Gradual Parametricity, Revisited (with Appendix)

MATÍAS TORO, University of Chile
ELIZABETH LABRADA, University of Chile
ÉRIC TANTER, University of Chile and Inria Paris

Bringing the benefits of gradual typing to a language with parametric polymorphism like System F, while preserving relational parametricity, has proven extremely challenging: first attempts were formulated a decade ago, and several designs were recently proposed. Among other issues, these proposals can however signal parametricity errors in unexpected situations, and improperly handle type instantiations when imprecise types are involved. These observations further suggest that existing polymorphic cast calculi are not well suited for supporting a gradual counterpart of System F. Consequently, we revisit the challenge of designing a gradual language with explicit parametric polymorphism, exploring the extent to which the Abstracting Gradual Typing methodology helps us derive such a language, GSF. We present the design and metatheory of GSF, and provide a reference implementation. In addition to avoiding the uncovered semantic issues, GSF satisfies all the expected properties of a gradual parametric language, save for one property: the dynamic gradual guarantee, which was left as conjecture in all prior work, is here proven to be simply incompatible with parametricity. We nevertheless establish a weaker property that allows us to disprove several claims about gradual free theorems, clarifying the kind of reasoning supported by gradual parametricity.

1 INTRODUCTION

There are many approaches to integrate static and dynamic type checking (Abadi et al. 1991; Bierman et al. 2010; Cartwright and Fagan 1991; Matthews and Findler 2007; Tobin-Hochstadt and Felleisen 2006). In particular, gradual typing supports the smooth integration of static and dynamic type checking by introducing the notion of imprecision at the level of types, which induces a notion of consistency between plausibly equal types (Siek and Taha 2006). A gradual type checker does a best effort statically, treating imprecision optimistically. The runtime semantics of the gradual language detects at runtime any invalidation of optimistic static assumptions. Such detection is usually achieved by compilation to an internal language with explicit casts, called a cast calculus. In addition to being type safe, a gradually-typed language is expected to satisfy a number of properties, in particular that it conservatively extends a corresponding statically-typed language, that it can faithfully embed dynamically-typed terms, and that the static-to-dynamic transition is smooth, a property formally captured as the (static and dynamic) gradual guarantees (Siek et al. 2015a).

Since its early formulation in a simple functional language (Siek and Taha 2006), gradual typing has been explored in a number of increasingly challenging settings such as subtyping (Garcia et al. 2016; Siek and Taha 2007), references (Herman et al. 2010; Siek et al. 2015b), effects (Bañados Schwerner et al. 2014, 2016), ownership (Sergey and Clarke 2012), typestates (Garcia et al. 2014; Wolff et al. 2011), information-flow typing (Disney and Flanagan 2011; Fennell and Thiemann 2013; Toro et al. 2018), session types (Igarashi et al. 2017b), refinements (Lehmann and Tanter 2017), set-theoretic types (Castagna and Lanvin 2017), Hoare logic (Bader et al. 2018) and parametric polymorphism (Ahmed et al. 2011, 2017; Igarashi et al. 2017a; Ina and Igarashi 2011; Xie et al. 2018).
In the case of parametric polymorphism, a long-standing challenge has been to prove that the gradual language preserves a rich semantic property known as relational parametricity (Reynolds 1983), which dictates that a polymorphic value must behave uniformly for all possible instantiations. The first gradual language to come with a proof of parametricity is the cast calculus $\lambda B$ (Ahmed et al. 2017), recently used as a target language by Xie et al. (2018). Another recent effort is System F$_G$, an actual gradual source language (i.e. without explicit casts), which is compiled to a cast calculus akin to $\lambda B$, called System F$_C$ (Igarashi et al. 2017a).

Contributions. This work starts from the novel identification of design issues in existing proposals, especially in their dynamic semantics. In short, parametricity errors can be raised in unexpected situations, and type instantiations are ignored when imprecise types are involved. Consequently, we argue that neither $\lambda B$ nor System F$_C$ are adequate targets for an explicitly-parametric gradual language (§2).

To this end, we introduce GSF, a gradual counterpart of System F that addresses the design issues identified in prior work and satisfies parametricity (§8). We explicitly lay out the design principles, goals and non-goals of GSF and introduce the language briefly through examples (§3). We then explain how we derive GSF from a variant of System F called SF (§4), by following the Abstracting Gradual Typing methodology (AGT) (Garcia et al. 2016). While the statics of GSF follow naturally from SF using AGT (§5), the dynamic semantics are more challenging (§6/§7). GSF satisfies the expected properties of gradual languages (§5/§7), except the dynamic gradual guarantee. This property was left open as a conjecture in prior work; here we prove that it is in fact incompatible with parametricity (§9). We uncover a novel, weaker property that GSF satisfies, which allows us to disprove several claims related to gradual free theorems for imprecise type signatures (§10).

Complete definitions and proofs of the main results can be found in appendices. Additionally, GSF is implemented as an interactive prototype that exhibits both typing derivations and reduction traces. All the examples mentioned in this paper, as well as others, are readily available in the online demo: https://pleiad.cl/gsf.

