - w'.j + w'.j - w''.j = w.j - w''.j$ so the result holds.

□

Lemma G.10 (Pure Anti-Reduction). *If $(w, M_l', M_r') \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$ and $(w.\Sigma_l, M_l) \mapsto^{0} (w.\Sigma_l, M_l')$ and $(w.\Sigma_r, M_r) \mapsto^{0} (w.\Sigma_r, M_r')$, then $(w, M_l, M_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$.*

Proof. Immediate corollary of anti-reduction lemma G.9 □

Lemma G.11 (Pure Forward Reduction). *If* $(w, M_l', M_r') \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$ *and* $(w.\Sigma_l, M_l) \mapsto^0 (w.\Sigma_l, M_l')$ *and* $(w.\Sigma_r, M_r) \mapsto^0 (w.\Sigma_r, M_r')$*, then* $(w, M_l, M_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$.

Proof. By determinism of evaluation. □

Lemma G.12 (Monadic bind). *If* $(w, M_l, M_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$ *and for all* $w' \sqsupseteq w$*, and* $(w', V_l, V_r) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$*,* $(w', S_l[\mathtt{ret}\ V_l], S_r[\mathtt{ret}\ V_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$*, then* $(w, S_l[M_l], S_r[M_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$.

Proof. We show the proof for $\mathcal{E}^{\prec}[\![A^{\sqsubseteq}]\!]$, the $\succ$ case is symmetric. By case analysis on $(w, M_l, M_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$.

(1) If $w.\Sigma_l, M_l \mapsto^{w.j+1}$, then $w.\Sigma_l, S[M_l] \mapsto^{w.j+1}$.
(2) If $w.\Sigma_l, M_l \mapsto^{j} w.\Sigma_l', \mho$, then $w.\Sigma_l, S[M_l] \mapsto^{j} w.\Sigma_l', \mho$.
(3) Otherwise there exists w' and $(w', V_l, V_r) \in \mathcal{V}^{\prec}[\![A^{\sqsubseteq}]\!]\gamma\delta$ with $w.\Sigma_l, M_l \mapsto^{w.j-w'.j} w'.\Sigma_l, \mathtt{ret}\ V_l$ and $w.\Sigma_r, M_r \mapsto^{*} w'.\Sigma_r, \mathtt{ret}\ V_r$. Then $w.\Sigma_l, S_l[M_l] \mapsto^{w.j-w'.j} w'.\Sigma_l, S[\mathtt{ret}\ V_l]$ and $w.\Sigma_r, S_r[M_r] \mapsto^{*} w'.\Sigma_r, S_r[\mathtt{ret}\ V_r]$, and the result follows by the assumption.

□

Pure evaluation is monotone.

Lemma G.13. *If* $\Sigma, M \mapsto^{*} \Sigma, N$*, then for any* $\Sigma' \sqsupseteq \Sigma$*,* $\Sigma', M \mapsto^{*} \Sigma', N$.

Clamping

Lemma G.14. *If* $(w, V_l, V_r) \in R$ *and* $w.j \leq n$*, then* $(w, V_l, V_r) \in \lfloor R \rfloor_n$.

Proof. Direct from definition. □

Tag-to-type

Lemma G.15. $\mathcal{V}_n^{\sim}[\![G]\!]\gamma\delta = \lfloor \delta_R(\mathit{case}(G)) \rfloor_n$

Proof. Direct from definition □

Lemma G.16 (Weakening). *If* $\Gamma^{\sqsubseteq} \vdash A^{\sqsubseteq}$ *and* $\Gamma^{\sqsubseteq} \subseteq \Gamma^{\sqsubseteq\prime}$ *and* $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!]$ *and* $(w, \gamma', \delta') \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq\prime}]\!]$*, where* $\gamma \subseteq \gamma'$ *and* $\delta \subseteq \delta'$*, then all of the following are true:*

$$\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta = \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma'\delta'$$

$$\mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta = \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma'\delta'$$

Proof. Straightforward, by induction over $\Gamma^{\sqsubseteq\prime}$. □

G.1 Cast Lemmas

To prove the cast left lemma, we need the following lemma that casts always either error or terminate with a value on well-typed inputs.

Lemma G.17 (Casts don't diverge). *If* $\Gamma \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r$*, then for any* Σ*, and* $\Sigma \mid \cdot \vdash \gamma : [\![\Gamma]\!]$*,*

(1) *If* $\Sigma \mid \cdot \vdash V : A_l$*, then either* $\Sigma, [\![\langle A^{\sqsubseteq} \rangle\!\downarrow]\!][\mathtt{ret}\ V_l][\gamma] \mapsto^{*} \Sigma, \mho$ *or* $\Sigma, [\![\langle A^{\sqsubseteq} \rangle\!\downarrow]\!][\mathtt{ret}\ V_l][\gamma] \mapsto^{*} \Sigma, \mathtt{ret}\ V'$.
(2) *If* $\Sigma \mid \cdot \vdash V : A_r$*, then either* $\Sigma, [\![\langle A^{\sqsubseteq} \rangle\!\downarrow]\!][\mathtt{ret}\ V][\gamma] \mapsto^{*} \Sigma, \mho$ *or* $\Sigma, [\![\langle A^{\sqsubseteq} \rangle\!\downarrow]\!][\mathtt{ret}\ V][\gamma] \mapsto^{*} \Sigma, \mathtt{ret}\ V'$.

Proof. By induction on $A^{\sqsubseteq}$.

(1) If $A^{\sqsubseteq} \in \{?, \mathbb{B}\}$ the cast is trivial.
(2) Case $A^{\sqsubseteq} = \mathrm{tag}_G(AG^{\sqsubseteq})$:

(a) The upcast definition expands as follows:

$$[\![\langle \mathrm{tag}_G(AG^{\sqsubseteq})\rangle\uparrow]\!][\mathtt{ret}\ V][\gamma] = x \leftarrow [\![\langle AG^{\sqsubseteq}\rangle\uparrow]\!][\mathtt{ret}\ V][\gamma]; \mathtt{ret}\ \mathtt{inj}_\sigma\ x$$

where $\sigma = \gamma(\mathrm{case}(G))$. By inductive hypothesis, $[\![\langle AG^{\sqsubseteq}\rangle\uparrow]\!][\mathtt{ret}\ V][\gamma]$ either errors (in which case the whole term errors), or runs to a value V', in which case

$$x \leftarrow \mathtt{ret}\ V'; \mathtt{ret}\ \mathtt{inj}_\sigma\ x \mapsto^* \mathtt{ret}\ \mathtt{inj}_\sigma\ V'$$

(b) The downcast definition expands as follows:

$$[\![\langle \mathrm{tag}_G(AG^{\sqsubseteq})\rangle\downarrow]\!][\mathtt{ret}\ V][\gamma] = x \leftarrow \mathtt{ret}\ V; \mathtt{match}\ x\,\mathtt{with}\ \sigma\{\mathtt{inj}\ y.[\![\langle AG^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ y][\gamma] \mid \mho\}$$

Since $\Sigma \mid \cdot V : \mathrm{OSum}$, $V = \mathtt{inj}_{\sigma'}\ V'$ for some $\sigma' \in \Sigma$.

(i) If $\sigma' = \sigma$, then

$$\mathtt{match}\ \mathtt{inj}_\sigma\ V'\,\mathtt{with}\ \sigma\{\mathtt{inj}\ y.[\![\langle AG^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ y][\gamma] \mid \mho\} \mapsto^1 [\![\langle AG^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V'][\gamma]$$

and then it follows by inductive hypothesis with $AG^{\sqsubseteq}$.

(ii) If $\sigma' \neq \sigma$, then

$$\mathtt{match}\ \mathtt{inj}_\sigma\ V'\,\mathtt{with}\ \sigma\{\mathtt{inj}\ y.[\![\langle AG^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ y][\gamma] \mid \mho\} \mapsto^1 \mho$$

and the result holds.

(3) If $A^{\sqsubseteq} = A_1^{\sqsubseteq} \times A_2^{\sqsubseteq}$, we consider the downcast case, the upcast is entirely symmetric. First,

$$\begin{aligned}[\![\langle A_1^{\sqsubseteq} \times A_2^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V][\gamma] = {} & x \leftarrow \mathtt{ret}\ V;\\ & \mathtt{let}\ (x_1, x_2)\ =\ x;\\ & y_1 \leftarrow [\![\langle A_1^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ x_1][\gamma];\\ & y_2 \leftarrow [\![\langle A_2^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ x_2][\gamma];\\ & \mathtt{ret}\ (y_1, y_2)\end{aligned}$$

Next, since V is well-typed, $V = (V_1, V_2)$. Then,

$$\begin{array}{ll} x \leftarrow \mathtt{ret}\ V; & \mapsto^0 y_1 \leftarrow [\![\langle A_1^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V_1][\gamma];\\ \mathtt{let}\ (x_1, x_2)\ =\ x; & \quad\ \ y_2 \leftarrow [\![\langle A_2^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V_2][\gamma];\\ y_1 \leftarrow [\![\langle A_1^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ x_1][\gamma]; & \quad\ \ \mathtt{ret}\ (y_1, y_2)\\ y_2 \leftarrow [\![\langle A_2^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ x_2][\gamma]; & \\ \mathtt{ret}\ (y_1, y_2) & \end{array}$$

Applying the inductive hypothesis to $A_1^{\sqsubseteq}$, either $[\![\langle A_1^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V_1][\gamma]$ errors (in which case the whole term errors), or it runs to a value V_1'. Then we need to show

$$\begin{aligned}& y_2 \leftarrow [\![\langle A_2^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V_2][\gamma];\\ & \mathtt{ret}\ (V_1', y_2)\end{aligned}$$

errors or terminates. Applying the inductive hypothesis to $A_2^{\sqsubseteq}$, either $[\![\langle A_2^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V_2][\gamma]$ errors (in which case the whole term errors), or it runs to a value V_2'. Then the whole term runs to $\mathtt{ret}\ (V_1', V_2')$.

(4) If $A^{\sqsubseteq} = A_i^{\sqsubseteq} \to A_o^{\sqsubseteq}$, we consider the downcast case (upcast is symmetric). The downcast definition expands as follows:

$$[\![\langle A_i^{\sqsubseteq} \to A_o^{\sqsubseteq}\rangle\downarrow]\!][\mathtt{ret}\ V][\gamma] = f \leftarrow \mathtt{ret}\ V; \mathtt{ret}\ \mathtt{thunk}\ \lambda x.[\![\langle A_o^{\sqsubseteq}\rangle\downarrow]\!][y \leftarrow [\![\langle A_i^{\sqsubseteq}\rangle\uparrow]\!][\mathtt{ret}\ x][\gamma]; \mathtt{force}\ f\ y][\gamma]$$

Which steps immediately to a value.

(5) If $A^{\sqsubseteq} = \forall^\nu X.A_o^{\sqsubseteq}$, then it follows by similar reasoning to the function case, that is, it immediately terminates.

(6) If $A^{\sqsubseteq} = \exists^{\nu} X.A_o^{\sqsubseteq}$, we consider the downcast case (upcast is symmetric). The definition expands as follows:

$$[\![\langle \exists^{\nu} X.A_o^{\sqsubseteq} \rangle\!\downarrow]\!][\texttt{ret } V][\gamma] = \texttt{unpack } (X, x) = \texttt{ret } V; \texttt{ret thunk } \lambda c_X.[\![\langle A_o^{\sqsubseteq} \rangle\!\downarrow]\!][\texttt{force } x\, c_X]$$

Which steps immediately to a value.

□

Lemma G.18 (Cast Right). *For any $\Gamma^{\sqsubseteq} : \Gamma$, if $\Gamma^{\sqsubseteq} \vdash AC^{\sqsubseteq} : A \sqsubseteq C$ and $\Gamma^{\sqsubseteq} \vdash AB^{\sqsubseteq} : A \sqsubseteq B$, and $\Gamma' \vdash BC^{\sqsubseteq} : B \sqsubseteq C$, Then if $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!]$,*

(1) *If $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![AB^{\sqsubseteq}]\!]\gamma\delta$, then $(w, \texttt{ret } V_l, [\![\langle BC^{\sqsubseteq} \rangle\!\uparrow]\!][\texttt{ret } V_r][\gamma_r]) \in \mathcal{E}^{\sim}[\![AC^{\sqsubseteq}]\!]\gamma\delta$*
(2) *If $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![AC^{\sqsubseteq}]\!]\gamma\delta$, then $(w, \texttt{ret } V_l, [\![\langle BC^{\sqsubseteq} \rangle\!\downarrow]\!][\texttt{ret } V_r][\gamma_r]) \in \mathcal{E}^{\sim}[\![AB^{\sqsubseteq}]\!]\gamma\delta$*

Proof. By induction on $BC^{\sqsubseteq}$.

(1) If $BC^{\sqsubseteq} \in \{\mathbb{B}, ?, X\}$, then the cast is trivial.
(2) If $BC^{\sqsubseteq} = \mathrm{tag}_G(BG^{\sqsubseteq})$, then $AC^{\sqsubseteq} = \mathrm{tag}_G(AG^{\sqsubseteq})$.
  (a) For the upcast case, we are given that $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![AB^{\sqsubseteq}]\!]\gamma\delta$ and we need to prove that

$$(w, \texttt{ret } V_l, y \leftarrow [\![\langle BG^{\sqsubseteq} \rangle\!\uparrow]\!][V_r][\gamma_r]; \texttt{ret inj}_G\ y) \in \mathcal{E}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

  By inductive hypothesis, we know

$$(w, \texttt{ret } V_l, [\![\langle BG^{\sqsubseteq} \rangle\!\uparrow]\!]V_r[\gamma_r]) \in \mathcal{E}^{\sim}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

  We then use monadic bind (lemma G.12). Suppose $w' \sqsupseteq w$ and $(w', V_l', V_r') \in \mathcal{V}^{\sim}[\![AG^{\sqsubseteq}]\!]\gamma\delta$. We need to show that

$$(w', \texttt{ret } V_l', y \leftarrow \texttt{ret } V_r'; \texttt{ret inj}_G\ y) \in \mathcal{E}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

  By anti reduction, it is sufficient to show

$$(w', V_l, \texttt{inj}_{\gamma_r(G)}\ V_r) \in \mathcal{V}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

  (i) If $\sim = \prec$, we need to show $(w', V_l, V_r) \in \mathcal{V}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$, which follows by inductive hypothesis.
  (ii) If $\sim = \succ$, we need to show $(w', V_l, V_r) \in \triangleright\mathcal{V}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$, that is for any $w'' \sqsupset w'$, $(w'', V_l, V_r) \in \mathcal{V}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$ which also follows by inductive hypothesis.
  (b) For the downcast case, we know $(V_l, V_r) \in \mathcal{V}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]w\gamma$. Let $\sigma_r = \gamma_r(\mathrm{case}(G))$.
  (i) In the $\prec$ case, we know $V_r = \texttt{inj}_{\sigma_r}\ V_r'$ and

$$(w, V_l, V_r') \in \mathcal{V}^{\prec}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

  we need to show

$$(w, \texttt{ret } V_l, \texttt{match } (\texttt{inj}_{\sigma_r}\ V_r') \texttt{ with } \sigma_r\{\texttt{inj } x.[\![\langle BG^{\sqsubseteq} \rangle\!\downarrow]\!][x][\gamma_r] \mid \mho\}) \in \mathcal{E}^{\prec}[\![AB^{\sqsubseteq}]\!]\gamma\delta$$

  the right hand side reduces to $[\![\langle BG^{\sqsubseteq} \rangle\!\downarrow]\!][V'']$, so by anti-reduction it is sufficient to show

$$(w, \texttt{ret } V_l, [\![\langle BG^{\sqsubseteq} \rangle\!\downarrow]\!][V_r']) \in \mathcal{E}^{\prec}[\![AB^{\sqsubseteq}]\!]\gamma\delta$$

  which follows by inductive hypothesis.
  (ii) In the $\succ$ case, we know $V_r = \texttt{inj}_{\sigma_r}\ V_r'$ and

$$(w, V, V_r') \in \triangleright(\mathcal{V}^{\prec}[\![AG^{\sqsubseteq}]\!]\gamma\delta)$$

  (note the $\triangleright$). We need to show

$$(w, \texttt{ret } V_l, \texttt{match } (\texttt{inj}_{\sigma_r}\ V_r') \texttt{ with } \sigma_r\{\texttt{inj } x.[\![\langle BG^{\sqsubseteq} \rangle\!\downarrow]\!][x][\gamma_r] \mid \mho\}) \in \mathcal{E}^{\succ}[\![AB^{\sqsubseteq}]\!]\gamma\delta$$

  the right hand side takes 1 step to $[\![\langle BG^{\sqsubseteq} \rangle\!\downarrow]\!][V_r']$.
  (A) If $w.j = 0$, then we are done.

(B) Otherwise, define $w' = (w.j-1, w.\Sigma_l, w.\Sigma_r, \lfloor w.\eta \rfloor_{w.j-1})$. Then by anti-reduction, it is sufficient to show

$$(w', \texttt{ret } V_l, [\![\langle BG^{\sqsubseteq}\rangle\downarrow]\!][V_r']) \in \mathcal{E}^{\succ}[\![AB^{\sqsubseteq}]\!]\gamma\delta$$

$$(w', \texttt{ret } V_l, [\![\langle BG^{\sqsubseteq}\rangle\downarrow]\!][V_r']) \in \mathcal{E}^{\succ}[\![AB^{\sqsubseteq}]\!]$$

By inductive hypothesis, it is sufficient to show

$$(w', V_l, V_r') \in \mathcal{V}^{\prec}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

which follows by assumption because $w' \sqsupset w$.

(3) If $BC^{\sqsubseteq} = BC_1^{\sqsubseteq} \times BC_2^{\sqsubseteq}$, then by precision inversion also $AC^{\sqsubseteq} = AC_1^{\sqsubseteq} \times AC_2^{\sqsubseteq}$ and $AB^{\sqsubseteq} = AB_1^{\sqsubseteq} \times AB_2^{\sqsubseteq}$. We consider the upcast case, the downcast case follows by an entirely analogous argument.

Given $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![BC_1^{\sqsubseteq} \times BC_2^{\sqsubseteq}]\!]\gamma\delta$, we need to show

$$(w, \texttt{ret } V_l, [\![\langle BC_1^{\sqsubseteq} \times BC_2^{\sqsubseteq}\rangle\uparrow]\!][\texttt{ret } V_l][\gamma_r]) \in \mathcal{E}^{\sim}[\![AC_1^{\sqsubseteq} \times AC_2^{\sqsubseteq}]\!]\gamma\delta$$

Expanding definitions, and applying anti-reductino , this reduces to showing

$$\begin{aligned}(w, \texttt{ret } V_l, &\texttt{let } (y_1, y_2) = V_r; \qquad\qquad) \in \mathcal{E}^{\sim}[\![AC_1^{\sqsubseteq} \times AC_2^{\sqsubseteq}]\!]\gamma\delta\\ &z_1 \leftarrow [\![\langle BC_1^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } y_1][\gamma_r];\\ &z_2 \leftarrow [\![\langle BC_2^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } y_2][\gamma_r];\\ &\texttt{ret } (z_1, z_2)\end{aligned}$$

Since $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![BC_1^{\sqsubseteq} \times BC_2^{\sqsubseteq}]\!]\gamma\delta$, we know

$$V_l = (V_{l1}, V_{l2}) \qquad\qquad V_r = (V_{r1}, V_{r2})$$

$$(V_{l1}, V_{r1}) \in \mathcal{V}^{\sim}[\![BC_1^{\sqsubseteq}]\!]\gamma\delta \qquad\qquad (V_{l2}, V_{r2}) \in \mathcal{V}^{\sim}[\![BC_2^{\sqsubseteq}]\!]\gamma\delta$$

So after a reduction we need to show

$$\begin{aligned}(w, \texttt{ret } V_l, &z_1 \leftarrow [\![\langle BC_1^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_{r1}][\gamma_r];) \in \mathcal{E}^{\sim}[\![AC_1^{\sqsubseteq} \times AC_2^{\sqsubseteq}]\!]\gamma\delta\\ &z_2 \leftarrow [\![\langle BC_2^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_{r2}][\gamma_r];\\ &\texttt{ret } (z_1, z_2)\end{aligned}$$

By forward reduction, it is sufficient to prove the following, (which is amenable to monadic bind):

$$\begin{aligned}(w, &z \leftarrow \texttt{ret } V_{l1};_1, z_1 \leftarrow [\![\langle BC_1^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_{r1}][\gamma_r];) \in \mathcal{E}^{\sim}[\![AC_1^{\sqsubseteq} \times AC_2^{\sqsubseteq}]\!]\gamma\delta w\gamma\\ &z_2 \leftarrow \texttt{ret } V_{l2};\ z_2 \leftarrow [\![\langle BC_2^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_{r2}][\gamma_r];\\ &\texttt{ret } (z_1, z_2) \quad \texttt{ret } (z_1, z_2)\end{aligned}$$

We then apply monadic bind with the inductive hypothesis for $BC_1^{\sqsubseteq}$. Given $w' \sqsupseteq w$ and $(V_{l1}', V_{r1}') \in \mathcal{V}^{\sim}[\![AC_1^{\sqsubseteq}]\!]\gamma\delta$ the goal reduces to

$$\begin{aligned}(w', &z_2 \leftarrow \texttt{ret } V_{l2};, z_2 \leftarrow [\![\langle BC_2^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_{r2}][\gamma_r];) \in \mathcal{E}^{\sim}[\![AC_1^{\sqsubseteq} \times AC_2^{\sqsubseteq}]\!]\gamma\delta\\ &\texttt{ret } (V_{l1}', z_2) \quad \texttt{ret } (V_{r1}', z_2)\end{aligned}$$

We then apply another monadic bind with the inductive hypothesis for $BC_2^{\sqsubseteq}$. Given $w'' \sqsupseteq w'$ and $(V_{l2}', V_{r2}') \in \mathcal{V}^{\sim}[\![AC_2^{\sqsubseteq}]\!]w'\gamma$, the goal reduces to

$$(w'', (V_{l1}', V_{l2}'), (V_{r1}', V_{r2}')) \in \mathcal{V}^{\sim}[\![AC_1^{\sqsubseteq} \times AC_2^{\sqsubseteq}]\!]w''\gamma$$

which follows immediately by our assumptions from monadic bind.

(4) If $BC^{\sqsubseteq} = BC_i^{\sqsubseteq} \to BC_o^{\sqsubseteq}$, then by precision inversion also $AC^{\sqsubseteq} = AC_i^{\sqsubseteq} \to AC_o^{\sqsubseteq}$ and $AB^{\sqsubseteq} = AB_i^{\sqsubseteq} \to AB_o^{\sqsubseteq}$. We consider the upcast case, the downcast case follows by an entirely analogous argument.
Given $(V_l, V_r) \in \mathcal{V}^{\sim}[\![BC_i^{\sqsubseteq} \to BC_o^{\sqsubseteq}]\!]\gamma\delta$, we need to show

$$(w, \mathtt{ret}\ V_l, [\![\langle BC_i^{\sqsubseteq} \to BC_o^{\sqsubseteq}\rangle\!\uparrow]\!][\mathtt{ret}\ V_r][\gamma_r]) \in \mathcal{E}^{\sim}[\![AC_i^{\sqsubseteq} \to AC_o^{\sqsubseteq}]\!]\gamma\delta$$

Expanding definitions, this reduces to showing

$$\begin{aligned}(w, V_l, \mathtt{thunk}\ (\lambda x.y \leftarrow\ & [\![\langle BC_i^{\sqsubseteq}\rangle\!\downarrow]\!][\mathtt{ret}\ x][\gamma_r];)) \in \mathcal{V}^{\sim}[\![AC_i^{\sqsubseteq} \to AC_o^{\sqsubseteq}]\!]\gamma\delta\\ & z \leftarrow \mathtt{force}\ V_r y;\\ & [\![\langle BC_o^{\sqsubseteq}\rangle\!\uparrow]\!][\mathtt{ret}\ z][\gamma_r]\end{aligned}$$

Let $w' \sqsupseteq w$ be a future world and $(w', V_{li}, V_{ri}) \in \mathcal{V}^{\sim}[\![AC_i^{\sqsubseteq}]\!]\gamma\delta$. Then our goal reduces to showing

$$\begin{aligned}(w', \mathtt{force}\ V_l V_{li}, y \leftarrow\ & [\![\langle BC_i^{\sqsubseteq}\rangle\!\downarrow]\!][\mathtt{ret}\ V_{ri}][\gamma_r];) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma\delta\\ & z \leftarrow \mathtt{force}\ V_r y;\\ & [\![\langle BC_o^{\sqsubseteq}\rangle\!\uparrow]\!][\mathtt{ret}\ z][\gamma_r]\end{aligned}$$

by forward reduction, it is sufficient to show

$$\begin{aligned}(w', y \leftarrow V_{li};\quad &, y \leftarrow [\![\langle BC_i^{\sqsubseteq}\rangle\!\downarrow]\!][\mathtt{ret}\ V_{ri}][\gamma_r];) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma\delta\\ \mathtt{force}\ V_l y\ & z \leftarrow \mathtt{force}\ V_r y;\\ & [\![\langle BC_o^{\sqsubseteq}\rangle\!\uparrow]\!][\mathtt{ret}\ z][\gamma_r]\end{aligned}$$

We then use the inductive hypothesis on $BC_i^{\sqsubseteq}$ (which applies because of downward-closure) and monadic bind: assume $w'' \sqsupseteq w'$ and $(w'', V'_{li}, V'_{ri}) \in \mathcal{V}^{\sim}[\![AC_i^{\sqsubseteq}]\!]\gamma\delta$. We need to show

$$\begin{aligned}(w'', \mathtt{force}\ V_l V'_{li}, z \leftarrow\ & \mathtt{force}\ V_r V'_{ri};\quad) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma\delta\\ & [\![\langle BC_o^{\sqsubseteq}\rangle\!\uparrow]\!][\mathtt{ret}\ z][\gamma_r]\end{aligned}$$

We apply monadic bind again, noting that the applications are related by assumption and downward closure. Assume $w''' \sqsupseteq w''$ and $(w''', V_{lo}, V_{ro}) \in \mathcal{V}^{\sim}[\![AB_o^{\sqsubseteq}]\!]\gamma\delta$. By anti-reduction , the goal reduces to showing

$$(\mathtt{ret}\ V_o, [\![\langle BC_o^{\sqsubseteq}\rangle\!\uparrow]\!][\mathtt{ret}\ V_o'][\gamma]) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma\delta$$

which follows by inductive hypothesis for $BC_o^{\sqsubseteq}$.

(5) If $BC^{\sqsubseteq} = \forall^{\nu}X.BC_o^{\sqsubseteq}$, then by precision inversion also $AC^{\sqsubseteq} = \forall^{\nu}X.AC_o^{\sqsubseteq}$ and $AB^{\sqsubseteq} = \forall^{\nu}X.AB_o^{\sqsubseteq}$. We consider the upcast case, the downcast case follows by an entirely analogous argument.
Given $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![\forall^{\nu}X.BC_o^{\sqsubseteq}]\!]\gamma\delta$, we need to show

$$(w, \mathtt{ret}\ V_l, [\![\langle \forall^{\nu}X.BC_o^{\sqsubseteq}\rangle\!\uparrow]\!][\mathtt{ret}\ V_r][\gamma_r]) \in \mathcal{E}^{\sim}[\![\forall^{\nu}AC_o^{\sqsubseteq}]\!]\gamma\delta$$

Expanding definitions and applying anti-reduction, this reduces to showing

$$(w, V_l, V_r') \in \mathcal{V}^{\sim}[\![\forall^{\nu}AC_o^{\sqsubseteq}]\!]\gamma\delta$$

where

$$V_r' = \mathtt{thunk}\ (\Lambda X.\lambda c_X : \mathrm{Case}\ X.[\![\langle BC_o^{\sqsubseteq}\rangle\!\downarrow]\!][(\mathtt{force}\ V_r)\, X\, c_X][\gamma_r])$$

Let $w' \sqsupseteq w$, $R \in Rel[A_l, A_r]$, and $w.\eta \vDash (\sigma_l, \sigma_r, \lfloor R\rfloor_{w.j})$, then we need to show that

$$(w', (\mathtt{force}\ V_l)\, A_l\, \sigma_l, (\mathtt{force}\ V_r')\, A_r\, \sigma_r) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma'\delta'$$

where $\gamma' = (\gamma, c_X \mapsto (\sigma_l, \sigma_r))$ and $\delta' = (\delta, X \mapsto (A_l, A_r, R))$ which reduces in 0 steps to showing

$$(w', (\mathtt{force}\ V_l)\, A_l\, \sigma_l, [\![\langle BC_o^{\sqsubseteq}\rangle\!\downarrow]\!][(\mathtt{force}\ V_r)\, A_r\, \sigma][\gamma_r']) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma'\delta'$$

by noting that by definition, $\gamma_r' = \gamma_r, c_X \mapsto \sigma_r$.
Then, we invoke monadic bind using $(w', (\texttt{force } V_l)\, A_l\, \sigma_l, (\texttt{force } V_r)\, A_r\, \sigma_r) \in \mathcal{E}^{\sim}[\![BC_o^{\sqsubseteq}]\!]\gamma'\delta'$.
Let $w'' \sqsupseteq w'$ and $(w'', V_{lo}, V_{ro}) \in \mathcal{V}^{\sim}[\![BC_o^{\sqsubseteq}]\!]\gamma'\delta'$. We then need to show

$$(w'', \texttt{ret } V_{lo}, [\![\langle BC_o^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_{ro}][\gamma_r'] \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma'\delta')$$

which follows by inductive hypothesis.

(6) If $BC^{\sqsubseteq} = \exists^{\nu}X.BC_o^{\sqsubseteq}$, then by precision inversion also $AC^{\sqsubseteq} = \exists^{\nu}X.AC_o^{\sqsubseteq}$ and $AB^{\sqsubseteq} = \exists^{\nu}X.AB_o^{\sqsubseteq}$. We consider the upcast case, the downcast case follows by an entirely analogous argument. Given $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![\exists^{\nu}X.BC_o^{\sqsubseteq}]\!]\gamma\delta$, we need to show

$$(w, \texttt{ret } V_l, [\![\langle \exists^{\nu}X.BC_o^{\sqsubseteq}\rangle\updownarrow]\!][\texttt{ret } V_r][\gamma_r]) \in \mathcal{E}^{\sim}[\![\exists^{\nu}AC_o^{\sqsubseteq}]\!]\gamma\delta$$

Expanding definitions and applying anti-reduction, this reduces to showing

$$\begin{aligned}(w, \texttt{ret } V_l, \texttt{unpack } (X, y) = \texttt{ret } V_r;& \\ \texttt{ret pack}(X, ()\texttt{thunk } (\lambda c_X : \text{Case } X.[\![\langle BC_o^{\sqsubseteq}\rangle\updownarrow]\!][(\texttt{force } y)\, c_X][\gamma_r]))&) \in \mathcal{E}^{\sim}[\![\exists^{\nu}.AC_o^{\sqsubseteq}]\!]\gamma\delta\end{aligned}$$

By definition of $\mathcal{V}^{\sim}[\![\exists^{\nu}X.BC_o^{\sqsubseteq}]\!]\gamma\delta$, we know

$$V_l = \mathsf{pack}(A_l, V_l')$$

$$V_r = \mathsf{pack}(A_r, V_r')$$

and there is an associated relation $R \in \text{Rel}_{\omega}[A_l, A_r]$. Then the goal reduces to showing

$$(\mathsf{pack}(A_l, V_l'), \mathsf{pack}(A_l, ()\texttt{thunk } (\lambda c_X.[\![\langle BC_o^{\sqsubseteq}\rangle\updownarrow]\!][(\texttt{force } V_r')\, c_X][\gamma_r]))) \in \mathcal{V}^{\sim}[\![\exists^{\nu}X.AC_o^{\sqsubseteq}]\!]\gamma\delta$$

we choose R as the relation for X, and then we need to show (after applying anti-reduction) that for any $w' \sqsupseteq w$, $w' \vDash (\sigma_l, \sigma_r, \lfloor R\rfloor_{w'.j})$ that

$$(w', (\texttt{force } V_l')\, \sigma, [\![\langle BC_o^{\sqsubseteq}\rangle\updownarrow]\!][(\texttt{force } V_r')\, \sigma][\gamma_r']) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma'\delta'$$

where $\gamma' = \gamma, c_X \mapsto (\sigma_l, \sigma_r)$. and $\delta' = \delta, X \mapsto (A_l, A_r, R)$. We use the relatedness assumption and monadic bind again. Then we are given $w'' \sqsupseteq w'$, and $(V_{lo}, V_{ro}) \in \mathcal{V}^{\sim}[\![AB_o^{\sqsubseteq}]\!]\gamma'\delta'$ and need to show

$$(w'', \texttt{ret } V_{lo}, [\![\langle BC_o^{\sqsubseteq}\rangle\updownarrow]\!][\texttt{ret } V_{ro}]) \in \mathcal{E}^{\sim}[\![AC_o^{\sqsubseteq}]\!]\gamma'\delta'$$

which follows by inductive hypothesis.

□

Lemma G.19 (Cast Left). *For any* $\Gamma^{\sqsubseteq} : \Gamma, \Gamma^{\sqsubseteq} \vdash AC^{\sqsubseteq} : A \sqsubseteq C, \Gamma \vdash AB^{\sqsubseteq} : A \sqsubseteq B, \Gamma^{\sqsubseteq} \vdash BC^{\sqsubseteq} : B \sqsubseteq C$ *and* $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma^{\sqsubseteq}]\!]$,

(1) *If* $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![BC^{\sqsubseteq}]\!]\gamma\delta$, *then* $(w, [\![\langle AB^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_l][\gamma_l], \texttt{ret } V_r) \in \mathcal{E}^{\sim}[\![AC^{\sqsubseteq}]\!]\gamma\delta$
(2) *If* $(V_l, V_r) \in \mathcal{V}^{\sim}[\![AC^{\sqsubseteq}]\!]\gamma\delta$, *then* $(w, [\![\langle AB^{\sqsubseteq}\rangle\updownarrow]\!][\texttt{ret } V_l][\gamma_l], \texttt{ret } V_r) \in \mathcal{E}^{\sim}[\![BC^{\sqsubseteq}]\!]\gamma\delta$

Proof. By nested induction on $AB^{\sqsubseteq}$ and $AC^{\sqsubseteq}$, i.e., if $AB^{\sqsubseteq}$ becomes smaller $AC^{\sqsubseteq}$ can be anything but if $AC^{\sqsubseteq}$ becomes smaller, then $AB^{\sqsubseteq}$ must stay the same.