2 THE NEED TO REVISIT GRADUAL PARAMETRICITY

We start with a quick introduction to parametric polymorphism and parametricity, before motivating gradual parametricity through examples and finally exposing different issues in both the static and dynamic semantics of existing languages.

2.1 Background: Parametric Polymorphism

Parametric polymorphism allows the definition of terms that can operate over any type, with the introduction of type variables and universally-quantified types. For instance, a function of type $\forall X. X \to X$ can be used at any type, and returns a value of the same type as its actual argument. For the sake of this work, it is important to recall two crucial distinctions that apply to languages with parametric polymorphism, one syntactic—whether polymorphism is explicit or implicit—and one semantic—whether polymorphic types impose strong behavioral guarantees or not.

Explicit vs Implicit. In a language with explicit polymorphism, such as the Girard-Reynolds polymorphic lambda calculus (a.k.a. System F) (Girard 1972; Reynolds 1974), the term language includes explicit type abstraction $\Lambda X. e$ and explicit type application $e [T]$, as illustrated next:

let f : $\forall X. X \to X$ = $\Lambda X. \lambda x:X. x$ in f [Int] 10

The function $f$ has the polymorphic (or universal) type $\forall X. X \to X$. By applying $f$ to type Int (we also say that $f$ is instantiated to Int), the resulting function has type Int $\to$ Int; it is then passed the number 10. Hence the program evaluates to 10.
In contrast to this intrinsic, Church-style formulation, the Curry-style presentation of polymorphic type assignment (Curry et al. 1972) does not require type abstraction and type application to be reflected in terms. This approach, known as implicit polymorphism, has inspired many languages such as ML and Haskell. Technically, implicit polymorphism induces a notion of subtyping that relates polymorphic types to their instantiations (Mitchell 1988; Odersky and Läuer 1996); e.g. \( \forall X.X \to X <: \text{Int} \to \text{Int} \). Implicitly-polymorphic languages generally use an explicitly-polymorphic language underneath (e.g. GHC Core), providing the convenience of implicitness through an inference phase that produces an explicitly-annotated program. In essence, the use of the subtyping judgment \( \forall X.X \to X <: \text{Int} \to \text{Int} \) is materialized in terms by introducing an explicit instantiation [\( \text{Int} \)], and vice-versa, the use of the judgment \( \text{Int} \to \text{Int} <: \forall X.\text{Int} \to \text{Int} \) is materialized by inserting a type abstraction constructor \( \Lambda X \).

**Genericity vs. Parametricity.** Some languages with universal type quantification also support intensional type analysis or reflection, which allows a function to behave differently depending on the type to which it is instantiated. For instance, in Java, a generic method of type \( \forall X.X \to X \) can use instanceof to discriminate the actual type of the argument, and behave differently for \text{String}, say, than for \text{Integer}. Therefore these languages only support genericity, i.e. the fact that a value of a universal type can be safely instantiated at any type.\(^1\)

Parametricity is a much stronger interpretation of universal types, which dictates that a polymorphic value must behave uniformly for all possible instantiations (Reynolds 1983). This implies that one can derive interesting theorems about the behavior of a program by just looking at its type, hence the name “free theorems” coined by Wadler (1989). For instance, one can prove using parametricity that any polymorphic list permutation function commutes with the polymorphic map function. Technically, parametricity is expressed in terms of a (type-indexed) logical relation that denotes when two terms behave similarly when viewed at a given type. All well-typed terms of System F are related to themselves in this logical relation, meaning in particular that all polymorphic terms behave uniformly at all instantiations (Reynolds 1983).

Simply put, if a value \( f \) has type \( \forall X.X \to X \), genericity only tells us that \( f [\text{Int}]\ 10 \) reduces to some integer, while parametricity tells the much stronger result that \( f [\text{Int}]\ 10 \) necessarily evaluates to 10 (i.e. \( f \) has to be the identity function). In the context of gradual typing, Ina and Igarashi (2011) have explored genericity with a gradual variant of Java. All other work has focused on the challenge of enforcing parametricity (Ahmed et al. 2011, 2017; Igarashi et al. 2017a; Xie et al. 2018).

### 2.2 Gradual Parametricity in a Nutshell

**Basics** Gradual parametricity supports imprecise typing information, yet ensures that assumptions about parametricity are enforced at runtime whenever they are not provable statically. In the following program, function \( f \) expects a function \( g \) of type \( \forall X.X \to X \) as argument. It is applied to an argument \( h \) of the unknown type. By consistency, this program is well-typed; however the compliance of \( h \) with respect to its assumed parametric signature is unknown statically.

\[
\text{let } f = \lambda g:(\forall X.X \to X).g [\text{Int}]\ 10 \text{ in let } h : ? = \ldots \text{ in } f \ h
\]

By parametricity, function \( f \) can deduce that \( g \) behaves like the identity function (§2.1). In presence of gradual types—as in any variant of System F with errors and non-termination—this conclusion should be relaxed: gradual simple types admit both non-termination (Siek and Taha 2006) and runtime type errors. Therefore, as a consequence of parametricity, we can prove that if the program above terminates, it should either produce 10, or fail with a runtime error, possibly denoting that \( h \) was in fact not a proper identity function.