(1) If $AC^{\sqsubseteq} \in \{\mathbb{B}, AC_0^{\sqsubseteq} \times AC_1^{\sqsubseteq}, AC_i^{\sqsubseteq} \to AC_o^{\sqsubseteq}, \forall^{\nu}X.AC_o^{\sqsubseteq}, \exists^{\nu}X.AC_o^{\sqsubseteq}\}$, then $AB^{\sqsubseteq}$ has the same top-level connective, and the proof is symmetric to the case of lemma G.18, which always makes $AB^{\sqsubseteq}$ and $AC^{\sqsubseteq}$ smaller in uses of the inductive hypothesis.
(2) If $AC^{\sqsubseteq} = ?$, then also $AB^{\sqsubseteq} = BC^{\sqsubseteq} = ?$ and the cast is trivial.
(3) If $AC^{\sqsubseteq} = \text{tag}_G(AG^{\sqsubseteq})$, there are two cases: either $BC^{\sqsubseteq} = ?$ or $BC^{\sqsubseteq} = \text{tag}_G(BG^{\sqsubseteq})$.
  (a) If $BC^{\sqsubseteq} = ?$, then $AB^{\sqsubseteq} = \text{tag}_G(AG^{\sqsubseteq}) = AC^{\sqsubseteq}$. Define $\sigma_l = \gamma_l(\text{case}(G))$, $\sigma_r = \gamma_r(\text{case}(G))$.
    (i) In the upcast case, we know $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![\text{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$. In which case, $V_r = \texttt{inj}_{\sigma_r}\, V_r'$

(A) In the $\prec$ case, we know $(w, V_l, V_r') \in \mathcal{V}^{\prec}[\![AG^{\sqsubseteq}]\!]\gamma\delta$, and we need to show

$(w, (x \leftarrow [\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l]; \texttt{ret inj}_{\sigma_l}\ x), \texttt{ret inj}_{\sigma_r}\ V_r') \in \mathcal{E}^{\prec}[\![?]\!]\gamma\delta$

which by forward reduction is equivalent to showing

$$(w, (x \leftarrow [\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l]; \texttt{ret inj}_{\sigma_l}\ x), x \leftarrow \texttt{ret}\ V_r'; \texttt{ret inj}_{\sigma_r}\ x) \in \mathcal{E}^{\prec}[\![?]\!]\gamma\delta$$

By inductive hypothesis, we know

$$(w, [\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l], \texttt{ret}\ V_r') \in \delta_R(\mathrm{case}(G))$$

so can we apply monadic bind. Let $w' \sqsupseteq w$, and $(w', V_l', V_r'') \in \mathcal{V}^{\prec}[\![G]\!]\gamma\delta$. Then we need to show (after applying anti-reduction)

$$(w', \texttt{inj}_{\sigma_l}\ V_l', \texttt{inj}_{\sigma_r}\ V_r'') \in \mathcal{V}^{\prec}[\![?]\!]\gamma\delta$$

To do this, we need to give a relation R such that $w' \vDash (\sigma_l, \sigma_r, R)$ and $(w', V_l', V_r'') \in \triangleright R$. Since $\gamma(\mathrm{case}(G)) = (\sigma_l, \sigma_r)$, we know $R = \lfloor \delta_R(\mathrm{case}(G)) \rfloor_{w'.j}$. And we need to show that for any $w'' \sqsupset w'$, that $(w'', V_l', V_r'') \in \lfloor R \rfloor_{w'.j}$. Which follows by monotonicity because $w'' \sqsupseteq w$.

(B) The $\succ$ case is slightly more complicated. This time we only know we know $(w, V_l, V_r') \in \triangleright\mathcal{V}^{\succ}[\![AG^{\sqsubseteq}]\!]\gamma\delta$ (note the $\triangleright$), and we need to show

$(w, (x \leftarrow [\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l]; \texttt{ret inj}_{\sigma_l}\ x), \texttt{ret inj}_{\sigma_r}\ V_r') \in \mathcal{E}^{\succ}[\![?]\!]\gamma\delta$

By lemma G.17, we know $[\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l]$ either runs to error or terminates. If it runs to an error then our goal holds. Otherwise, let $[\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l] \mapsto^* \texttt{ret}\ V_l'$. Applying anti-reduction, we need to show

$$(w, \texttt{inj}_{\sigma_l}\ V_l', \texttt{inj}_{\sigma_r}\ V_r') \in \mathcal{V}^{\succ}[\![?]\!]\gamma\delta$$

by the same reasoning as above, we need to show

$$(w, V_l', V_r') \in \triangleright\lfloor \delta_R(\mathrm{case}(G)) \rfloor_{w.j}$$

Let $w' \sqsupset w$. We need to show

$$(w', V_l', V_r') \in \lfloor \delta_R(\mathrm{case}(G)) \rfloor_{w.j}$$

By our assumption, we know

$$(w', V_l, V_r') \in \mathcal{V}^{\succ}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

so by inductive hypothesis, we know

$$(w', [\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l], \texttt{ret}\ V_r') \in \mathcal{E}^{\succ}[\![G]\!]\gamma\delta = \delta_R(\mathrm{case}(G))$$

And since we know $w'.\Sigma_l, [\![\langle AG^{\sqsubseteq}\rangle\!\!\upharpoonleft\!\downharpoonright]\!][\texttt{ret}\ V_l][\gamma_l] \mapsto^* w'.\Sigma_l, \texttt{ret}\ V_l'$ by lemma G.17, this means

$$(w', V_l', V_r') \in \mathcal{V}^{\succ}[\![G]\!]\gamma\delta = \delta_R(\mathrm{case}(G))$$

so the result follows by lemma G.14.

(ii) For the downcast case, we know $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![?]\!]\gamma\delta$ which means there exists σ_l, σ_r, R with $w.\eta \vDash (\sigma_l, \sigma_r, R)$ and $V_l = \texttt{inj}_{\sigma_l}\ V_l'$ and $V_r = \texttt{inj}_{\sigma_r}\ V_r'$ and

$$(w, V_l', V_r') \in \triangleright R$$

Expanding the definition of the cast and applying anti-reduction, we need to show

$$(w, \texttt{match}\ V_l\ \texttt{with}\ \sigma_l\{\texttt{inj}\ x.[\![\langle AG^{\sqsubseteq}\rangle\!\!\downarrow]\!][\texttt{ret}\ x] \mid \mho\}, \texttt{ret}\ V_r) \in \mathcal{E}^{\sim}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

If $\gamma_l(\mathrm{case}(G)) = \sigma_l$, the left side errors and the result holds. Otherwise,

$$\texttt{match}\ V_l\ \texttt{with}\ \sigma_l\{\texttt{inj}\ x.[\![\langle AG^{\sqsubseteq}\rangle\!\!\downarrow]\!][\texttt{ret}\ x] \mid \mho\} \mapsto^1 [\![\langle AG^{\sqsubseteq}\rangle\!\!\downarrow]\!]\texttt{ret}\ V_l'$$

(A) In the $\succ$ case, we need to show

$$(w, [\![\langle AG^{\sqsubseteq}\rangle\downarrow]\!]\texttt{ret } V_l', \texttt{ret } V_r) \in \mathcal{E}^{\succ}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

Which follows by inductive hypothesis if we can show

$$(w, V_l', V_r) \in \mathcal{V}^{\succ}[\![\mathrm{tag}_G(G)]\!]\gamma\delta$$

which reduces to showing

$$(w, V_l', V_r') \in \triangleright\mathcal{V}^{\succ}[\![G]\!]\gamma\delta$$

So let $w' \sqsupset w$. We need to show

$$(w', V_l', V_r') \in \mathcal{V}^{\succ}[\![G]\!]\gamma\delta$$

But we know $(w', V_l', V_r') \in R$ where $R = \lfloor \delta_R(\mathrm{case}(G)) \rfloor_{w.j}$ so the result follows by lemma G.15.

(B) In the $\prec$ case, we check $w.j$

- If $w.j = 0$, then the left side takes 1 step so the result holds.
- Otherwise, define $w' = (w.j - 1, w.\Sigma_l, w.\Sigma_r, \lfloor w.\eta \rfloor_{w.j-1})$. Then it is sufficient to show

$$(w', [\![\langle AG^{\sqsubseteq}\rangle\downarrow]\!]\texttt{ret } V_l', \texttt{ret } V_r) \in \mathcal{E}^{\prec}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

To apply the inductive hypothesis we need to show

$$(w', V_l', \texttt{ret } V_r) \in \mathcal{V}^{\prec}[\![\mathrm{tag}_G(G)]\!]\gamma\delta$$

Unrolling definitions, it is sufficient to show

$$(w', V_l', V_r') \in \mathcal{V}^{\prec}[\![G]\!]\gamma\delta$$

But we know already that $(w', V_l', V_r') \in R$ where $R = \lfloor \delta_R(\mathrm{case}(G)) \rfloor_{w.j}$ so the result follows.

(b) Finally, if $BC^{\sqsubseteq} = \mathrm{tag}_G(BG^{\sqsubseteq})$, then $AC^{\sqsubseteq} = \mathrm{tag}_G(AG^{\sqsubseteq})$. We consider the downcast case, the upcast case is entirely symmetric. Let $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![\mathrm{tag}_G(BG^{\sqsubseteq})]\!]\gamma\delta$. Then we know $V_r = \texttt{inj}_{\sigma_r}\ V_r'$ where $\sigma_r = \gamma_r(\mathrm{case}(G))$.

(i) If $\sim = \prec$, we furthermore know $(w, V_l, V_r') \in \mathcal{V}^{\prec}[\![BG^{\sqsubseteq}]\!]\gamma\delta$. We need to show that

$$(w, [\![\langle AB^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_l][\gamma_l], \texttt{ret inj}_{\sigma_r}\ V_r') \in \mathcal{E}^{\prec}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

by forward reduction it is sufficient to show

$$(w, [\![\langle AB^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_l][\gamma_l], x \leftarrow \texttt{ret } V_r'; \texttt{ret inj}_{\sigma_r}\ x) \in \mathcal{E}^{\prec}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

We know by inductive hypothesis that

$$(w, [\![\langle AB^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_l][\gamma_l], \texttt{ret } V_r)$$

so we can apply monadic bind. Let $w' \sqsupseteq w$, and $(w', V_l', V_r'') \in \mathcal{V}^{\prec}[\![AG^{\sqsubseteq}]\!]\gamma\delta$. Then we need to show (after applying anti-reduction) that

$$(w', V_l', \texttt{inj}_{\sigma_r}\ V_r'') \in \mathcal{V}^{\prec}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

Which, unrolling the definition, is

$$(w', V_l', V_r'') \in \mathcal{V}^{\prec}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

which was our assumption.

(ii) If $\sim = \succ$, we only know $(w, V_l, V_r') \in \triangleright\mathcal{V}^{\succ}[\![BG^{\sqsubseteq}]\!]\gamma\delta$ (note the $\triangleright$).

$$(w, [\![\langle AB^{\sqsubseteq}\rangle\downarrow]\!][\texttt{ret } V_l][\gamma_l], \texttt{ret inj}_{\sigma_r}\ V_r') \in \mathcal{E}^{\succ}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

By lemma G.17, the left hand side either errors or terminates with a value.

(A) If $w.\Sigma_l, [\![\langle AB^{\sqsubseteq}\rangle\!\Downarrow]\!][\texttt{ret}\ V_l][\gamma_l] \mapsto^* w.\Sigma_l, \mho$, then the result holds.
(B) If $w.\Sigma_l, [\![\langle AB^{\sqsubseteq}\rangle\!\Downarrow]\!][\texttt{ret}\ V_l][\gamma_l] \mapsto^* w.\Sigma_l, \texttt{ret}\ V_l'$, then we need to show that

$$(w, V_l', \texttt{inj}_{\sigma_r}\ V_r') \in \mathcal{V}^{\succ}[\![\mathrm{tag}_G(AG^{\sqsubseteq})]\!]\gamma\delta$$

Which unrolls to

$$(w, V_l', V_r') \in \triangleright\mathcal{V}^{\succ}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

So let $w' \sqsupset w$. We need to show

$$(w', V_l', V_r') \in \mathcal{V}^{\succ}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

By inductive hypothesis, we know

$$(w', [\![\langle AB^{\sqsubseteq}\rangle\!\Downarrow]\!][\texttt{ret}\ V_l][\gamma_l], \texttt{ret}\ V_r') \in \mathcal{E}^{\succ}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

and so by determinism of evaluation, we know

$$(w', V_l', V_r') \in \mathcal{V}^{\succ}[\![AG^{\sqsubseteq}]\!]\gamma\delta$$

so the result holds.

□

G.2 Compatibility Lemmas

Lemma G.20.

$$\Gamma_1^{\sqsubseteq}, x : A^{\sqsubseteq}, \Gamma_2^{\sqsubseteq} \vDash x \sqsubseteq\sim x \in A^{\sqsubseteq}; \cdot$$

Proof. We need to show

$$(w, \texttt{ret}\ V_l, \texttt{ret}\ V_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$$

where $V_i = \gamma_i(x)$. Since both sides are values, it is sufficient to show

$$(w, \gamma_l(x), \gamma_r(x)) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$$

By definition of $\mathcal{G}^{\sim}[\![\Gamma_1^{\sqsubseteq}, x : A^{\sqsubseteq}, \Gamma_2^{\sqsubseteq}]\!]$, we know $\gamma = \gamma_1, x \mapsto (V_l, V_r), \gamma_2$ and $\delta = \delta_1, \delta_2$ where $(w, \gamma_1, \delta_2) \in \mathcal{G}^{\sim}[\![\Gamma_1^{\sqsubseteq}]\!]$ and $(w, V_l, V_r) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma_1\delta_1$.

Then the result follows because $\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma_1\delta_1 = \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$ by Lemma G.16 □

Lemma G.21. *Generation compatibility.*

$$\frac{\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq\sim M_r \in B^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}, X \cong A^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vDash A^{\sqsubseteq} : A_l \sqsubseteq A_r}{\Gamma^{\sqsubseteq} \vDash \texttt{hide}\ X \cong A_l; M_l \sqsubseteq\sim \texttt{hide}\ X \cong A_r; M_r \in B^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}, \Gamma^{\sqsubseteq\prime\prime}}$$

Proof. The translation is defined as

$$[\![\texttt{hide}\ X \cong A_i; M_i]\!] = \mathsf{newcase}_{[\![A_i]\!]}\ c_X; [\![M_i]\!]$$

We need to show

$$(w, \mathsf{newcase}_{[\![A_l]\!][\delta_l]}\ c_X; [\![M_l]\!][\gamma_l][\delta_l], \mathsf{newcase}_{[\![A_r]\!][\delta_r]}\ c_X; [\![M_r]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

Define $w' = (w.j, (w.\Sigma_l, [\![A_l]\!][\delta]), (w.\Sigma_r, [\![A_r]\!][\delta]), \eta \boxplus (w.\Sigma_l.size, w.\Sigma_r.size, \lfloor\mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta\rfloor_{w}.j))$. Then

$$w.\Sigma_l, \mathsf{newcase}_{[\![A_l]\!][\delta_l]}\ c_X; [\![M_l]\!][\gamma_l][\delta_l] \mapsto^0 w'.\Sigma_l, [\![M_l]\!][\gamma_l][\delta_l][w'.\Sigma_l.size/c_X]$$

and similarly for the right side

$$w.\Sigma_r, \mathsf{newcase}_{[\![A_r]\!][\delta_r]}\ c_X; [\![M_r]\!][\gamma_r][\delta_r] \mapsto^0 w'.\Sigma_r, [\![M_r]\!][\gamma_r][\delta_r][w'.\Sigma_r.size/c_X]$$

Then by the anti-reduction lemma G.9, it is sufficient to show

$$(w', [\![M_l]\!][\gamma_l][\delta_l][w'.\Sigma_l.size/c_X], [\![M_r]\!][\gamma_r][\delta_r][w'.\Sigma_r.size/c_X]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

Define $\gamma' = \gamma, c_X \mapsto (w'.\Sigma_l.size, w'.\Sigma_r.size)$ and $\delta' = \delta, X \mapsto (\delta_l(A_l), \delta_r(A_r), \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta\gamma\delta)$. Then $(w', \gamma', \delta') \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq\prime}, X \cong A, \Gamma^{\sqsubseteq\prime\prime}]\!]$ since $(w', \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq\prime}, \Gamma^{\sqsubseteq\prime\prime}]\!]$ (by monotonicity lemma G.7). Furthermore, $[\![M_i]\!][\gamma_i][\delta_i][w'.\Sigma_i.size/c_X] = [\![M_i]\!][\gamma_i'][\delta_i']$, so the result is equivalent to

$$(w', [\![M_l]\!][\gamma_l'][\delta_l'], [\![M_r]\!][\gamma_r'][\delta_r']) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

which is equivalent to showing

$$(w', [\![M_l]\!][\gamma_l'][\delta_l'], [\![M_r]\!][\gamma_r'][\delta_r']) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta'$$

by lemma G.16. Finally this result holds by applying the hypothesis. □

Lemma G.22 (Compatibility: Upcast Left).

$$\frac{\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vDash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma, \Gamma' \vDash AB^{\sqsubseteq} : A \sqsubseteq B \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vDash BC^{\sqsubseteq} : B \sqsubseteq C}{\Gamma^{\sqsubseteq} \vDash \langle AB^{\sqsubseteq} \rangle\!\upharpoonleft M_l \sqsubseteq M_r \in BC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

Proof. We need to show

$$(w, [\![\langle AB^{\sqsubseteq} \rangle\!\upharpoonleft]\!][[\![M_l]\!]][\gamma_l][\delta_l], [\![M_r]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![BC^{\sqsubseteq}]\!]\gamma\delta$$

By assumption, we know

$$(w, [\![M_l]\!][\gamma_l][\delta_l], [\![M_r]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![AC^{\sqsubseteq}]\!]\gamma\delta$$

So we apply monadic bind. Let $w' \sqsupseteq w$ and $(w', V_l, V_r) \in \mathcal{V}^{\sim}[\![AC^{\sqsubseteq}]\!]\gamma\delta$. We need to show

$$(w', [\![\langle AB^{\sqsubseteq} \rangle\!\upharpoonleft]\!][\mathtt{ret}\ V_l][\gamma_l], \mathtt{ret}\ V_r) \in \mathcal{E}^{\sim}[\![BC^{\sqsubseteq}]\!]\gamma\delta$$

which follows by lemma G.19. □

Lemma G.23 (Compatibility: Downcast Left).

$$\frac{\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in BC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma, \Gamma' \vdash AB^{\sqsubseteq} : A \sqsubseteq B \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash BC^{\sqsubseteq} : B \sqsubseteq C}{\Gamma^{\sqsubseteq} \vDash \langle AB^{\sqsubseteq} \rangle\!\downharpoonleft M_l \sqsubseteq M_r \in AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

Proof. By same argument as lemma G.22. □

Lemma G.24 (Compatibility: Upcast Right).

$$\frac{\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in AB^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma, \Gamma' \vdash BC^{\sqsubseteq} : B \sqsubseteq C \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AB^{\sqsubseteq} : A \sqsubseteq B}{\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq \langle BC^{\sqsubseteq} \rangle\!\upharpoonleft M_r \in AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

Proof. By same argument as lemma G.22, but using lemma G.18. □

Lemma G.25.