\(^1\)We call this property *genericity*, by analogy to the name *generics* in use in object-oriented languages like Java and C#.
Let us consider three possible implementations of \( h \):

\[
\begin{align*}
h_1 &= \Lambda X. \lambda x : X. x \\
h_2 &= \Lambda X. \lambda x : ? . x \\
h_3 &= \Lambda X. \lambda x : ? . x + 1
\end{align*}
\]

Function \( h_1 \) is the standard System F identity function, and function \( h_2 \) is a less precise version, which behaves identically. Therefore, using either of these functions in the program above produces the result 10. Conversely, function \( h_3 \) is not a proper identity function. Note that the function is well-typed, because \( x \) has type \( ? \) in the body. Also, using \( h_3 \) in the program above is type safe, because \( f \) happens to instantiate its argument at type \( \text{Int} \), so execution could proceed safely without errors and yield 11; this would however be a violation of parametricity, so an error should be raised.

**State of the Art.** While the basics of gradual parametricity are well understood, the details are tricky. In particular, establishing that a gradual parametric language enforces parametricity has been a long-standing open issue: early work on the polymorphic blame calculus did not prove parametricity (Ahmed et al. 2009, 2011); only very recent work on a variant of that calculus, \( \lambda B \), has achieved this result (Ahmed et al. 2017). In fact, \( \lambda B \) is a cast calculus, not a gradual source language, meaning that the program written above would not be valid; explicit casts should be sprinkled in different places to achieve the same result. Igarashi et al. recently developed a gradual source language, System \( F_G \), which does support the intended lightweight, cast-free syntax of gradual languages. Following the early tradition of gradual typing (Siek and Taha 2006), the semantics of System \( F_G \) are given by translation to a cast calculus, System \( F_C \), which is a close cousin of \( \lambda B \). Igarashi et al. in fact do not prove parametricity, but conjecture that due to the similarity between System \( F_G \) and \( \lambda B \), parametricity should hold. Xie et al. (2018) develop a language with implicit polymorphism (here referred to as CSA), which compiles to \( \lambda B \) and therefore satisfies parametricity.

On the metatheoretic front, beyond parametricity, there are other important properties that are relevant for gradual languages, most notably the conservative extension and the gradual guarantees (Siek et al. 2015a). The former states that, on fully static programs, a gradual language should behave exactly like its static counterpart. The latter states that making types less precise does not introduce static or dynamic type errors. \( \lambda B \) is not a conservative extension of System F (§2.3), and the gradual guarantees are left as an open question. System \( F_G \) is a conservative extension of System F, and CSA of an implicit variant of System F. Both System \( F_G \) and CSA satisfy the static gradual guarantee, although System \( F_G \) uses an ad hoc notion of precision tuned to that effect (§2.3). The dynamic gradual guarantee for both System \( F_G \) and CSA are still open questions.

Finally, gradual free theorems about imprecise type signatures have not been formally studied, beyond a number of claims that we mention below and disprove in §10.

### 2.3 Static Semantics Issues

While the static semantics of simple gradual languages are uncontroversial, devising the static semantics of gradual polymorphic languages has proven to be fairly challenging, yielding systems that are arguably hard to grasp. We highlight the most salient issues with \( \lambda B \) and System \( F_G \) below, and then relate to CSA, which addresses them to some extent.

**Mixing Explicit and Implicit Polymorphism.** Both \( \lambda B \) and System \( F_G \) are languages with *explicit* polymorphism, i.e. with explicit type abstraction and type application terms. However, instead of focusing on explicit polymorphism only, both languages accommodate some form of implicitness, but with different flavors. Consider the type of a polymorphic identity function, \( \forall X. X \rightarrow X \). In \( \lambda B \) this type is compatible with \( \text{Int} \rightarrow \text{Int} \), which is a defining feature of *implicit* polymorphism. More surprisingly, this type is also compatible with \( \text{Int} \rightarrow \text{Bool} \). (Runtime errors will account for the obvious mistake.) This means in particular that \( \lambda B \) is not a proper conservative extension of System F, as both type systems disagree on some fully static terms. Technically, instead
of the traditional consistency relation, \( \lambda B \) introduces two close but distinct relations on types, called convertibility and compatibility, in order to orchestrate these non-trivial semantics. Conversely, System \( F_G \) relies on a notion of consistency, and is a proper conservative extension of System F. As an explicitly polymorphic language, System \( F_G \) does not relate \( \forall X.X \to X \) with any of its static instantiations. However, it does relate that type with \( X \to ? \), considered to be quasi-polymorphic, on the basis that using the unknown type should bring some of the flexibility of implicit polymorphism.