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : AC^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AC^{\sqsubseteq} : A \sqsubseteq C \qquad \Gamma, \Gamma' \vdash BC^{\sqsubseteq} : B \sqsubseteq C \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq\prime} \vdash AB^{\sqsubseteq} : A \sqsubseteq B}{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq \langle BC^{\sqsubseteq} \rangle\!\downharpoonleft M_r : AB^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}}$$

Proof. By same argument as lemma G.22, but using lemma G.18. □

Lemma G.26. *If* $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_1^{\sqsubseteq}, X \cong A^{\sqsubseteq}, \Gamma_2^{\sqsubseteq}]\!]$ *of* $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_1^{\sqsubseteq}, X, \Gamma_2^{\sqsubseteq}]\!]$, *then*

$$\mathcal{V}^{\sim}[\![X]\!]\gamma\delta = \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$$

Proof. By definition, $\gamma = \gamma_1, c_X \mapsto (\sigma_l, \sigma_r), \gamma_2$ and $\delta = \delta_1, X \mapsto (A_l, A_r, R), \delta_2$, where $(w, \gamma_1, \delta_1) \in \mathcal{G}^\sim[\![\Gamma_1^\sqsubseteq]\!]$. Then

$$\mathcal{V}^\sim[\![X]\!]\gamma\delta = \delta(X) = \mathcal{V}^\sim[\![A^\sqsubseteq]\!]\gamma_1\delta_1$$

So it is sufficient to show

$$\mathcal{V}^\sim[\![A^\sqsubseteq]\!]\gamma_1\delta_1 = \mathcal{V}^\sim[\![A^\sqsubseteq]\!]\gamma\delta$$

which follows by Lemma G.16. □

Lemma G.27. *Seal*

$$\frac{(X \cong A^\sqsubseteq) \in \Gamma^\sqsubseteq, \Gamma^{\sqsubseteq\prime} \qquad \Gamma^\sqsubseteq \vdash M_l \sqsubseteq M_r : A^\sqsubseteq; \Gamma^{\sqsubseteq\prime}}{\Gamma^\sqsubseteq \vdash \mathtt{seal}_X M_l \sqsubseteq \mathtt{seal}_X M_r : X; \Gamma^{\sqsubseteq\prime}}$$

Proof. Assume $(M_l[\gamma_l][\delta_l], M_r[\gamma_l][\delta_l]) \in \mathcal{E}^\sim[\![A^\sqsubseteq]\!]\gamma\delta$. We need to show

$$(M_l[\gamma_l][\delta_l], M_r[\gamma_l][\delta_l]) \in \mathcal{E}^\sim[\![X]\!]\gamma\delta$$

This follows immediately from Lemma G.26 □

Lemma G.28. *UnSeal*

$$\frac{(X \cong A^\sqsubseteq) \in \Gamma^\sqsubseteq, \Gamma^{\sqsubseteq\prime} \qquad \Gamma^\sqsubseteq \vdash M_l \sqsubseteq M_r : X; \Gamma^{\sqsubseteq\prime}}{\Gamma^\sqsubseteq \vdash \mathtt{unseal}_X M_l \sqsubseteq \mathtt{unseal}_X M_r : A^\sqsubseteq; \Gamma^{\sqsubseteq\prime}}$$

Proof. By same reasoning as the seal case. □

Lemma G.29.

$$\Gamma_i \vdash \mathtt{true} \sqsubseteq\sim \mathtt{true} : \mathbb{B} \qquad\qquad \Gamma_i \vdash \mathtt{false} \sqsubseteq\sim \mathtt{false} : \mathbb{B}$$

Proof. The result $(w, \mathtt{ret\ true}, \mathtt{ret\ true}) \in \mathcal{E}^\sim[\![\mathbb{B}]\!]\gamma\delta$ follows because $(w, \mathtt{true}, \mathtt{true}) \in \mathcal{V}^\sim[\![\mathbb{B}]\!]\gamma\delta$. Similarly for false. □

Lemma G.30.

$$\frac{\Gamma^\sqsubseteq \vDash M_l \sqsubseteq M_r \in \mathbb{B}; \Gamma_0^\sqsubseteq \qquad \Gamma^\sqsubseteq, \Gamma_0^\sqsubseteq \vDash N_{lt} \sqsubseteq N_{rt} \in B^\sqsubseteq; \Gamma^\sqsubseteq \qquad \Gamma^\sqsubseteq, \Gamma_0^\sqsubseteq \vDash N_{lf} \sqsubseteq N_{rf} \in B^\sqsubseteq; \Gamma^\sqsubseteq}{\Gamma^\sqsubseteq \vDash \mathtt{if}\ M_l\ \mathtt{then}\ N_{lt}\ \mathtt{else}\ N_{lf} \sqsubseteq \mathtt{if}\ M_r\ \mathtt{then}\ N_{rt}\ \mathtt{else}\ N_{rf} \in B^\sqsubseteq; \Gamma_0^\sqsubseteq, \Gamma^\sqsubseteq}$$

Proof. Define $M_l' = [\![M_l]\!][\gamma_l][\delta_l]$ and similarly for the rest of the subterms. Then we need to show

$$(w, x \leftarrow M_l'; \mathtt{case}\ x\{x_t.N_{lt}' \mid x_f.N_{lf}'\}, x \leftarrow M_r'; \mathtt{case}\ x\{x_t.N_{rt}' \mid x_f.N_{rf}'\}) \in \mathcal{E}^\sim[\![B^\sqsubseteq]\!]\gamma\delta$$

By assumption and weakening, $(w, M_l', M_r') \in \mathcal{E}^\sim[\![\mathbb{B}]\!]\gamma\delta$, and we apply monadic bind. Suppose $w' \sqsupseteq w$ and $(w', V_l, V_r) \in \mathcal{V}^\sim[\![\mathbb{B}]\!]\gamma\delta$. We need to show

$$(w, x \leftarrow \mathtt{ret}\ V_l; \mathtt{case}\ x\{x_t.N_{lt}' \mid x_f.N_{lf}'\}, x \leftarrow \mathtt{ret}\ V_r; \mathtt{case}\ x\{x_t.N_{rt}' \mid x_f.N_{rf}'\}) \in \mathcal{E}^\sim[\![B^\sqsubseteq]\!]\gamma\delta$$

By definition either $V_l = V_r = \mathtt{true}$ or $V_l = V_r = \mathtt{false}$. WLOG assume it is true. Then

$$x \leftarrow \mathtt{ret}\ V_l; \mathtt{case}\ x\{x_t.N_{lt}' \mid x_f.N_{lf}'\} \mapsto^0 N_{lt}'$$

and similarly for the right side. By anti-reduction (lemma G.9), it is sufficient to show

$$(w', N_{lt}', N_{rt}') \in \mathcal{E}^\sim[\![B^\sqsubseteq]\!]\gamma\delta$$

which follows by hypothesis. □

Lemma G.31. *Product intro*

$$\frac{\Gamma^\sqsubseteq \vDash M_{l0} \sqsubseteq M_{r0} \in A_0^\sqsubseteq; \Gamma_0^\sqsubseteq \qquad \Gamma^\sqsubseteq, \Gamma_0^\sqsubseteq \vDash M_{l1} \sqsubseteq M_{r1} \in A_1^\sqsubseteq; \Gamma_1^\sqsubseteq}{\Gamma^\sqsubseteq \vDash (M_{l0}, M_{l1}) \sqsubseteq (M_{r0}, M_{r1}) \in A_0^\sqsubseteq \times A_1^\sqsubseteq; \Gamma_0^\sqsubseteq, \Gamma_1^\sqsubseteq}$$

Proof. We need to show that

$$\begin{array}{ll}(w, x \leftarrow [\![M_{l0}]\!][\gamma_l][\delta_l]; ,& x \leftarrow [\![M_{r0}]\!][\gamma_r][\delta_r];) \in \mathcal{E}^{\sim}[\![A_0^{\sqsubseteq} \times A_1^{\sqsubseteq}]\!]\gamma\delta \\ \quad y \leftarrow [\![M_{l1}]\!][\gamma_l][\delta_l]; & y \leftarrow [\![M_{r1}]\!][\gamma_r][\delta_r]; \\ \quad \texttt{ret}\ (x, y) & \texttt{ret}\ (x, y)\end{array}$$

By inductive hypothesis and weakening, we know $(w, [\![M_{l0}]\!][\gamma_l][\delta_l], [\![M_{r0}]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A_0^{\sqsubseteq}]\!]\gamma\delta$. Applying monadic bind we get some $w' \sqsupseteq w$ and $(w', V_{l0}, V_{r0}) \in \mathcal{V}^{\sim}[\![A_0^{\sqsubseteq}]\!]\gamma\delta$ and applying anti-reduction, we need to show

$$\begin{array}{ll}(w', y \leftarrow [\![M_{l1}]\!][\gamma_l][\delta_l]; ,& y \leftarrow [\![M_{r1}]\!][\gamma_r][\delta_r];) \in \mathcal{E}^{\sim}[\![A_0^{\sqsubseteq} \times A_1^{\sqsubseteq}]\!]\gamma\delta \\ \quad \texttt{ret}\ (V_{l0}, y) & \texttt{ret}\ (V_{r0}, y)\end{array}$$

By inductive hypothesis, weakening and monotonicity, we know $(w', [\![M_{l1}]\!][\gamma_l][\delta_l], [\![M_{r1}]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A_1^{\sqsubseteq}]\!]\gamma\delta$. Applying monadic bind we get some $w'' \sqsupseteq w'$ and $(w'', V_{l1}, V_{r1}) \in \mathcal{V}^{\sim}[\![A_1^{\sqsubseteq}]\!]\gamma\delta$ and applying anti-reduction we need to show

$$(w'', (V_{l0}, V_{l1}), (V_{r0}, V_{r1})) \in \mathcal{V}^{\sim}[\![A_0^{\sqsubseteq} \times A_1^{\sqsubseteq}]\!]\gamma\delta$$

That is that for each $i \in \{0, 1\}$ that

$$(w'', V_{li}, V_{ri}) \in \mathcal{V}^{\sim}[\![A_i^{\sqsubseteq}]\!]\gamma\delta$$

which follows by assumption and monotonicity. □

Lemma G.32. *Product elim*

$$\frac{\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in A_0^{\sqsubseteq} \times A_1^{\sqsubseteq}; \Gamma_M^{\sqsubseteq} \qquad \Gamma^{\sqsubseteq}, \Gamma_M^{\sqsubseteq}, x : A_0^{\sqsubseteq}, y : A_1^{\sqsubseteq} \vDash N_l \sqsubseteq N_r \in B^{\sqsubseteq}; \Gamma_N^{\sqsubseteq}}{\Gamma^{\sqsubseteq} \vDash \texttt{let}\ (x, y)\ =\ M_l; N_l \sqsubseteq \texttt{let}\ (x, y)\ =\ M_r; N_r \in B^{\sqsubseteq}; \Gamma_M^{\sqsubseteq}, \Gamma_N^{\sqsubseteq}}$$

Proof. We need to show

$$\begin{array}{ll}(w, z \leftarrow [\![M_l]\!][\gamma_l][\delta_l]; ,& z \leftarrow [\![M_r]\!][\gamma_r][\delta_r];) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta \\ \quad \texttt{let}\ (x, y)\ =\ z; & \texttt{let}\ (x, y)\ =\ z; \\ \quad [\![N_l]\!][\gamma_l][\delta_l] & [\![N_r]\!][\gamma_r][\delta_r]\end{array}$$

First, by inductive hypothesis and weakening, we know

$$(w, [\![M_l]\!][\gamma_l][\delta_l], [\![M_r]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A_0^{\sqsubseteq} \times A_1^{\sqsubseteq}]\!]\gamma\delta$$

Applying monadic bind, we get some $w' \sqsupseteq w$ with $(w', V_{l0}, V_{r0}) \in \mathcal{V}^{\sim}[\![A_0^{\sqsubseteq}]\!]\gamma\delta$ and $(w', V_{l1}, V_{r1}) \in \mathcal{V}^{\sim}[\![A_1^{\sqsubseteq}]\!]\gamma\delta$, and applying anti-reduction we need to show

$$(w', [\![N_l]\!][\gamma_l'][\delta_l], [\![N_r]\!][\gamma_r'][\delta_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

Where we define $\gamma' = \gamma, x \mapsto (V_{l0}, V_{r0}), y \mapsto (V_{l1}, V_{r1})$. By weakening, it is sufficient to show

$$(w', [\![N_l]\!][\gamma_l'][\delta_l], [\![N_r]\!][\gamma_r'][\delta_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta$$

which follows by inductive hypothesis if we can show

$$(w', \gamma', \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq}, \Gamma_M^{\sqsubseteq}, \Gamma_N^{\sqsubseteq}, x : A_0^{\sqsubseteq}, y : A_1^{\sqsubseteq}]\!]$$

which follows by definition and monotonicity. □

Lemma G.33. *Fun intro*

$$\frac{\Gamma^{\sqsubseteq}, x : A^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in B^{\sqsubseteq}; \cdot}{\Gamma^{\sqsubseteq} \vDash \lambda x : A_l.M_l \sqsubseteq \lambda x : A_l.M_l \in A^{\sqsubseteq} \to B^{\sqsubseteq}; \cdot}$$

Proof. It is sufficient to show

$$(w, \texttt{thunk } (\lambda x : A_l[\delta_l].M'_l), \texttt{thunk } (\lambda x : A_r[\delta_r].M'_r)) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq} \to B^{\sqsubseteq}]\!]\gamma\delta$$

where $M'_l = [\![M_l]\!][\gamma_l][\delta_l]$ and similarly define M'_r. Suppose $w' \sqsupseteq w$ and $(w', V_l, V_r) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$. We need to show

$$(w', \texttt{force } (\texttt{thunk } (\lambda x : A_l[\delta_l].[\![M_l]\!][\gamma_l][\delta_l]))\, V_l, \texttt{force } (\texttt{thunk } (\lambda x : A_r[\delta_r].[\![M_r]\!][\gamma_r][\delta_r]))\, V_r) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

By anti-reduction it is sufficient to show

$$(w', [\![M_l]\!][\gamma'_l][\delta_l], [\![M_r]\!][\gamma'_r][\delta_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

where $\gamma' = \gamma, x \mapsto (V_l, V_r)$. By monotonicity, we have $(w', \gamma', \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq}]\!]$ so the result follows by hypothesis. □

Lemma G.34.

$$\frac{\Gamma^{\sqsubseteq} \vdash M_l \sqsubseteq M_r : A^{\sqsubseteq} \to B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M \qquad \Gamma^{\sqsubseteq}, \Gamma^{\sqsubseteq}_M \vdash N_l \sqsubseteq N_r : A^{\sqsubseteq}; \Gamma^{\sqsubseteq}_N}{\Gamma^{\sqsubseteq} \vdash M_l\, N_l \sqsubseteq M_r\, N_r : B^{\sqsubseteq}; \Gamma^{\sqsubseteq}_M, \Gamma^{\sqsubseteq}_N}$$

Proof. Define $M'_l = [\![M_l]\!][\gamma_l][\delta_l]$, etc. We need to show

$$(w, f \leftarrow M'_l; x \leftarrow N'_l; \texttt{force } f\, x f \leftarrow M'_r; x \leftarrow N'_r; \texttt{force } f\, x) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

We apply monadic bind with $(w, M'_l, M'_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq} \to B^{\sqsubseteq}]\!]\gamma\delta$ which holds by hypothesis and weakening. Suppose $w' \sqsupseteq w$, with anti-reduction it is sufficient to show

$$(w', x \leftarrow N'_l; \texttt{force } V_l\, x, x \leftarrow N'_r; \texttt{force } V_r\, x) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

where $(w', V_l, V_r) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq} \to B^{\sqsubseteq}]\!]\gamma\delta$. Then we apply monadic bind with $(w', N'_l, N'_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$ which holds by hypothesis, weakening and monotonicity. Suppose $w'' \sqsupseteq w'$; with anti-reduction it is sufficient to show

$$(w'', \texttt{force } V_l\, V'_l, \texttt{force } V_r\, V'_r) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

Which follows by definition of the value relation and transitivity of $\sqsupseteq$. □

Lemma G.35. *Forall intro*

$$\frac{\Gamma^{\sqsubseteq}, X \vDash M_l \sqsubseteq M_r \in A^{\sqsubseteq}; \cdot}{\Gamma^{\sqsubseteq} \vDash \Lambda^{\nu} X.M_l \sqsubseteq \Lambda^{\nu} X.M_r \in \forall^{\nu} X.A^{\sqsubseteq}; \cdot}$$

Proof. Define $M'_i = [\![M_i]\!][\gamma_i][\delta_i]$ It is sufficient to show

$$(w, \texttt{thunk } (\Lambda X.\lambda c_X : \text{Case } X.M'_l), \texttt{thunk } (\Lambda X.\lambda c_X : \text{Case } X.M'_r)) \in \mathcal{V}^{\sim}[\![\forall^{\nu} X.A^{\sqsubseteq}]\!]\gamma\delta$$

Given $w' \sqsupseteq w$, and $R \in \text{Rel}_n[B_l, B_r]$ and $w'.\eta \vDash (\sigma_l, \sigma_r, \lfloor R \rfloor_{w'.j})$, we need to show

$$(w', \texttt{force thunk } (\Lambda X.\lambda c_X : \text{Case } X.M'_l)\, B_l\, \sigma_l \texttt{force thunk } (\Lambda X.\lambda c_X : \text{Case } X.M'_r)\, B_r\, \sigma_r) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma'\delta'$$

where $\gamma' = \gamma, c_X \mapsto (\sigma_l, \sigma_r)$ and $\delta' = \delta, X \mapsto (B_l, B_r, R)$. By anti-reduction it is sufficient to show

$$(w', [\![M_l]\!][\gamma'_l][\delta'_l], [\![M_r]\!][\gamma'_r][\delta'_r]) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma'\delta'$$

which follows by hypothesis if since $(w', \gamma', \delta') \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq}, X]\!]$ by definition and monotonicity. □

Lemma G.36. *Forall elim*

$$\frac{\Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in \forall^{\nu} X.A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime} \qquad \Gamma^{\sqsubseteq} \vDash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vDash M_l\{X \cong B_l\} \sqsubseteq M_r\{X \cong B_r\} \in A^{\sqsubseteq}; \Gamma^{\sqsubseteq\prime}, X \cong B^{\sqsubseteq}}$$

Proof. By definition, instantiation translates as

$$[\![M_i\{X \cong B_i\}]\!] = f \leftarrow [\![M_i]\!]; \texttt{force}\ f\ [\![B_i]\!]\ c_X$$