**Ad-hoc Precision.** Conversely to System \( F_G \), \( \lambda B \) has no notion of type precision, and does not discuss any of the gradual guarantees. The precision relation of System \( F_G \) features some constraints that might be surprising to programmers. Specifically, System \( F_G \) allows loss of precision only in non-parametric positions of a polymorphic type. For instance, \( \forall X.X \to \text{Int} \) is considered more precise than \( \forall X.X \to ? \), but unrelated to \( \forall X.? \to \text{Int} \). Because precision induces consistency, it means that \( \forall X.X \to \text{Int} \) and \( \forall X.? \to \text{Int} \) are inconsistent with each other. This choice is motivated by the desire to avoid a counterexample of the gradual guarantee: they claim that a function of type \( \forall X.? \to X \) must fail on all inputs in order to respect parametricity (we disprove this claim in §10), so accepting that this type is less precise than \( \forall X.X \to X \) breaks the dynamic gradual guarantee.

But tailoring the precision relation to avoid a class of counterexamples is not benign. First, changing the definition of precision to accommodate a theorem does not necessarily result in a programmer’s expectations being adjusted. Let us recall that the gradual guarantees were introduced by Siek et al. (2015a) in order to formally capture the expectations of programmers using gradual languages. The restriction on precision imposed by System \( F_G \) breaks the intuition of programmers that, starting program from a well-typed program, removing static type information yields a program that is by definition less precise—and should also be well-typed.

Second, the restricted rule excludes instances of precision that are harmless for the dynamic gradual guarantee. For instance, in System \( F_G \), \( \forall X.X \to X \) is not more precise than \( \forall X.X \to ? \), despite the fact that a function of type \( \forall X.X \to ? \) can be a proper identity function (§10).

Third, Igarashi et al. (2017a) only prove the static guarantee based on this ad hoc precision, and leave the dynamic guarantee as a conjecture, so it is unclear whether the restriction on precision imposed by System \( F_G \) is indeed sufficient.

**Separating Concerns.** Recently, Xie et al. (2018) raise similar concerns about the static semantics of \( \lambda B \) and System \( F_G \), in particular regarding the mixing of explicit and implicit polymorphism. In response, they clearly separate the subtyping relation induced by implicit polymorphism from the consistency relation induced by gradual types. Their notion of consistent subtyping extends the notion of Siek and Taha (2007). As a result, CSA features intuitive and straightforward definitions of precision and consistency, while accommodating the flexibility of implicit polymorphism in full.

We fully concur with the necessity to untangle implicitness from consistency in order to achieve a principled design. Xie et al. leave open the question of designing an explicitly-polymorphic gradual language. Additionally, Xie et al. do not deal with the dynamic semantics of their language beyond a translation to \( \lambda B \). Therefore CSA inherits both the virtues of \( \lambda B \), such as parametricity, and its issues, uncovered next.\(^2\)

### 2.4 Dynamic Semantics Issues

In the design of gradually-typed languages, cast calculi are typically used as target languages to give runtime semantics to gradual programs. However, as observed by Garcia et al. (2016), there is little justification or guidance available to design or choose a cast calculus for interpreting a given

---

\(^2\)The implicit polymorphism of Xie et al. (2018) faces other challenges, most notably the lack of coherence of the runtime semantics. This issue is entirely related to implicit polymorphism and is therefore not addressed here.
Matías Toro, Elizabeth Labrada, and Éric Tanter

gradual source language. To this date, only the Abstracting Gradual Typing methodology (AGT) provides a systematic approach to derive the dynamic semantics of gradual languages by directly giving meaning to gradual typing derivations (García et al. 2016).

Since the early work on the polymorphic blame calculus (Ahmed et al. 2009, 2011), all existing work has built upon variants of that cast calculus. While a cast language like \( \lambda B \) can be used as a source language (Ahmed et al. 2017), \( \lambda B \) has been used in recent work as the target language of choice for gradual source languages (Igarashi et al. 2017a; Xie et al. 2018). In this section, we identify two questionable design decisions in both \( \lambda B \) and System F that arguably make them inadequate as internal languages of a gradual version of System F.

**Excess of Failure.** Consider the following example, written in System \( F_G \) (the \( \lambda B \) and System \( F_C \) versions are more verbose because of explicit casts):

```plaintext
let f : \( \forall X. X \rightarrow ? \) = \( \lambda x : X. x \) in (f [Int] 1) + 1
```

What would the programmer expect out of this program? While the annotated return type of \( f \) is left unknown, the function itself is the System F identity function. Therefore, one might expect that instantiating the function to \( \text{Int} \), passing 1 and adding 1, should yield 2 as a result.

However, in both \( \lambda B \) and System \( F_C \), the above program fails with a runtime error. The reason is that the result of \( f \) \([\text{Int}]\) 1 is sealed, and therefore unusable directly. Ahmed et al. (2011) justify this behavior (already present in early work (Ahmed et al. 2009)), or the alternative of always failing before returning, based on a claim about gradual free theorems. Intuitively, this can be surprising because the underlying value is the System F identity function, which does behave parametrically; it is therefore unclear what parametricity violation is being reported. As we will see later, this failing behavior is in fact not formally demanded by parametricity (§10).