Define $M_i' = [\![M_i]\!][\gamma_i][\delta_i]$, then we need to show

$$(w, f \leftarrow M_l'; \texttt{force}\ f\ [\![B_l]\!][\delta_l]\ \gamma_l(c_X), f \leftarrow M_r'; \texttt{force}\ f\ [\![B_r]\!][\delta_r]\ \gamma_r(c_X) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$$

We know $(w, M_l', M_r') \in \mathcal{E}^{\sim}[\![\forall^\nu X.A^{\sqsubseteq}]\!]\gamma\delta$ by hypothesis and weakening (lemma G.16), so we can apply monadic bind. Suppose $w' \sqsupseteq w$, then by anti-reduction it is sufficient to show

$$(w', \texttt{force}\ V_l\ [\![B_l]\!][\delta_l]\ \gamma_l(c_X), \texttt{force}\ V_r\ [\![B_r]\!][\delta_r]\ \gamma_r(c_X)) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma\delta$$

where $(w', V_l, V_r) \in \mathcal{V}^{\sim}[\![\forall^\nu X.A^{\sqsubseteq}]\!]\gamma\delta$. Thus it is sufficient to give some relation $R \in \mathrm{Rel}_n[[\![B_l]\!][\delta_l], [\![B_r]\!][\delta_r]]$ such that $w' \vDash (\gamma_l(c_X), \gamma_r(c_X), R)$. By definition of $\mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq\prime}, X \cong B^{\sqsubseteq}]\!]$, this holds for $R = \mathcal{V}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$. □

Lemma G.37.

$$\frac{\Gamma^{\sqsubseteq}, X \cong B^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in A^{\sqsubseteq}; \cdot \qquad \Gamma^{\sqsubseteq} \vdash B^{\sqsubseteq} : B_l \sqsubseteq B_r}{\Gamma^{\sqsubseteq} \vDash \mathsf{pack}^\nu(X \cong B_l, M_l) \sqsubseteq \mathsf{pack}^\nu(X \cong B_r, M_r) \in \exists^\nu X.A^{\sqsubseteq}; \cdot}$$

Proof. Recall the translation is

$$[\![\mathsf{pack}^\nu(X \cong B_i, M_i)]\!] = \texttt{ret}\ (\texttt{pack}([\![B_i]\!], \texttt{thunk}\ \lambda c_X : \mathrm{Case}\ [\![B_i]\!].[\![M_i]\!])\ \texttt{as}\ [\![\exists^\nu X.A_i]\!])$$

where $\Gamma^{\sqsubseteq}, X \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r$. So it is sufficient to show

$$(w, \texttt{pack}([\![B_l]\!][\delta_l], \texttt{thunk}\ \lambda c_X : \mathrm{Case}\ ([\![B_l]\!][\delta_l]).[\![M_l]\!][\gamma_l][\delta_l])\ \texttt{as}\ [\![\exists^\nu X.A^{\sqsubseteq}]\!][\delta_l], \texttt{pack}([\![B_r]\!][\delta_r], \texttt{thunk}\ \lambda c_X : \mathrm{C}$$

For the relation, we pick $R = \mathcal{V}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta$. Let $w' \sqsupseteq w$ and σ_l, σ_r be seals such that $w'.\eta \vDash (\sigma_l, \sigma_r, \lfloor R \rfloor_{w'.j})$. Then we need to show

$$(w, \texttt{force}\ (\texttt{thunk}\ \lambda c_X : \mathrm{Case}\ ([\![B_l]\!][\delta_l]).[\![M_l]\!][\gamma_l][\delta_l])\ \sigma_l, \texttt{force}\ (\texttt{thunk}\ \lambda c_X : \mathrm{Case}\ ([\![B_r]\!][\delta_r]).[\![M_r]\!][\gamma_r][\delta_r])$$

where

$$\begin{aligned}\gamma' &= \gamma, c_X \mapsto (\sigma_l, \sigma_r)\\ \delta' &= \delta, X \mapsto ([\![B_l]\!][\delta_l], [\![B_l]\!][\delta_l], R)\end{aligned}$$

By anti reduction this reduces to showing

$$(w', [\![M_l]\!][\gamma_l'][\delta_l'], [\![M_r]\!][\gamma_r'][\delta_r']) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma'\delta'$$

which follows by assumption. □

Lemma G.38.

$$\frac{\begin{array}{cc}\Gamma^{\sqsubseteq}, \Gamma_M^{\sqsubseteq} \vDash \Gamma_N^{\sqsubseteq} & \Gamma^{\sqsubseteq}, \Gamma_M^{\sqsubseteq}, \Gamma_N^{\sqsubseteq} \vDash B^{\sqsubseteq}\\ \Gamma^{\sqsubseteq} \vDash M_l \sqsubseteq M_r \in \exists^\nu X.A^{\sqsubseteq}; \Gamma_M^{\sqsubseteq} & \Gamma^{\sqsubseteq}, \Gamma_M^{\sqsubseteq}, X, x : A^{\sqsubseteq} \vDash N_l \sqsubseteq N_r \in B^{\sqsubseteq}; \Gamma_N^{\sqsubseteq}\end{array}}{\Gamma^{\sqsubseteq} \vDash \mathsf{unpack}\ (X, x) = M_l; N_l \sqsubseteq \mathsf{unpack}\ (X, x) = M_r; N_r \in B^{\sqsubseteq}; \Gamma_M^{\sqsubseteq}, \Gamma_N^{\sqsubseteq}}$$

Proof. Recall the translation:

$$\begin{aligned}[\![\mathsf{unpack}\ (X, x) = M_i; N_i]\!] = p &\leftarrow [\![M_i]\!];\\ &\texttt{unpack}\ (X, f) = p;\\ &\texttt{newcase}_X\ c_X;\\ &x \leftarrow (\texttt{force}\ f\ c_X);\\ &[\![N_i]\!]\end{aligned}$$

Let $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma^{\sqsubseteq}, \Gamma_M^{\sqsubseteq}, \Gamma_N^{\sqsubseteq}]\!]$. By assumption (and weakening), we know $(w, [\![M_l]\!][\gamma_l][\delta_l], [\![M_r]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![\exists^\nu X.A^{\sqsubseteq}]\!]$. We apply monadic bind. Let $w' \sqsupseteq w$ and $(w', V_{\exists^\nu l}, V_{\exists^\nu r}) \in \mathcal{V}^{\sim}[\![\exists^\nu X.A^{\sqsubseteq}]\!]$.

Then $V_{\exists^\nu l} = \texttt{pack}(A_{Xl}, V_{fl})$ and $V_{\exists^\nu r} = \texttt{pack}(A_{Xr}, V_{fr})$ with some associated relation $R \in \mathrm{Rel}[A_{Xl}, A_{Xr}]$. Then by anti-reduction we need to show

$$\begin{array}{llll}(w', & \texttt{newcase}_{A_{Xl}}\ c_X; & ,\texttt{newcase}_{A_{Xr}}\ c_X; &) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma\delta \\ & x \leftarrow (\texttt{force}\ V_{fl}\ c_X); & x \leftarrow (\texttt{force}\ V_{fr}\ c_X); & \\ & [\![N_l]\!][\gamma_l][\delta_l] & [\![N_r]\!][\gamma_l][\delta_l] & \end{array}$$

Define

$$\begin{aligned} w'' &= (w'.j, (w'.\Sigma_l, A_{Xl}), (w'.\Sigma_r, A_{Xr}), w'.\eta \boxplus (A_{Xl}, A_{Xr}, \lfloor R \rfloor_{w'.j})) \\ \gamma' &= \gamma, c_X \mapsto (w'.\Sigma_l.size, w'.\Sigma_r.size) \\ \delta' &= \delta, X \mapsto (A_{Xl}, A_{Xr}, R) \end{aligned}$$

Then by anti-reduction it is sufficient to show

$$(w'', (x \leftarrow (\texttt{force}\ V_{fl}\ \sigma_l); [\![N_l]\!][\gamma'_l][\delta'_l]), (x \leftarrow (\texttt{force}\ V_{fr}\ \sigma_r); [\![N_r]\!][\gamma'_r][\delta'_r])) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta'$$

By assumption (and monotonicity), $(w'', \texttt{force}\ V_{fl}\ \sigma_l, \texttt{force}\ V_{fr}\ \sigma_r) \in \mathcal{E}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma'\delta'$, so we can apply monadic bind. Let $w''' \sqsupseteq w''$ and $(w''', V_{Al}, V_{Ar}) \in \mathcal{V}^{\sim}[\![A^{\sqsubseteq}]\!]\gamma'\delta'$. Then after anti-reduction we need to show

$$(w''', [\![N_l]\!][\gamma''_l][\delta'_l], [\![N_r]\!][\gamma''_r][\delta'_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma'\delta'$$

where $\gamma'' = \gamma', x \mapsto (V_{Al}, V_{Ar})$. By weakening this is equivalent to

$$(w''', [\![N_l]\!][\gamma''_l][\delta'_l], [\![N_r]\!][\gamma''_r][\delta'_r]) \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma''\delta'$$

which follows by assumption. □

G.3 Proof of Graduality

Lemma G.39. *If $C : (\Gamma \vdash \cdot : A; \Gamma_o) \Rightarrow (\Gamma' \vdash \cdot : B; \Gamma'_o)$ and $\Gamma \vDash M_l \sqsubseteq M_r \in A; \Gamma_o$, then $\Gamma' \vDash C[M_l] \sqsubseteq C[M_r] \in B; \Gamma'_o$.*

Proof. By induction on C, using a corresponding compatibility lemma in each case. □

We define contextual error approximation, following [New and Ahmed 2018]:

Definition G.40. Let $\Gamma \vdash M_1 : A; \Gamma_M$ and $\Gamma \vdash M_2 : A; \Gamma_M$. Then we say M_1 (contextually) *error approximates* M_2, written $\Gamma \vDash M_1 \sqsubseteq^{ctx} M_2 \in A; \Gamma_M$ when for any context $C : (\Gamma \vdash A; \Gamma_M) \Rightarrow (\cdot \vdash B; \cdot)$, all of the following hold:

(1) If $C[M_1] \Uparrow$ then $C[M_2] \Uparrow$
(2) If $C[M_1] \Downarrow V_1$ then there exists V_2 such that $C[M_2] \Downarrow V_2$.

Definition G.41. If $\Gamma \vdash M_1 : A; \Gamma_M$ and $\Gamma \vdash M_2 : A; \Gamma_M$, then M_1 and M_2 are *contextually equivalent*, $\Gamma \vDash M_1 \approx^{\mathrm{ctx}} M_2 \in A; \Gamma_M$, when for any context $C : (\Gamma \vdash A; \Gamma_M) \Rightarrow (\cdot \vdash B; \cdot)$, both diverge $C[M_1], C[M_2] \Uparrow$, or error $C[M_1], C[M_2] \Downarrow \mho$ or terminate successfully $C[M_1] \Downarrow V_1, C[M_2] \Downarrow V_2$.

From syntactic type safety, it is clear that mutual error approximation implies equivalence:

Lemma G.42. *If $\Gamma \vDash M_1 \sqsubseteq^{ctx} M_2 : A; \Gamma_M$ and $\Gamma \vDash M_2 \sqsubseteq^{ctx} M_1 : A; \Gamma_M$, then $\Gamma \vDash M_1 \approx^{ctx} M_2 : A; \Gamma_M$*

Proof. The first two cases are direct. For the third case, we know by type safety that there are only 3 possibilities for a closed term's behavior: $C[M_i] \Uparrow$, $C[M_i] \Downarrow V_i$ or $C[M_i] \Downarrow \mho$. If $C[M_1] \Downarrow \mho$, then it is not the case that $C[M_1] \Uparrow$ or $C[M_1] \Downarrow V_1$ but by the first two cases that means that it is not the case that $C[M_2] \Uparrow$ or $C[M_1] \Downarrow V$, so it must be the case that $C[M_2] \Downarrow \mho$. The opposite direction follows by symmetry. □

To prove the soundness of the logical relation with respect to error approximation, we first need to construct for each step-index n a world $w_p^{\sim}(n)$ to hold the invariants for the cases generated in the preamble of a whole program, a substitution $\delta_p^{\sim}$ to give the relational interpretation of γ_p, and a "binary" version γ_p^2 of the preamble substitution γ_p.

Definition G.43 (Preamble World, Relational Substitution).

$$\begin{aligned}
\eta_p^{\sim}(n) &= \emptyset \boxplus (\mathbb{B}, \mathbb{B}, \mathcal{V}_n^{\sim}[\![\mathbb{B}]\!]\emptyset\emptyset) \boxplus (\mathrm{OSum} \times \mathrm{OSum}, \mathrm{OSum} \times \mathrm{OSum}, \mathcal{V}_n^{\sim}[\![? \times ?]\!]\emptyset\emptyset)\\
&\quad \boxplus (U(\mathrm{OSum} \to F\mathrm{OSum}), U(\mathrm{OSum} \to F\mathrm{OSum}), \lfloor \mathcal{V}_n^{\sim}[\![? \to ?]\!]\emptyset\emptyset \rfloor_{)}\\
&\quad \boxplus (\exists X.U(\mathrm{Case}\ X \to F\mathrm{OSum}), \exists X.U(\mathrm{Case}\ X \to F\mathrm{OSum}), \mathcal{V}_n^{\sim}[\![\exists^{\nu} X.?]\!]\emptyset\emptyset)\\
&\quad \boxplus (U(\forall X.\mathrm{Case}\ X \to F\mathrm{OSum}), \forall X.U(\mathrm{Case}\ X \to F\mathrm{OSum}), \mathcal{V}_n^{\sim}[\![\forall^{\nu} X.?]\!]\emptyset\emptyset)\\
w_p^{\sim}(n) &= (n, \Sigma_p, \Sigma_p, \eta_p^{\sim}(n))\\
\delta_p^{\sim} &= \emptyset, \mathrm{Bool} \mapsto (\mathbb{B}, \mathbb{B}, \mathcal{V}^{\sim}[\![\mathbb{B}]\!]\emptyset\emptyset), \mathrm{Times} \mapsto (\mathrm{OSum} \times \mathrm{OSum}, \mathrm{OSum} \times \mathrm{OSum}, \mathcal{V}^{\sim}[\![? \times ?]\!]\emptyset\emptyset),\\
&\quad \mathrm{Fun} \mapsto (U(\mathrm{OSum} \to F\mathrm{OSum}), U(\mathrm{OSum} \to F\mathrm{OSum}), \mathcal{V}^{\sim}[\![? \to ?]\!]\emptyset\emptyset),\\
&\quad \mathrm{Ex} \mapsto (\exists X.U(\mathrm{Case}\ X \to F\mathrm{OSum}), \exists X.U(\mathrm{Case}\ X \to F\mathrm{OSum}), \mathcal{V}^{\sim}[\![\exists^{\nu} X.?]\!]\emptyset\emptyset),\\
&\quad \mathrm{All} \mapsto (U(\forall X.\mathrm{Case}\ X \to F\mathrm{OSum}), \forall X.U(\mathrm{Case}\ X \to F\mathrm{OSum}), \mathcal{V}^{\sim}[\![\forall^{\nu} X.?]\!]\emptyset\emptyset)\\
\gamma_p^2(x) &= (\gamma_p(x), \gamma_p(x))
\end{aligned}$$

The crucial property is that these together satisfy $\mathcal{G}^{\sim}[\![\Gamma_p]\!]$:

Lemma G.44 (Validity of Preamble World). *For every n, $(w_p^{\sim}(n), \gamma_p^2, \delta_p^{\sim}) \in \mathcal{G}^{\sim}[\![\Gamma_p]\!]$.*

Proof. Clear by definition. □

Lemma G.45.

$$\frac{\Gamma \vDash M_l \sqsubseteq M_2 \in A; \Gamma_M}{\Gamma \vDash M_l \sqsubseteq^{ctx} M_2 \in A; \Gamma_M}$$

Proof. Let C be an appropriately typed closing context. By the congruence Lemma G.39 we know

$$\cdot \vDash C[M_1] \sqsubseteq C[M_2] \in B; \cdot$$

By Lemma G.44, we know that

$$(w_p^{\sim}(n), [\![C[M_1]]\!][\gamma_p], [\![C[M_2]]\!][\gamma_p] \in \mathcal{E}^{\sim}[\![B^{\sqsubseteq}]\!]\gamma_p^2\delta_p^{\sim}$$

(noting that $[\![C[M_i]]\!][\gamma_p][\delta_p^{\sim}] = [\![C[M_i]]\!][\gamma_p]$).

- If $C[M_1] \Uparrow$, then by the simulation Theorem F.16 we know $\Sigma_p, [\![C[M_1]]\!][\gamma_p] \Uparrow$. Then, by adequacy for divergence Corollary F.18, to show $C[M_2] \Uparrow$ it is sufficient to show that $\Sigma_p, [\![C[M_2]]\!][\gamma_p] \Uparrow$. We will show that for every n, $\Sigma_p, [\![C[M_2]]\!][\gamma_p] \mapsto^n \Sigma_n, N_n$ for some Σ_n, N_n. We know

$$(w_p^{\succ}(n), [\![C[M_1]]\!][\gamma_p], [\![C[M_2]]\!][\gamma_p]) \in \mathcal{E}^{\succ}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

  we proceed by the cases of $\mathcal{E}^{\succ}[\![B^{\sqsubseteq}]\!]\gamma\delta$
  - If $w_p^{\succ}(n).\Sigma_r, [\![[C[M_2]][\gamma_p]]\!] \mapsto^{w_p^{\succ}(n).j}$ we are done because $w_p^{\succ}(n).\Sigma_r = \Sigma_p$ and $w_p^{\succ}(n).j = n$.
  - If $w_p^{\succ}(n).\Sigma_l, [\![[C[M_1]][\gamma_p]]\!] \mapsto^* \Sigma', \mho$ we have a contradiction because $\Sigma_p, [\![[C[M_1]][\gamma_p]]\!] \Uparrow$
  - If $w_p^{\succ}(n).\Sigma_l, [\![[C[M_1]][\gamma_p]]\!] \mapsto^* \Sigma', \mathtt{ret}\ V_1$, we also have a contradiction for the same reason.

- If $C[M_1] \Downarrow V_1$ then by simulation we know $\Sigma_p, [\![C[M_1]]\!] \mapsto^n \Sigma', \mathtt{ret}\ V_1'$ for some n, V_1. Furthermore, to show $C[M_2] \Downarrow V_2$ it is sufficient by simulation to show $\Sigma_p, [\![C[M_2]]\!] \mapsto^* \mathtt{ret}\ V_2'$. We know

$$(w_p^{\prec}(n), [\![C[M_1]]\!][\gamma_p], [\![C[M_2]]\!][\gamma_p]) \in \mathcal{E}^{\prec}[\![B^{\sqsubseteq}]\!]\gamma\delta$$

we proceed by the cases of $\mathcal{E}^{\prec}[\![B^{\sqsubseteq}]\!]\gamma\delta$.
  - If $w_p^{\prec}(n).\Sigma_l, [\![C[M_1]]\!][\gamma_p] \mapsto^{w_p^{\prec}(n).j+1}$ or $w_p^{\prec}(n).\Sigma_l, [\![C[M_1]]\!][\gamma_p] \mapsto^j \mho$ this contradicts the fact that $\Sigma_p, [\![C[M_1]]\!] \mapsto^n \Sigma', \mathtt{ret}\ V_1'$.
  - Otherwise, $w_p^{\prec}(n).\Sigma_r, [\![C[M_2]]\!][\gamma_p] \mapsto^* \Sigma', \mathtt{ret}\ V_2'$, so the result holds because $w_p^{\prec}(n).\Sigma_r = \Sigma_p$.

□

Finally we prove Lemma G.39 that states that semantic term precision is a congruence.

Proof. By induction over C. In ever non-cast case we use the corresponding compatibility rule. The two casts cases are precisely dual.

- If $C = \langle A^{\sqsubseteq} \rangle\!\uparrow C'$, where $\Gamma, \Gamma_o' \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r$ then we need to show

$$\Gamma' \vDash \langle A^{\sqsubseteq} \rangle\!\uparrow C'[M_l] \sqsubseteq \langle A^{\sqsubseteq} \rangle\!\uparrow C'[M_r] \in A_r; \Gamma_o'$$

First we use the upcast-left compatibility rule, meaning we need to show

$$\Gamma' \vDash C'[M_l] \sqsubseteq \langle A^{\sqsubseteq} \rangle\!\uparrow C'[M_r] \in A^{\sqsubseteq}; \Gamma_o'$$

Next, we use the upcast-right compatibility rule, meaning we need to show

$$\Gamma' \vDash C'[M_l] \sqsubseteq C'[M_r] \in A_l; \Gamma_o'$$

which follows by inductive hypothesis.
- If $C = \langle A^{\sqsubseteq} \rangle\!\downarrow C'$, where $\Gamma, \Gamma_o' \vdash A^{\sqsubseteq} : A_l \sqsubseteq A_r$ then we need to show

$$\Gamma' \vDash \langle A^{\sqsubseteq} \rangle\!\downarrow C'[M_l] \sqsubseteq \langle A^{\sqsubseteq} \rangle\!\downarrow C'[M_r] \in A_l; \Gamma_o'$$

First we use the downcast-right compatibility rule, meaning we need to show

$$\Gamma' \vDash \langle A^{\sqsubseteq} \rangle\!\downarrow C'[M_l] \sqsubseteq C'[M_r] \in A^{\sqsubseteq}; \Gamma_o'$$

Next, we use the downcast-left compatibility rule, meaning we need to show

$$\Gamma' \vDash C'[M_l] \sqsubseteq C'[M_r] \in A^{\sqsubseteq}; \Gamma_o'$$

which follows by inductive hypothesis.

□

G.4 Free Theorems

Most of the free theorems are stated in terms of contextual equivalence. We define contextual equivalence of PolyG$^{\nu}$ terms to mean contextual equivalence of their elaborations into PolyC$^{\nu}$ terms. By lemma G.42, we can prove a contextual equivalence $M_1 \approx^{\mathrm{ctx}} M_2$ by proving contextual error approximation both ways ($M_1 \sqsubseteq^{ctx} M_2$ and $M_2 \sqsubseteq^{ctx} M_1$). We can prove contextual error approximation by proving logical relatedness by soundness of the logical relation for open terms (Lemma G.45), which is defined in terms of the two logical relations $\sqsubseteq\prec, \sqsubseteq\succ$, giving us technically 4 things to prove: $\sqsubseteq\prec, \sqsubseteq\succ, \succ\sqsupseteq$ and $\prec\sqsupseteq$. However, these cases are all very similar so we show only one case and the other cases follow by essentially symmetric arguments.

Standard Free Theorems.

$$\frac{M : \forall^{\nu} X.X \to X \qquad V_A : A \qquad V_B : B}{\lambda\_ : ?.\texttt{unseal}_X(M\{X \cong A\}(\texttt{seal}_X V_A)) \approx^{ctx} \lambda\_ : ?.\texttt{let}\ y = M\{X \cong B\}V_B; V_A}$$

Theorem G.46.

Proof. There are 4 cases: $\sqsubseteq\prec, \sqsubseteq\succ, \succ\sqsupseteq$ and $\prec\sqsupseteq$ but they are all by a similar argument. Let $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma]\!]$. We need to show

$$(w, \begin{array}{l}\texttt{ret thunk}\ \lambda\_ : ?. \\ \quad \texttt{newcase}_{[\![A]\!][\delta_l]}\ c_X; \\ \quad [\![\texttt{unseal}_X(M\{X \cong A\}(\texttt{seal}_X V_A))]\!][\gamma_l][\delta_l]\end{array}, \begin{array}{l}\texttt{ret thunk}\ \lambda\_ : ?. \\ \texttt{newcase}_{[\![A]\!][\delta_r]}\ c_X; \\ [\![\texttt{let}\ y = M\{X \cong B\}V_B; V_A]\!][\gamma_r][\delta_r]\end{array}) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

Given some $w' \sqsupseteq w$ and (irrelevant) values (w', V_{duml}, V_{dumr}), we need to show (after applying anti-reduction) that

$$(w', \begin{array}{l}\texttt{newcase}_{[\![A]\!][\delta_l]}\ c_X; \\ \quad [\![\texttt{unseal}_X(M\{X \cong A\}(\texttt{seal}_X V_A))]\!][\gamma_l][\delta_l]\end{array}, \begin{array}{l}\texttt{newcase}_{[\![A]\!][\delta_r]}\ c_X; \\ [\![\texttt{let}\ y = M\{X \cong B\}V_B; V_A]\!][\gamma_r][\delta_r]\end{array} \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

Each side allocates a new case and we need to pick a relation with which to instantiate it. By Lemmas F.10 and F.2, we have that $[\![V_A]\!][\gamma_l][\delta_l] \mapsto^0 \texttt{ret}\ V_{Al}$ and $[\![V_B]\!][\gamma_r][\delta_r] \mapsto^0 \texttt{ret}\ V_{Br}$ for some V_{Al} and V_{Br}. We define R to be the "singleton" relation:

$$R = \{(w, V_{Al}, V_{Br}) \in \mathrm{Atom}[[\![A]\!][\delta_l], [\![B]\!][\delta_r]] \mid w \sqsupseteq w'\}$$

Then we define w'' to be the world extended with $\lfloor R \rfloor_{w'.j}$:

$$w'' = (w'.j, w'.\Sigma_l, [\![A]\!][\delta_l], w'.\Sigma_r[\delta_r], w'.\eta \boxplus (w'.\Sigma_l.size, w'.\Sigma_r.size, \lfloor R \rfloor_{w'.j})$$

Then clearly $w'' \sqsupseteq w'$ and

$$\begin{array}{l}w'.\Sigma_l, \texttt{newcase}_{[\![A]\!][\delta_l]}\ c_X; \\ \quad [\![\texttt{unseal}_X(M\{X \cong A\}(\texttt{seal}_X V_A))]\!][\gamma_l][\delta_l]\end{array} \mapsto^0 w''.\Sigma_l, [\![\texttt{unseal}_X(M\{X \cong A\}(\texttt{seal}_X V_A))]\!][\gamma_l'][\delta_l]$$

and similarly for the right hand side:

$$\begin{array}{l}w'.\Sigma_r, \texttt{newcase}_{[\![A]\!][\delta_r]}\ c_X; \\ \quad [\![\texttt{let}\ y = M\{X \cong B\}V_B; V_A]\!][\gamma_r][\delta_r]\end{array} \mapsto^0 w''.\Sigma_r, [\![\texttt{let}\ y = M\{X \cong B\}V_B; V_A]\!][\gamma_r'][\delta_r]$$

where $\gamma' = \gamma, c_X \mapsto (w'.\Sigma_l.size, w'.\Sigma_r.size)$. Expanding definitions (and noting that c_X is free in M), we need to show

$$(w'', S_l[[\![M]\!][\gamma_l'][\delta_l]], S_2[[\![M]\!][\gamma_r'][\delta_r]]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

where

$$\begin{array}{rl}S_l = & f \leftarrow (z \leftarrow \bullet; \texttt{force}\ z([\![A]\!][\delta_l])\ \gamma_l'(c_X)); \\ & x \leftarrow [\![V_A]\!][\gamma_l][\delta_l]; \\ & \texttt{force}\ f\ x\end{array}$$

and

$$S_2 = y \leftarrow \left(\begin{array}{l} \\ f \leftarrow (z \leftarrow \bullet; \texttt{force}\ z([\![B]\!][\delta_l])\ \gamma_r'(c_X)); \\ x \leftarrow [\![V_B]\!][\gamma_r][\delta_r]; \\ \texttt{force}\ f\ x\end{array}\right); [\![V_A]\!][\gamma_r][\delta_r]$$

We apply monadic bind. Let $w''' \sqsupseteq w''$ and $(w''', V_l, V_r) \in \mathcal{V}^{\sim}[\![\forall^\nu X.X \to X]\!]\gamma\delta$. After anti-reduction we need to show

$$(w''', \begin{array}{l} f \leftarrow \texttt{force}\ V_l([\![A]\!][\delta_l])\ \gamma_l'(c_X);, S_r' \\ x \leftarrow [\![V_A]\!][\gamma_l][\delta_l]; \\ \texttt{force}\ f\ x \end{array} \left[\begin{array}{l} f \leftarrow \texttt{force}\ V_r([\![B]\!][\delta_l])\ \gamma_r'(c_X); \\ x \leftarrow [\![V_B]\!][\gamma_r][\delta_r]; \\ \texttt{force}\ f\ x \end{array}\right]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

where

$$S_r' = y \leftarrow \bullet; [\![V_A]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

By weakening it is equivalent to showing the terms are in the relation $\mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$ where $\delta' = ([\![A]\!][\delta_l], [\![B]\!][\delta_r], R)$. By definition of $\mathcal{V}^{\sim}[\![\forall^\nu X.X \to X]\!]$, we have

$$(w''', \texttt{force}\ V_l([\![A]\!][\delta_l])\ \gamma_l'(c_X), \texttt{force}\ V_r([\![B]\!][\delta_r])\ \gamma_r'(c_X)) \in \mathcal{E}^{\sim}[\![X \to X]\!]\gamma'\delta'$$

So we apply monadic bind. Let $w'''' \sqsupseteq w'''$ and let $(w'''', V_{fl}, V_{fr}) \in \mathcal{V}^{\sim}[\![X \to X]\!]\gamma'\delta'$. Applying anti-reduction, we need to show

$$(w'''', \texttt{force}\ V_{fl}\ V_{Al}, y \leftarrow \texttt{force}\ V_{fr}\ V_{Br}; [\![V_A]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$$

By definition of $\mathcal{V}^{\sim}[\![X \to X]\!]$, $(w'''', \texttt{force}\ V_{fl}\ V_{Al}, \texttt{force}\ V_{fr}\ V_{Br}) \in \mathcal{E}^{\sim}[\![X]\!]\gamma'\delta'$ if $(w'''', V_{Al}, V_{Br}) \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$ which holds because $(w'''', V_{Al}, V_{Br}) \in R$. By Lemmas F.10 and F.2, we have that $[\![V_A]\!][\gamma_r][\delta_r] \mapsto^0 \texttt{ret}\ V_{Ar}$ for some V_{Ar}. Applying monadic bind a final time we get $w_{fin} \sqsupseteq w''''$ and $(w_{fin}, V_{Al}', V_{Br}') \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$ and after anti-reduction need to show that

$$(w_{fin}, \texttt{ret}\ V_{Al}', \texttt{ret}\ V_{Ar}) \in \mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$$

By weakening this is equivalent to

$$(w_{fin}, \texttt{ret}\ V_{Al}', \texttt{ret}\ V_{Ar}) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

By the definition of $\mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$, $V_{Al}' = V_{Al}$, so this follows by reflexivity; specifically that $\Gamma \vDash V_A \sqsubseteq V_A \in A; \cdot$. This is because, by the definition of $\mathcal{E}^{\sim}[\![A]\!]\gamma\delta$, since $V_A[\gamma_l][\delta_l] \mapsto^0 \texttt{ret}\ V_{Al}$ and $V_A[\gamma_r][\delta_r] \mapsto^0 \texttt{ret}\ V_{Ar}$, those values are related. □

Theorem G.47.

$$\frac{M : \forall^\nu X.\forall^\nu Y.(X \times Y) \to (Y \times X) \qquad V_A : A \qquad V_B : B}{\begin{array}{c}\lambda\_ : ?.\texttt{let}\ (y,x)\ =\ (M\{X \cong A\}\{Y \cong B\}\ (\texttt{seal}_X V_A, \texttt{seal}_Y V_B)); (\texttt{unseal}_X x, \texttt{unseal}_Y y) \\ \approx^{ctx} \lambda\_ : ?.\texttt{let}\ (y,x)\ =\ (M\{X \cong B\}\{Y \cong A\}\ (\texttt{seal}_X V_B, \texttt{seal}_Y V_A)); (\texttt{unseal}_Y y, \texttt{unseal}_X x)\end{array}}$$

Proof. We show the $\sqsubseteq\sim$ case, the $\sim\sqsupseteq$ case is symmetric. Let $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p, \Gamma]\!]$. Define the following terms

$$N_l = [\![\texttt{let}\ (y,x)\ =\ (M\{X \cong A\}\{Y \cong B\}\ (\texttt{seal}_X V_A, \texttt{seal}_Y V_B)); (\texttt{unseal}_X x, \texttt{unseal}_Y y)]\!]$$
$$N_r = [\![\texttt{let}\ (y,x)\ =\ (M\{X \cong B\}\{Y \cong A\}\ (\texttt{seal}_X V_B, \texttt{seal}_Y V_A)); (\texttt{unseal}_Y y, \texttt{unseal}_X x)]\!]$$

Then we need to show for any $w_1 \sqsupseteq w$ that

$$(w_1, \texttt{newcase}_A\ c_X; \texttt{newcase}_B\ c_Y; N_l[\gamma_l][\delta_l], \texttt{newcase}_B\ c_X; \texttt{newcase}_A\ c_Y; N_r[\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A{\times}B]\!]\gamma\delta$$

By Lemmas F.10 and F.2, we have that $[\![V_A]\!][\gamma_i][\delta_i] \mapsto^0 \texttt{ret}\ V_{Ai}$ and $[\![V_B]\!][\gamma_i][\delta_i] \mapsto^0 \texttt{ret}\ V_{Bi}$ for some V_{Ai} and V_{Bi}. Define

$$\begin{aligned}
A_i &= [\![A]\!][\delta_i] \\
B_i &= [\![B]\!][\delta_i] \\
R_X &= \{(w, V_{Al}, V_{Br}) \in \mathrm{Atom}[A_l, B_r] \mid w \sqsupseteq w_1\} \\
R_Y &= \{(w, V_{Bl}, V_{Ar}) \in \mathrm{Atom}[B_l, A_r] \mid w \sqsupseteq w_1\} \\
w_2 &= (w_1.j, (w_1.\Sigma_l, A_l, B_l), (w_1.\Sigma_r, B_r, A_r), (w_1.\eta \boxplus (A_l, B_r, \lfloor R_X \rfloor_{w_1.j}) \boxplus (B_l, A_r, \lfloor R_Y \rfloor_{w_1.j}))) \\
\sigma_{Xi} &= w_1.\Sigma_i.size \\
\sigma_{Yi} &= w_1.\Sigma_i.size + 1 \\
\gamma' &= \gamma, c_X \mapsto (\sigma_{Xl}, \sigma_{Xr}), c_Y \mapsto (\sigma_{Yl}, \sigma_{Yr}) \\
\delta' &= \delta, X \mapsto (A_l, B_r, R_X), Y \mapsto (B_l, A_r, R_Y)
\end{aligned}$$

Then it is sufficient to show that

$$(w_2, N_l[\gamma_l'][\delta_l'], N_r[\gamma_r'][\delta_r']) \in \mathcal{E}^{\sim}[\![A \times B]\!]\gamma\delta$$

which is equivalent by weakening to

$$(w_2, N_l[\gamma_l'][\delta_l'], N_r[\gamma_r'][\delta_r']) \in \mathcal{E}^{\sim}[\![A \times B]\!]\gamma'\delta'$$

Next, note that by reflexivity,

$$(w_2, [\![M]\!][\gamma_l'][\delta_l'], [\![M]\!][\gamma_r'][\delta_r']) \in \mathcal{E}^{\sim}[\![\forall^\nu X.\forall^\nu Y.(X \times Y) \to (Y \times X)]\!]\gamma'\delta'$$

and

$$N_i = [\![S_i]\!][[\![M]\!]][\gamma_i'][\delta_i']$$

for stacks

$$\begin{aligned}
S_l &= \texttt{let}\ (y, x)\ =\ (\bullet\{X \cong A\}\{Y \cong B\}\ (\texttt{seal}_X V_A, \texttt{seal}_Y V_B)); (\texttt{unseal}_X x, \texttt{unseal}_Y y) \\
S_r &= \texttt{let}\ (y, x)\ =\ (\bullet\{X \cong B\}\{Y \cong A\}\ (\texttt{seal}_X V_B, \texttt{seal}_Y V_A)); (\texttt{unseal}_Y y, \texttt{unseal}_X x)
\end{aligned}$$