**Lack of Failure.** A major interest of gradual types is that they *soundly* augment the expressiveness of the original static type system. Let us illustrate first in a simply-typed setting (STLC refers to the simply-typed lambda calculus with base types):

- Consider the STLC term \( t = \lambda x : \_x, \) which behaves as the identity function. \( t \) is incomplete because the type annotation on \( x \) is missing so far.
- \( t \) is operationally valid at different types, but it cannot be given a general type in STLC. Its type has to be fixed at either \( \text{Int} \rightarrow \text{Int}, \text{Bool} \rightarrow \text{Bool}, etc. \)
- Intuitively, a proper characterization of \( t \) requires going from simple types to parametric polymorphism, such as System F. In System F, we could use the type \( \forall X.X \rightarrow X \) to precisely specify that \( t \) can be applied with any argument type and return the same type.
- With a gradual variant of STLC, we can give term \( t \) the imprecise type \( \_ \rightarrow \_ \) to statically capture the fact that \( t \) is definitely a function, without committing to specific domain and codomain types.
- This lack of precision is soundly backed by runtime enforcement, such that the term \((t 3) 1\) evaluates to a runtime type error.

The exact same line of reasoning should apply when starting from System F, as follows:

- Consider the System F term \( t = \lambda x : \_x \right[\text{Int}\], \) which behaves as an instantiation function to \( \text{Int}. \) \( t \) is incomplete because the type annotation on \( x \) is missing so far.
- \( t \) is operationally valid at different types, but it cannot be given a general type in System F. It has to be fixed at either \( \forall X.X \rightarrow X \rightarrow (\text{Int} \rightarrow \text{Int}), (\forall XY.X \rightarrow Y \rightarrow X) \rightarrow (\forall Y.\text{Int} \rightarrow Y \rightarrow \text{Int}), etc. \)
- Intuitively, a proper characterization of \( t \) requires going from System F to higher-order polymorphism, such as System \( F_{\omega} \). In System \( F_{\omega} \), we could use the type \( \forall P.(\forall X.P X) \rightarrow (P \text{Int}) \) to precisely specify that \( t \) instantiates any polymorphic argument to \( \text{Int} \).
With a gradual variant of System F, we ought to be able to give term \( t \) the imprecise type \( (\forall X.? \rightarrow ?) \rightarrow ? \) to statically capture the fact that \( t \) is definitely a function that operates on a polymorphic argument, without committing to a specific domain scheme and codomain type.

This lack of precision ought to be soundly backed by runtime enforcement, such that, given \( id : \forall X.X \rightarrow X \), the term \( (t \ id) \ true \) should evaluate to a runtime type error.

However, the runtime semantics of \( \lambda B \) and System \( F_C \) suffer from a fundamental issue that breaks the argument above: they do not respect type instantiations that involve the unknown type, and consequently do not fail as expected. Below is another simple example in System \( F_G \) in which the polymorphic identity function is instantiated to \( \text{Int} \) and passed a \( \text{Bool} \) value:

\[
\text{let } g : ? = \Lambda X. \lambda x : X. x \text{ in } g \ [\text{Int}] \ true
\]

This System \( F_G \) program (and its translation to \( \lambda B \)) returns \( \text{true} \), despite the explicit instantiation to \( \text{Int} \). Internally, this happens because \( g \) is first consistently considered to be of type \( \forall X.? \) in order to accommodate the type instantiation, but then the instantiation yields a substitution of \( \text{Int} \) for \( X \) in \( ? \), which in both languages is just \( ? \). There is no tracking of the decision to instantiate the underlying value to \( \text{Int} \). Consequently, current polymorphic cast calculi such as \( \lambda B \) and System \( F_C \) are inadequate to serve as the runtime support of a gradual variant of System \( F \).

### 3 GSF, INFORMALLY

This paper presents the design, semantics and metatheory of GSF, a gradual counterpart of System \( F \) that addresses the issues raised above. This section focuses on the informal aspects of GSF: design principles and methodology, as well as some illustrative examples of GSF in action.

#### 3.1 Design Principles, Goals and Non-Goals

Considering the many concerns involved in developing a gradual language with parametric polymorphism, we should be very clear about the principles, goals and non-goals of a specific design. In designing GSF, we respect the following design principles:

**Explicit polymorphism:** GSF is a gradual counterpart to System \( F \), and as such, is a fully *explicitly* polymorphic language: type abstraction and type application are part of the term language, reflected in types. GSF gradualizes type information, not term structure.

**Simple statics:** GSF must embody the complexity of dynamically enforcing parametricity solely in its dynamic semantics; its static semantics should be as straightforward as possible.

**Natural precision:** Precision is intended to capture the level of static typing information of a gradual type, with \( ? \) as the least precise and static types as the most precise (Siek et al. 2015a). GSF should preserve this simple intuition.

The mandatory goals for GSF, *i.e.* the properties that it should definitely satisfy, are:

**Type safety:** GSF should be type safe, meaning all programs should either evaluate to a value, halt with a runtime error, or diverge. Well-typed GSF terms should not get stuck.

**Conservative extension:** GSF should be a conservative extension of System \( F \): both languages should coincide in their static and dynamic semantics for fully static programs.

**Faithful instantiations:** GSF should respect type instantiations. In particular, explicit instantiations of imprecise types should be enforced.

**Parametricity:** GSF should enforce the notion of parametricity understood for gradual programs (Ahmed et al. 2017). In particular, a polymorphic function should behave uniformly across all its instantiations—*i.e.* always take related inputs to related outputs, or always fail or diverge.

\[ \text{In System } F_C, (t \ id) \ true \text{ fails because } \forall X.? \text{ is not deemed consistent with } \forall X.X \rightarrow X. \text{ Consequently, } t \text{ must be declared to take an argument of type } ? \text{ instead of } \forall X.? . \text{ The result is the same as in } \lambda B \text{ however: no runtime error is raised.} \]
**Static gradual guarantee:** By virtue of the simple statics principle stated above, GSF should satisfy the static gradual guarantee, i.e. typeability should be monotonic with respect to the natural notion of precision.

Similarly important are the explicit non-goals that we adopt when designing GSF:

**Dynamic gradual guarantee:** While GSF should strive to satisfy the dynamic gradual guarantee, this should not be at the expense of any of the above-stated principles and goals. In other words, the dynamic gradual guarantee is the first candidate property to abandon (or weaken) if need be.

**Implicit polymorphism:** While implicit polymorphism is certainly a desirable feature for usability, the integration of implicit polymorphism in GSF is future work.

**Blame tracking:** Tracking blame in order to report more informative error messages is valuable, but most important is to properly identify error cases. As discussed in §2.4, \( \lambda B \) and System \( F_G \) both miss important errors and raise errors in unexpected situations.

**Performance:** We focus on the semantics and meta-theoretical properties of GSF, without explicitly taking into account efficiency considerations such as pay-as-you-go (Igarashi et al. 2017a; Siek and Taha 2006), space efficiency (Herman et al. 2010; Siek and Wadler 2010), cast elimination (Rastogi et al. 2012), etc. Optimizing the dynamic semantics of GSF is left for future work.

### 3.2 Design Methodology

In order to assist language designers in crafting new gradual languages, Garcia et al. (2016) proposed the Abstracting Gradual Typing methodology (AGT, for short). The promise of AGT is that, starting from a specification of the meaning of gradual types in terms of the set of possible static types they represent, one can systematically derive all relevant notions, including precision, consistent predicates (e.g. consistency and consistent subtyping), consistent functions (e.g. consistent meet and join), as well as a direct runtime semantics for gradual programs, obtained by reduction of gradual typing derivations augmented with evidence for consistent judgments.

The AGT methodology has so far proven effective to assist in the gradualization of a number of disciplines, including effects (Baños Schwerter et al. 2014, 2016), record subtyping (Garcia et al. 2016), set-theoretic types (Castagna and Lanvin 2017), union types (Toro and Tanter 2017), refinement types (Lehmann and Tanter 2017) and security types (Toro et al. 2018). The applicability of AGT to gradual parametricity is an open question repeatedly raised in the literature—see for instance the discussions of AGT by Igarashi et al. (2017a) and Xie et al. (2018). Considering the variety of successful applications of AGT, and the complexity of designing a gradual parametric language, in this work we decide to adopt this methodology, and report on its effectiveness.

### 3.3 GSF in Action

Recall the example from §2.2, in which a function \( f \) defined as:

\[
\lambda g. (\forall X. X \rightarrow X). g [\text{Int}] 10
\]

is applied to a function \( h \) of unknown type. GSF behaves exactly as described with each of the three variant implementations of \( h \), namely:

\[
\text{let } h : ? = \lambda x : X. x \text{ in } f \ h \quad ----> \ 10
\]

\[
\text{let } h : ? = \lambda x : ?. x \text{ in } f \ h \quad ----> \ 10
\]

\[
\text{let } h : ? = \lambda x : ?. x + 1 \text{ in } f \ h \quad ----> \text{error}
\]

In the last case, the runtime error is raised when the body of the function attempts to perform an addition, since this type-specific operation is a violation of parametricity.

The fact that GSF adopts explicit polymorphism à la System F means that a polymorphic type is not consistent with any of its instantiations. In practice, this means that:
We systematically derive GSF by applying AGT to a largely standard polymorphic language similar to System F, called SF (Figure 1). In addition to the standard System F types and terms, SF includes base types \( B \) inhabited by constants \( b \), typed using the auxiliary function \( ty \), and primitive n-ary operations \( op \) that operate on base types and are given meaning by the function \( \delta \). SF also includes pairs \( \langle t_1, t_2 \rangle \), and the associated projection operations \( \pi_i(t) \), as well as type ascriptions \( t :: T \).