So we can apply monadic bind. Let $w_3 \sqsupseteq w_2$ and $(w_3, V_{\forall^\nu l}, V_{\forall^\nu r}) \in \mathcal{V}^{\sim}[\![\forall^\nu X.\forall^\nu Y.(X \times Y) \to (Y \times X)]\!]\gamma'\delta'$. Then by anti-reduction it is sufficient to show

$$(w_3, [\![S_l']\!][\texttt{force}\ V_{\forall^\nu l}\, A_l\, \sigma_{Xl}], [\![S_r']\!][\texttt{force}\ V_{\forall^\nu r}\, B_r\, \sigma_{Xr}]) \in \mathcal{E}^{\sim}[\![A \times B]\!]\gamma'\delta'$$

for stacks

$$\begin{aligned}
S_l' &= \texttt{let}\ (y, x)\ =\ (\bullet\{Y \cong B\}\ (\texttt{seal}_X V_A, \texttt{seal}_Y V_B)); (\texttt{unseal}_X x, \texttt{unseal}_Y y) \\
S_r' &= \texttt{let}\ (y, x)\ =\ (\bullet\{Y \cong A\}\ (\texttt{seal}_X V_B, \texttt{seal}_Y V_A)); (\texttt{unseal}_Y y, \texttt{unseal}_X x)
\end{aligned}$$

By definition, we know $(w_3, \texttt{force}\ V_{\forall^\nu l}\, A_l\, \sigma_{Xl}, \texttt{force}\ V_{\forall^\nu r}\, B_r\, \sigma_{Xr}) \in \mathcal{E}^{\sim}[\![\forall^\nu Y.(X \times Y) \to (Y \times X)]\!]\gamma'\delta'$, so we can apply monadic bind again. Let $w_4 \sqsupseteq w_3$ and $(w_4, V_{\forall^\nu l}', V_{\forall^\nu r}') \in \mathcal{V}^{\sim}[\![\forall^\nu Y.(X \times Y) \to (Y \times X)]\!]\gamma'\delta'$. Then after anti-reduction we need to show

$$(w_4, [\![S_l'']\!][\texttt{force}\ V_{\forall^\nu l}'\, B_l\, \sigma_{Yl}], [\![S_r'']\!][\texttt{force}\ V_{\forall^\nu r}'\, A_r\, \sigma_{Yr}]) \in \mathcal{E}^{\sim}[\![A \times B]\!]\gamma'\delta'$$

for stacks

$$\begin{aligned}
S_l'' &= \texttt{let}\ (y, x)\ =\ (\bullet\ (\texttt{seal}_X V_A, \texttt{seal}_Y V_B)); (\texttt{unseal}_X x, \texttt{unseal}_Y y) \\
S_r'' &= \texttt{let}\ (y, x)\ =\ (\bullet\ (\texttt{seal}_X V_B, \texttt{seal}_Y V_A)); (\texttt{unseal}_Y y, \texttt{unseal}_X x)
\end{aligned}$$

Similarly to before, we know $(w_4, \texttt{force}\ V'_{\forall^\nu l}\ B_l\ \sigma_{Yl}, \texttt{force}\ V'_{\forall^\nu r}\ A_r\ \sigma_{Yr}) \in \mathcal{E}^\sim[\![(X \times Y) \to (Y \times X)]\!]\gamma'\delta'$ so we can apply monadic bind again. Let $w_5 \sqsupseteq w_r$ and $(w_5, V_{fl}, V_{fr}) \in \mathcal{V}^\sim[\![(X \times Y) \to (Y \times X)]\!]\gamma'\delta'$. Next, note that

$$[\![(\texttt{seal}_X V_A, \texttt{seal}_Y V_B)]\!][\gamma'_l][\delta'_l] \mapsto^0 \texttt{ret}\ (V_{Al}, V_{Bl})$$
$$[\![(\texttt{seal}_X V_B, \texttt{seal}_Y V_A)]\!][\gamma'_r][\delta'_r] \mapsto^0 \texttt{ret}\ (V_{Br}, V_{Ar})$$

So by anti-reduction we need to show

$$(w_5, [\![S'''_l]\!][\texttt{force}\ V'_{\forall^\nu l}\ (V_{Al}, V_{Bl})], [\![S'''_r]\!][\texttt{force}\ V'_{\forall^\nu r}\ (V_{Br}, V_{Ar})]) \in \mathcal{E}^\sim[\![A \times B]\!]\gamma'\delta'$$

where

$$S'''_l = \texttt{let}\ (y, x)\ =\ \bullet; (\texttt{unseal}_X x, \texttt{unseal}_Y y)$$
$$S'''_r = \texttt{let}\ (y, x)\ =\ \bullet; (\texttt{unseal}_Y y, \texttt{unseal}_X x)$$

To show that

$$(w_5, \texttt{force}\ V'_{\forall^\nu l}\ (V_{Al}, V_{Bl}), \texttt{force}\ V'_{\forall^\nu r}\ (V_{Br}, V_{Ar})) \in \mathcal{E}^\sim[\![Y \times X]\!]\gamma'\delta'$$

It is sufficient to show

$$(w_5, (V_{Al}, V_{Bl}), (V_{Br}, V_{Ar})) \in \mathcal{E}^\sim[\![X \times Y]\!]\gamma'\delta'$$

which follows because $(w_5, V_{Al}, V_{Br}) \in R_X$ and $(w_5, V_{Br}, V_{Ar}) \in R_Y$. Then we can apply monadic bind a final time. Let $w_{fin} \sqsupseteq w_5$ and $(w_{fin}, V_{ol}, V_{or}) \in \mathcal{V}^\sim[\![Y \times X]\!]\gamma'\delta'$. Then we need to show

$$(w_{fin}, [\![S'''_l]\!][\texttt{ret}\ V_{ol}], [\![S'''_r]\!][\texttt{ret}\ V_{or}]) \in \mathcal{E}^\sim[\![A \times B]\!]\gamma'\delta'$$

First, by definition, it must be the case that $V_{ol} = (V_{Bl}, V_{Al})$ and $V_{or} = (V_{Ar}, V_{Br})$. Then by anti-reduction we need to show

$$(w_{fin}, \texttt{ret}\ (V_{Al}, V_{Bl}), \texttt{ret}\ (V_{Ar}, V_{Br})) \in \mathcal{E}^\sim[\![A \times B]\!]\gamma'\delta'$$

which means we need to show

$$(w_{fin}, V_{Al}, V_{Ar}) \in \mathcal{V}^\sim[\![A]\!]\gamma'\delta'$$

and

$$(w_{fin}, V_{Bl}, V_{Br}) \in \mathcal{V}^\sim[\![B]\!]\gamma'\delta'$$

which follows by reflexivity and monotonicity. □

Lemma G.48.

$$\begin{aligned}&\mathsf{pack}^\nu(X \cong \mathbb{B}, (\mathsf{seal}_X\mathsf{true}, (NOT, \lambda x : X.\mathsf{unseal}_X x)))\\ \approx^{ctx}\ &\mathsf{pack}^\nu(X \cong \mathbb{B}, (\mathsf{seal}_X\mathsf{false}, (NOT, \lambda x : X.NOT(\mathsf{unseal}_X x))))\end{aligned}$$

Proof. We do the $\sqsubseteq\sim$ case, the $\sim\sqsupseteq$ case is symmetric. Let $(w, \gamma, \delta) \in \mathcal{G}^\sim[\![\Gamma_p]\!]$. The goal reduces to showing

$$\begin{pmatrix} w, & \mathsf{pack}(\mathbb{B}, (\mathsf{thunk}\ \lambda c_X : \mathrm{Case}\ X.[\![(\mathsf{seal}_X\mathsf{true}, (\mathrm{NOT}, (\lambda x : X.\mathsf{unseal}_X x)))]\!])), \\ & \mathsf{pack}(\mathbb{B}, \mathsf{thunk}\ \lambda c_X : \mathrm{Case}\ X.[\![(\mathsf{seal}_X\mathsf{false}, (\mathrm{NOT}, \lambda x : X.\mathrm{NOT}\,(\mathsf{unseal}_X x)))]\!]) \end{pmatrix}$$
$$\in \mathcal{V}^\sim[\![\exists^\nu X.X \times ((X \to X) \times (X \to \mathbb{B}))]\!]\gamma\delta$$

The relation we pick is $R = \{(w', \texttt{true}, \texttt{false}) \in \text{Atom}[\mathbb{B}, \mathbb{B}]\} \cup \{(w', \texttt{false}, \texttt{true}) \in \text{Atom}[\mathbb{B}, \mathbb{B}]\}$ Then we need to show for any future $w' \sqsupseteq w$ and $w' \vDash (\sigma_l, \sigma_r, \lfloor R \rfloor_{w'.j})$, that

$$\left(w', \begin{array}{l}(\texttt{force}\ (\texttt{thunk}\ \lambda c_X : \text{Case } X.[\![(\texttt{seal}_X \texttt{true}, (\text{NOT}, \lambda x : X.\texttt{unseal}_X x))]\!])\ \sigma_l),\\ (\texttt{force}\ \texttt{thunk}\ \lambda c_X : \text{Case } X.[\![(\texttt{seal}_X \texttt{true}, (\text{NOT}, \lambda x : X.\text{NOT}\ (\texttt{unseal}_X x)))]\!]\ \sigma_r)\end{array}\right)$$
$$\in \mathcal{E}^{\sim}[\![X \times ((X \to X) \times (X \to \mathbb{B}))]\!]\gamma'\delta'$$

where

$$\gamma' = \gamma, c_X \mapsto (\sigma_l, \sigma_r)$$
$$\delta' = \delta, X \mapsto (\mathbb{B}, \mathbb{B}, R)$$

And after applying anti-reduction, by monadic bind, we need to show the following 3 things:

$$(w', \texttt{true}, \texttt{false}) \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$$
$$(w', [\![\text{NOT}]\!][\gamma'_l][\delta'_l], [\![\text{NOT}[\gamma'_r][\delta'_r]]\!]) \in \mathcal{E}^{\sim}[\![X \to X]\!]\gamma'\delta'$$
$$(w', [\![\lambda x : X.\texttt{unseal}_X x]\!], [\![\lambda x : X.\text{NOT}\ (\texttt{unseal}_X x)]\!]) \in \mathcal{E}^{\sim}[\![X \to \mathbb{B}]\!]\gamma'\delta'$$

(1) First, $(w', \texttt{true}, \texttt{false}) \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$ follows directly from the definition of $\delta_R(X) = R$.
(2) Second, let $w'' \sqsupseteq w'$ and $(w'', V_l, V_r) \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$. Then we need to show

$$(w'', \texttt{force}\ V_{NOT}[\gamma'_l][\delta'_l]\ V_l, \texttt{force}\ V_{NOT}[\gamma'_r][\delta'_r]\ V_r) \in \mathcal{E}^{\sim}[\![X]\!]\gamma'\delta'$$

where $[\![\text{NOT}]\!] = \texttt{ret}\ V_{NOT}$. There are two cases: either $V_l = \texttt{true}$ and $V_r = \texttt{false}$ or vice-versa. In either case, NOT swaps the two values and the result holds.
(3) Finally, let $w'' \sqsupseteq w'$ and $(w'', V_l, V_r) \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$. Then we need to show,

$$(w'', \texttt{force}\ V_{fl}\ V_l, \texttt{force}\ V_{fr}\ V_r) \in \mathcal{E}^{\sim}[\![\mathbb{B}]\!]\gamma'\delta'$$

where $[\![\lambda x : X.\texttt{unseal}_X x]\!] = \texttt{ret}\ V_{fl}$ and $[\![\lambda x : X.\text{NOT}\ (\texttt{unseal}_X x)]\!] = \texttt{ret}\ V_{fr}$. By definition of R, either $V_l = \texttt{true}$ and $V_r = \texttt{false}$ or vice-versa. In either case, both sides evaluate to $\texttt{ret}\ V_l$, and we need to show

$$(w'', V_l, V_l) \in \mathcal{V}^{\sim}[\![\mathbb{B}]\!]\gamma'\delta'$$

which follows by definition.

□

Free Theorems with the Dynamic Type.

Theorem G.49. *If $M : \forall^\nu X.? \to X$ and $\cdot \vdash A$ and $\cdot \vdash V : ?$, then either*

$$((\lambda\_ : ?.\texttt{unseal}_X (M\{X \cong A\}\ V))\ \texttt{true})^+ \Uparrow$$

or

$$((\lambda\_ : ?.\texttt{unseal}_X (M\{X \cong A\}\ V))\ \texttt{true})^+ \mapsto^* \mho$$

Proof. Define $N = (\lambda\_ : ?.\texttt{unseal}_X (M\{X \cong A\}\ V))\ \texttt{true}$ By the adequacy lemmas ??,??, it is sufficient to show that for every $n \in \mathbb{N}$, either

$$\Sigma_p \triangleright [\![(N)^+]\!][\gamma_p] \mapsto^n$$

or

$$\Sigma_p \triangleright [\![(N)^+]\!][\gamma_p] \mapsto^{\leq n} \mho$$

Unraveling definitions, we get

$$\begin{aligned}\Sigma_p; [\![((\lambda\_ : ?.\texttt{unseal}_X (M\{X \cong A\}\ V))\ \texttt{true})^+]\!] &\mapsto^* \Sigma_p; \texttt{newcase}_{[\![A]\!]}\ c_X; [\![((M\{X \cong A\}\ V))^+]\!][\gamma_p]\\ &\mapsto \Sigma; [\![((M\{X \cong A\}\ V))^+]\!][\gamma_p, c_X \mapsto \sigma]\end{aligned}$$

where $\sigma = \Sigma_p.size$, $\Sigma = \Sigma_p, \sigma$.

Next, define $\gamma = \gamma_p^2, c_X \mapsto (\sigma, \sigma)$, $w = (w^{\prec}(n).j, \Sigma, \Sigma, w.\eta \boxplus (\llbracket A \rrbracket, \llbracket A \rrbracket, \emptyset))$ and $\delta = \delta_p^{\prec}, X \mapsto (\llbracket A \rrbracket, \llbracket A \rrbracket, \emptyset)$, noting that we use the empty relation $\emptyset$ as the interpretation of X.

To prove the claim, it is sufficient to show that

$$(w, \llbracket ((M\{X \cong A\}\, V))^+ \rrbracket [\gamma_l], \llbracket ((M\{X \cong A\}\, V))^+ \rrbracket [\gamma_r]) \in \mathcal{E}^{\prec}\llbracket X \rrbracket \gamma\delta$$

since $\mathcal{V}^{\prec}\llbracket X \rrbracket \gamma\delta = \emptyset$ and therefore the case where the two sides reduce to values is impossible.

First, since $(w, \gamma_p^2, \delta_p^{\prec}) \in \mathcal{G}^{\prec}\llbracket \Gamma_p \rrbracket$, by reflexivity and weakening we know $(w, \llbracket (M)^+ \rrbracket [\gamma_l], \llbracket (M)^+ \rrbracket [\gamma_r]) \in \mathcal{E}^{\prec}\llbracket \forall^{\nu} X.? \to X \rrbracket \gamma\delta$.

We apply monadic bind. Assume $w' \sqsupseteq w$ and $(w', V_{\forall,l}, V_{\forall,r}) \in \mathcal{V}^{\prec}\llbracket \forall^{\nu} X.? \to X \rrbracket \gamma\delta$, then by anti-reduction we need to show that

$$(w', \llbracket S \rrbracket [\mathtt{force}\ V_{\forall,l}\ [\llbracket A \rrbracket]\ \sigma] [\gamma_l][\delta_l], \llbracket S \rrbracket [\mathtt{force}\ V_{\forall,r}\ [\llbracket A \rrbracket]\ \sigma] [\gamma_r][\delta_r]) \in \mathcal{E}^{\prec}\llbracket X \rrbracket \gamma\delta$$

where

$$S = \bullet\, V$$

Next, since $w'.\eta \vDash (\sigma, \sigma, \emptyset)$ and $\delta(X) = (\llbracket A \rrbracket, \llbracket A \rrbracket, \emptyset)$, we have $(w', \mathtt{force}\ V_{\forall,l}\ [\llbracket A \rrbracket]\ \sigma, \mathtt{force}\ V_{\forall,l}\ [\llbracket A \rrbracket]\ \sigma) \in \mathcal{E}^{\prec}\llbracket ? \to X \rrbracket \gamma\delta$ Again we apply monadic bind. Assume $w'' \sqsupseteq w'$ and $(w'', V_{f,l}, V_{f,r}) \in \mathcal{V}^{\prec}\llbracket ? \to X \rrbracket \gamma\delta$. Then by Lemma F.10, we know $\Sigma \triangleright \llbracket V^+ \rrbracket [\gamma_i] \mapsto^* \Sigma \triangleright \mathtt{ret}\ V_x[\gamma_i]$, so applying anti-reduction it suffices to show

$$(w'', \mathtt{force}\ V_{f,l}\, V_x[\gamma_l], \mathtt{force}\ V_{f,r}\, V_x[\gamma_r]) \in \mathcal{E}^{\prec}\llbracket X \rrbracket \gamma\delta$$

for which it suffices to show that

$$(w'', V_x[\gamma_l], V_x[\gamma_r]) \in \mathcal{V}^{\prec}\llbracket ? \rrbracket \gamma\delta$$

for which it suffices to show that

$$(w'', \llbracket V^+ \rrbracket [\gamma_l], \llbracket V^+ \rrbracket [\gamma_r]) \in \mathcal{E}^{\prec}\llbracket ? \rrbracket \gamma\delta$$

which by weakening is equivalent to showing

$$(w'', \llbracket V^+ \rrbracket [\gamma_p], \llbracket V^+ \rrbracket [\gamma_p]) \in \mathcal{E}^{\prec}\llbracket ? \rrbracket \gamma_p^2 \delta_p^{\prec}$$

which follows by reflexivity and the fact that $(w'', \gamma_p^2, \delta_p^{\prec}) \in \mathcal{G}^{\prec}\llbracket \Gamma_p \rrbracket$. □

Theorem G.50. *For any* $\cdot \vdash A, B$ *and* $V_A : A$ *and* $V_B : B$,

$$\lambda\_ : ?.\mathtt{unseal}_X(M\{X \cong A\}\, \mathtt{seal}_X V_A) \approx^{ctx}$$

$$\lambda\_ : ?.\mathtt{let}\ y = (\mathtt{unseal}_X(M\{X \cong B\}\, \mathtt{seal}_X V_B)); V_A : ? \to A$$