4 PRELIMINARY: THE STATIC LANGUAGE SF

We systematically derive GSF by applying AGT to a largely standard polymorphic language similar to System F, called SF (Figure 1). In addition to the standard System F types and terms, SF includes base types \( B \) inhabited by constants \( b \), typed using the auxiliary function \( ty \), and primitive n-ary operations \( op \) that operate on base types and are given meaning by the function \( \delta \). SF also includes pairs \( \langle t_1, t_2 \rangle \), and the associated projection operations \( \pi_i(t) \), as well as type ascriptions \( t :: T \).

The statics are standard. The typing judgment is defined over three contexts: a type name store \( \Sigma \) (explained below), a type variable set \( \Delta \) that keeps track of type variables in scope, and a standard type environment \( \Gamma \) that associates term variables to types. We adopt the convention of using partial type functions to denote computed types in the rules: \( \text{dom} \) and \( \text{cod} \) for domain and codomain types, \( \text{inst} \) for the resulting type of an instantiation, and \( \text{proj} \) for projected types. These partial functions are undefined if the argument is not of the appropriate shape. We also make the use of type equality explicit as a premise whenever necessary. These conventions are helpful for lifting the static semantics to the gradual setting (Garcia et al. 2016). For closed terms, we write \( ;; \cdot \cdot t : T \), or simply \( \cdot \cdot t : T \).

The dynamics are standard call-by-value semantics, specified using reduction frames. The only peculiarity is that they rely on runtime type generation: upon type application, a fresh type name \( \alpha \) is generated and bound to the instantiation type \( T \) in a global type name store \( \Sigma \). The notion of reduction and reduction rules all carry along the type name store. While type names only occur at runtime, and not in source programs, reasoning about SF terms as they reduce requires accounting for programs with type names in them. This is why the typing rules are defined relative to a type name store as well. Similarly, type equality is relative to a type name store: a type name \( \alpha \) is considered equal to its associated type in the store. The recursive definition of equality modulo type names is necessary to derive equalities (Igarashi et al. 2017a). For instance, in the reduction of the well-typed program \( (id \ [\text{Int} \to \text{Int}]) \ (id \ [\text{Int}]) \), where \( id \) is the polymorphic identity function, the equality \( \alpha := \text{Int} \to \text{Int}, \beta := \text{Int}; \Delta + \alpha = \beta \to \beta \) should be derivable.

Rules in Figure 1 appeal to auxiliary well-formedness judgments, omitted for brevity (§A). A type \( T \) is well-formed (\( \Sigma; \Delta + T \)) if it only contains type variables in the type variable environment \( \Delta \), and type names bound in a well-formed type name store. A type name store is well-formed (\( \cdot \cdot \cdot \Sigma \)) if all type names are distinct, and associated to well-formed types. A type environment \( \Gamma \) binds term variables to types, and is well-formed (\( \Sigma; \Delta + \Gamma \)) if all types are well-formed.

The decision of using type names instead of the traditional substitution semantics is in anticipation of gradualization: indeed, prior work has shown that runtime type generation is crucial in order to

\[
\text{let } h : ? = \lambda x : ?. x \text{ in } f h \quad \longrightarrow \text{ error}
\]

The runtime error occurs when the body of \( f \) performs the type application, because the value bound to \( g \) is not of the appropriate constructor \( (\Lambda) \). If changing the definition of \( h \) to include the \( \Lambda \) constructor is not an option, one can perform this adaptation explicitly upon application of \( f \):

\[
\text{let } h : ? = \lambda x : ?. x \text{ in } f \ (\Lambda x . h) \quad \longrightarrow \text{ 10}
\]

Finally, GSF does not report spurious parametricity violations, and enforces type instantiations even when applied to an imprecisely-typed value:

\[
\text{let } f : \forall x . x \to ? = \Lambda x . \lambda x : x . x \text{ in } (f \ [\text{Int}]) \ 1 \ + \ 1 \quad \longrightarrow \text{ 2}
\]

\[
\text{let } g : ? = \Lambda x . \lambda x : x \text{ in } g \ [\text{Int}] \ true \quad \longrightarrow \text{ error}
\]

Hence GSF addresses the issues in the dynamic semantics of \( \lambda B \) and System F_{C}, and soundly augments the expressiveness of System F (§2.4). Other illustrative examples are available online.
\( x \in \text{VAR}, X \in \text{TYPEVAR}, \alpha \in \text{TYPE_NAME} \quad \Sigma \in \text{TYPE_NAME} \rightarrow \text{TYPE}, \Delta \subset \text{TYPEVAR}, \Gamma \in \text{VAR} \rightarrow \text{TYPE} \)

\begin{align*}
T & ::= B \mid T \rightarrow T \mid \forall X.T \mid T \times T \mid X \mid \alpha \\
T & ::= b \mid \lambda x : T.t \mid \Lambda X.t \mid (t, t) \mid x \mid t :: T \mid \text{op}(t) \mid t \setminus t [T] \mid \pi_i(t) \\
v & ::= b \mid \lambda x : T.t \mid \Lambda X.t \mid \langle v, v \rangle
\end{align*}