Proof. We show one direction of the 4 cases, the others are analogous. Let $(w, \gamma, \delta) \in \mathcal{G}^{\sim}\llbracket \Gamma_p \rrbracket$. We need to show

$$\begin{array}{l} (w, \mathtt{ret\ thunk}\ \lambda\_ : ?. \\ \quad \mathtt{newcase}_{\llbracket A \rrbracket}\ c_X; \\ \quad \llbracket M\{X \cong A\}\, (\langle ? \rangle\!\downarrow \langle \mathrm{tag}_X(X) \rangle\!\uparrow (\mathtt{seal}_X V_A)) \rrbracket [\gamma_l][\delta_l] \\ \quad , \\ \quad \mathtt{ret\ thunk}\ \lambda\_ : ?. \\ \quad \mathtt{newcase}_{\llbracket A \rrbracket}\ c_X; \\ \quad \llbracket \mathtt{let}\ y = M\{X \cong B\}\, (\langle ? \rangle\!\downarrow \langle \mathrm{tag}_X(X) \rangle\!\uparrow (\mathtt{seal}_X V_B)); V_A \rrbracket [\gamma_r][\delta_r] \\ ) \in \mathcal{E}^{\sim}\llbracket A \rrbracket \gamma\delta \end{array}$$

Given some $w' \sqsupseteq w$ and (irrelevant) values (w', V_{duml}, V_{dumr}), we need to show (after applying anti-reduction) that

$$\begin{array}{l}(w', \mathtt{newcase}_{[\![A]\!]}\ c_X; \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad , \qquad\qquad\qquad) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta \\ \quad [\![M\{X \cong A\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_X(X)\rangle\!\uparrow\ (\mathtt{seal}_X V_A))]\!][\gamma_l][\delta_l] \\ \\ \quad \mathtt{newcase}_{[\![A]\!]}\ c_X; \\ \quad [\![\mathtt{let}\ y = M\{X \cong B\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_X(X)\rangle\!\uparrow\ (\mathtt{seal}_X V_B)); V_A]\!][\gamma_r][\delta_r]\end{array}$$

Each side allocates a new case and we need to pick a relation with which to instantiate it. By Lemmas F.10 and F.2, we have that $[\![V_A]\!][\gamma_l][\delta_l] \mapsto^0 \mathtt{ret}\ V_{Al}$ and $[\![V_B]\!][\gamma_r][\delta_r] \mapsto^0 \mathtt{ret}\ V_{Br}$ for some V_{Al} and V_{Br}. As in the ordinary identity function proof, we define R to be the "singleton" relation:

$$R = \{(w, V_{Al}, V_{Br}) \in \mathrm{Atom}[[\![A]\!][\delta_l], [\![B]\!][\delta_r]] \mid w \sqsupseteq w'\}$$

Then we define w'' to be the world extended with $\lfloor R \rfloor_{w'.j}$:

$$w'' = (w'.j, w'.\Sigma_l, [\![A]\!][\delta_l], w'.\Sigma_r[\delta_r], w'.\eta \boxplus (w'.\Sigma_l.size, w'.\Sigma_r.size, \lfloor R \rfloor_{w'.j})$$

Then clearly $w'' \sqsupseteq w'$ and

$$\begin{array}{l} w'.\Sigma_l, \mathtt{newcase}_{[\![A]\!][\delta_l]}\ c_X; \\ \qquad [\![M\{X \cong A\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_X(X)\rangle\!\uparrow\ (\mathtt{seal}_X V_A))]\!][\gamma_l][\delta_l] \\ \\ \mapsto^0 w''.\Sigma_l, [\![M\{X \cong A\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_X(X)\rangle\!\uparrow\ (\mathtt{seal}_X V_A))]\!][\gamma_l'][\delta_l]\end{array}$$

and similarly for the right hand side:

$$\begin{array}{l} w'.\Sigma_r, \mathtt{newcase}_{[\![A]\!][\delta_r]}\ c_X; \\ \qquad [\![\mathtt{let}\ y = M\{X \cong B\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_X(X)\rangle\!\uparrow\ (\mathtt{seal}_X V_B)); V_A]\!][\gamma_r][\delta_r] \\ \\ \mapsto^0 w''.\Sigma_r, [\![\mathtt{let}\ y = M\{X \cong B\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_X(X)\rangle\!\uparrow\ (\mathtt{seal}_X V_B)); V_A]\!][\gamma_r'][\delta_r]\end{array}$$

where $\gamma' = \gamma, c_X \mapsto (w'.\Sigma_l.size, w'.\Sigma_r.size)$. Expanding definitions (and noting that c_X is free in M), we need to show

$$(w'', S_l[[\![M]\!][\gamma_l'][\delta_l]], S_2[[\![M]\!][\gamma_r'][\delta_r]]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

where

$$\begin{array}{l} S_l = f \leftarrow (z \leftarrow \bullet; \mathtt{force}\ z([\![A]\!][\delta_l])\ \gamma_l'(c_X)); \\ \qquad x \leftarrow [\![\langle \mathrm{tag}_X(X)\rangle\!\uparrow]\!][\gamma_l][\delta_l][[\![V_A]\!][\gamma_l][\delta_l]]; \\ \qquad \mathtt{force}\ f\ x\end{array}$$

and

$$S_2 = y \leftarrow \left(\begin{array}{l} f \leftarrow (z \leftarrow \bullet; \mathtt{force}\ z([\![B]\!][\delta_l])\ \gamma_r'(c_X)); \\ x \leftarrow [\![\langle \mathrm{tag}_X(X)\rangle\!\uparrow]\!][\gamma_r][\delta_r][[\![V_B]\!][\gamma_r][\delta_r]]; \\ \mathtt{force}\ f\ x\end{array}\right); [\![V_A]\!][\gamma_r][\delta_r]$$

We apply monadic bind. Let $w''' \sqsupseteq w''$ and $(w''', V_l, V_r) \in \mathcal{V}^{\sim}[\![\forall^\nu X.? \to X]\!]\gamma\delta$. After anti-reduction we need to show

$$\left(w''', \begin{array}{l} f \leftarrow \mathtt{force}\ V_l([\![A]\!][\delta_l])\, \gamma_l'(c_X); \\ \quad x \leftarrow [\![\langle \mathrm{tag}_X(X)\rangle\!\!\uparrow]\!][\gamma_l][\delta_l][[\![V_A]\!][\gamma_l][\delta_l]]; \\ \quad \mathtt{force}\ f\ x \end{array}, S_r'\left[\begin{array}{l} f \leftarrow \mathtt{force}\ V_r([\![B]\!][\delta_l])\, \gamma_r'(c_X); \\ x \leftarrow [\![\langle \mathrm{tag}_X(X)\rangle\!\!\uparrow]\!][\gamma_r][\delta_r][[\![V_B]\!][\gamma_r][\delta_r]]; \\ \mathtt{force}\ f\ x \end{array}\right]\right)$$

$$\in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

where

$$S_r' = y \leftarrow \bullet; [\![V_A]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

By weakening it is equivalent to showing the terms are in the relation $\mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$ where $\delta' = ([\![A]\!][\delta_l], [\![B]\!][\delta_r], R)$. By definition of $\mathcal{V}^{\sim}[\![\forall^\nu X.? \to X]\!]$, we have

$$(w''', \mathtt{force}\ V_l([\![A]\!][\delta_l])\, \gamma_l'(c_X), \mathtt{force}\ V_r([\![B]\!][\delta_r])\, \gamma_r'(c_X)) \in \mathcal{E}^{\sim}[\![? \to X]\!]\gamma'\delta'$$

So we apply monadic bind. Let $w'''' \sqsupseteq w'''$ and let $(w'''', V_{fl}, V_{fr}) \in \mathcal{V}^{\sim}[\![? \to X]\!]\gamma'\delta'$. Applying anti-reduction, we need to show

$$(w'''', \mathtt{force}\ V_{fl}\ (\mathtt{inj}_{\sigma_l}\ V_{Al}), y \leftarrow \mathtt{force}\ V_{fr}\ (\mathtt{inj}_{\sigma_r}\ V_{Br}); [\![V_A]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$$

By definition of $\mathcal{V}^{\sim}[\![? \to X]\!]$, we can prove $(w'''', \mathtt{force}\ V_{fl}\ (\mathtt{inj}_{\sigma_l}\ V_{Al}), \mathtt{force}\ V_{fr}\ (\mathtt{inj}_{\sigma_r}\ V_{Br})) \in \mathcal{E}^{\sim}[\![X]\!]\gamma'\delta'$ by showing that $(w'''', \mathtt{inj}_{\sigma_l}\ V_{Al}, \mathtt{inj}_{\sigma_r}\ V_{Br}) \in \mathcal{V}^{\sim}[\![?]\!]\gamma'\delta'$. This holds because $(w'''', V_{Al}, V_{Br}) \in \lfloor R \rfloor_{w''''.n}$ because $w''''.\eta \vDash (\sigma_l, \sigma_r, \lfloor R \rfloor_{w''''.n})$.

By Lemmas F.10 and F.2, we have that $[\![V_A]\!][\gamma_r][\delta_r] \mapsto^0 \mathtt{ret}\ V_{Ar}$ for some V_{Ar}. Applying monadic bind a final time we get $w_{fin} \sqsupseteq w''''$ and $(w_{fin}, V_{Al}', V_{Br}') \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$ and after anti-reduction need to show that

$$(w_{fin}, \mathtt{ret}\ V_{Al}', \mathtt{ret}\ V_{Ar}) \in \mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$$

By weakening this is equivalent to

$$(w_{fin}, \mathtt{ret}\ V_{Al}', \mathtt{ret}\ V_{Ar}) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

By the definition of $\mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$, $V_{Al}' = V_{Al}$, so this follows by reflexivity; specifically that $\Gamma \vDash V_A \sqsubseteq V_A \in A; \cdot$. This is because, by the definition of $\mathcal{E}^{\sim}[\![A]\!]\gamma\delta$, since $V_A[\gamma_l][\delta_l] \mapsto^0 \mathtt{ret}\ V_{Al}$ and $V_A[\gamma_r][\delta_r] \mapsto^0 \mathtt{ret}\ V_{Ar}$, those values are related. ◻

Theorem G.51. *For any $\cdot \vdash A, B$ and $V_A : A, V_B : B, \cdot \vdash V_d : ?$,*

$$\lambda\_ : ?.\mathtt{unseal}_X(M\{X \cong A\}\ (\mathtt{seal}_X V_A, V_d)) \approx^{ctx}$$

$$\lambda\_ : ?.\mathtt{let}\ y = (\mathtt{unseal}_X(M\{X \cong B\}\ (\mathtt{seal}_X V_B, V_d))); V_A$$

Proof. Again, we show one direction of the 4 cases, the others are analogous. Let $(w, \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p]\!]$. We need to show

$$\left(w, \begin{array}{l} \mathtt{ret\ thunk}\ \lambda\_ : ?. \\ \quad \mathtt{newcase}_{[\![A]\!]}\ c_X; \\ \quad [\![M\{X \cong A\}\ (\langle ?\rangle\!\!\downarrow \langle \mathrm{tag}_{?\times?}(X \times ?)\rangle\!\!\uparrow (\mathtt{seal}_X V_A), V_d)]\!][\gamma_l][\delta_l] \end{array}, \begin{array}{l} \mathtt{ret\ thunk}\ \lambda\_ : ?. \\ \quad \mathtt{newcase}_{[\![A]\!]}\ c_X; \\ \quad [\![\mathtt{let}\ y = M\{X \cong B\}\ (\langle ?\rangle\!\!\downarrow \langle \mathrm{tag}_{?\times?}(X \times ?)\rangle\!\!\uparrow (\mathtt{seal}_X V_B), V_d); V_A]\!][\gamma_r][\delta_r] \end{array}\right) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

Given some $w' \sqsupseteq w$ and (irrelevant) values (w', V_{duml}, V_{dumr}), we need to show (after applying anti-reduction) that

$$(w', \begin{array}{l}\texttt{newcase}_{[\![A]\!]}\ c_X;\\ \quad [\![M\{X \cong A\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_{?\times ?}(X \times ?)\rangle\!\uparrow\ (\texttt{seal}_X V_A), V_d)]\!][\gamma_l][\delta_l]\end{array}, \begin{array}{l}\texttt{newcase}_{[\![A]\!]}\ c_X;\\ \quad [\![\texttt{let}\ y = M\{X \cong B\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_{?\times ?}(X \times ?)\rangle\!\uparrow\ (\texttt{seal}_X V_B), V_d); V_A]\!][\gamma_r][\delta_r]\end{array}) \in \mathcal{E}^{\sim}[\![A]\!]\gamma\delta$$

Each side allocates a new case and again we to pick a relation with which to instantiate it. Again, by Lemmas F.10 and F.2, we have that $[\![V_A]\!][\gamma_l][\delta_l] \mapsto^0 \texttt{ret}\ V_{Al}$ and $[\![V_B]\!][\gamma_r][\delta_r] \mapsto^0 \texttt{ret}\ V_{Br}$ for some V_{Al} and V_{Br}. As in the ordinary identity function proof, we define R to be the "singleton" relation:

$$R = \{(w, V_{Al}, V_{Br}) \in \mathrm{Atom}[[\![A]\!][\delta_l], [\![B]\!][\delta_r]] \mid w \sqsupseteq w'\}$$

Then we define w'' to be the world extended with $\lfloor R \rfloor_{w'.j}$:

$$w'' = (w'.j, w'.\Sigma_l, [\![A]\!][\delta_l], w'.\Sigma_r[\delta_r], w'.\eta \boxplus (w'.\Sigma_l.size, w'.\Sigma_r.size, \lfloor R \rfloor_{w'.j})$$

Then clearly $w'' \sqsupseteq w'$ and

$$\begin{array}{l} w'.\Sigma_l, \texttt{newcase}_{[\![A]\!][\delta_l]}\ c_X;\\ \qquad [\![M\{X \cong A\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_{?\times ?}(X \times ?)\rangle\!\uparrow\ (\texttt{seal}_X V_A), V_d)]\!][\gamma_l][\delta_l] \\ \\ \mapsto^0 w''.\Sigma_l, [\![M\{X \cong A\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_{?\times ?}(X \times ?)\rangle\!\uparrow\ (\texttt{seal}_X V_A), V_d)]\!][\gamma_l'][\delta_l]\end{array}$$

and similarly for the right hand side:

$$\begin{array}{l} w'.\Sigma_r, \texttt{newcase}_{[\![A]\!][\delta_r]}\ c_X;\\ \qquad [\![\texttt{let}\ y = M\{X \cong B\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_{?\times ?}(X \times ?)\rangle\!\uparrow\ (\texttt{seal}_X V_B), V_d); V_A]\!][\gamma_r][\delta_r] \\ \\ \mapsto^0 w''.\Sigma_r, [\![\texttt{let}\ y = M\{X \cong B\}\ (\langle ?\rangle\!\downarrow\ \langle \mathrm{tag}_{?\times ?}(X \times ?)\rangle\!\uparrow\ (\texttt{seal}_X V_B), V_d); V_A]\!][\gamma_r'][\delta_r]\end{array}$$

where $\gamma' = \gamma, c_X \mapsto (w'.\Sigma_l.size, w'.\Sigma_r.size)$. Following the same argument as the previous cases, using monadic bind several times, we need to show that for some $w'''' \sqsupseteq w''$ and $(w'''', V_{fl}, V_{fr}) \in \mathcal{V}^{\sim}[\![? \to X]\!]\gamma'\delta'$, that

$$(w'''', \texttt{force}\ V_{fl}\ \texttt{inj}_{\gamma_l(c_{\mathrm{Times}})}\ (\texttt{inj}_{\sigma_l}\ V_{Al}, V_{dl}), y \leftarrow \texttt{force}\ V_{fr}\ \texttt{inj}_{\gamma_r(c_{\mathrm{Times}})}\ (\texttt{inj}_{\sigma_r}\ V_{Br}, V_{dr}); [\![V_A]\!][\gamma_r][\delta_r]) \in \mathcal{E}$$

By definition of $\mathcal{V}^{\sim}[\![? \to X]\!]$, we can prove

$$(w'''', \texttt{force}\ V_{fl}\ \texttt{inj}_{\gamma_l(c_{\mathrm{Times}})}\ (\texttt{inj}_{\sigma_l}\ V_{Al}, V_{dl}), \texttt{force}\ V_{fr}\ \texttt{inj}_{\gamma_r(c_{\mathrm{Times}})}\ (\texttt{inj}_{\sigma_r}\ V_{Br}, V_{dr})) \in ErelpX\gamma'\delta'$$

by showing that

$$(w'''', \texttt{inj}_{\gamma_l(c_{\mathrm{Times}})}\ (\texttt{inj}_{\sigma_l}\ V_{Al}, V_{dl}), \texttt{inj}_{\gamma_r(c_{\mathrm{Times}})}\ (\texttt{inj}_{\sigma_r}\ V_{Br}, V_{dr})) \in \mathcal{V}^{\sim}[\![?]\!]\gamma'\delta'.$$

since $(w'''', \gamma, \delta) \in \mathcal{G}^{\sim}[\![\Gamma_p]\!]$, which holds because $(w'''', V_{Al}, V_{Br}) \in R$ and $(w'''', V_{dl}, V_{dr}) \in \mathcal{E}^{\sim}[\![dyn]\!]\gamma'\delta'$.

So again we apply monadic bind, receiving some $w''''' \sqsupseteq w''''$ and $(w''''', V_{Xl}, V_{Xr}) \in \mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$ and we need to show (after applying anti-reduction) that

$$(w''''', \texttt{ret}\ V_{Xl}, [\![V_A]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$$

But by definition of $\mathcal{V}^{\sim}[\![X]\!]\gamma'\delta'$, we know that $V_{Xl} = V_{Al}$ and that $[\![V_A]\!][\gamma_l][\delta_l] \mapsto^0 \texttt{ret}\ V_{Al}$, so it is sufficient to show

$$(w''''', [\![V_A]\!][\gamma_l][\delta_l], [\![V_A]\!][\gamma_r][\delta_r]) \in \mathcal{E}^{\sim}[\![A]\!]\gamma'\delta'$$

which follows by reflexivity. □