\( \Sigma; \Delta; \Gamma \vdash t : T \)  

**Well-typed terms**

\[
\begin{align*}
\text{(Th)} & \quad ty(b) = B & \Sigma; \Delta; \Gamma \vdash b : B \\
\text{(TA)} & \quad \Sigma; \Delta, X; \Gamma \vdash t : T & \Sigma; \Delta; \Gamma \vdash \Lambda X.t : \forall X.T \\
\text{(Tx)} & \quad x : T \in \Gamma & \Sigma; \Delta; \Gamma \vdash \lambda x : T.t : T \\
\text{(Top)} & \quad \Sigma; \Delta, \Gamma \vdash t : T & \Sigma; \Delta; \Gamma \vdash \langle \text{op}(t) : T \rangle \\
\text{(Tap)} & \quad \Sigma; \Delta; \Gamma \vdash t : T & \Sigma; \Delta; \Gamma \vdash \langle \text{proj}(T) \rangle
\end{align*}
\]

\( \text{dom} : \text{TYPE} \rightarrow \text{TYPE} \quad \text{cod} : \text{TYPE} \rightarrow \text{TYPE} \quad \text{inst} : \text{TYPE}^2 \rightarrow \text{TYPE} \quad \text{proj} : \text{TYPE} \rightarrow \text{TYPE} \)

\( \text{dom}(T_1 \rightarrow T_2) = T_1 \quad \text{cod}(T_1 \rightarrow T_2) = T_2 \quad \text{inst}(\forall X.T, T') = T'[T'/X] \quad \text{proj}(T_1 \times T_2) = T_1 \)

\( \text{dom}(T) \text{ undefined o/w} \quad \text{cod}(T) \text{ undefined o/w} \quad \text{inst}(T, T') \text{ undefined o/w} \quad \text{proj}(T) \text{ undefined o/w} \)

\( \Sigma; \Delta \vdash T = T \)  

**Type equality**

\[
\begin{align*}
\Sigma; \Delta \vdash B = B \\
\Sigma; \Delta \vdash X = X \\
\Sigma; \Delta \vdash T_1 = T_2 \\
\Sigma; \Delta \vdash \forall X.T_1 = \forall X.T_2 \\
\Sigma; \Delta \vdash \alpha = \alpha \\
\Sigma; \Delta \vdash T = \alpha
\end{align*}
\]

\( \Sigma \vdash t \rightarrow \Sigma \vdash t \)  

**Notion of reduction**

\[
\begin{align*}
\Sigma \vdash v :: T & \rightarrow \Sigma \vdash v \\
\Sigma \vdash \text{op}(\overline{v}) & \rightarrow \Sigma \vdash \delta(\text{op}, \overline{v}) \\
\Sigma \vdash (\lambda x : T.t) v & \rightarrow \Sigma \vdash t[v/x] \\
\Sigma \vdash (\Lambda X.t) [T] & \rightarrow \Sigma, \alpha :: T = t[\alpha/X] \quad \text{where } \alpha \not\in \text{dom}(\Sigma) \\
\Sigma \vdash \pi_i(\langle v_1, v_2 \rangle) & \rightarrow \Sigma \vdash v_i
\end{align*}
\]

\( \Sigma \vdash t \rightarrow \Sigma \vdash t \)  

**Evaluation frames and reduction**

\[
\begin{align*}
f & ::= \square :: T \mid \text{op}(\overline{v}, \square, \overline{t}) \mid \square \rightarrow v \mid v \square \mid \square [T] \mid \langle \overline{v}, \overline{t} \rangle \mid \langle v, \square \rangle \mid \pi_i(\square) \\
\Sigma \vdash t & \rightarrow \Sigma' \vdash t' \\
\Sigma \vdash t & \rightarrow \Sigma' \vdash t'
\end{align*}
\]

Fig. 1. SF: Simple Static Polymorphic Language with Runtime Type Generation
We introduce the syntactic category of gradual types $G$ we can simply extend this syntactic approach to deal with universal types, type variables, and $\text{Int}$ words, wanting the dynamics and type soundness argument of the static language to help us with GSF. (and GSF is a language with explicit $\forall$ than $\exists$)

Unsurprisingly, SF is type safe, and all well-typed terms are parametric. These results also follow from the properties of GSF, and the strong relation between both languages.

5 GSF: STATICS

The first step of the Abstracting Gradual Typing methodology (AGT) is to define the syntax of gradual types and give them meaning through a concretization function to the set of static types they denote. Then, by finding the corresponding abstraction function to establish a Galois connection, the static semantics of the static language can be lifted to the gradual setting.

5.1 Syntax and Syntactic Meaning of Gradual Types

We introduce the syntactic category of gradual types $G \in \text{GType}$, by admitting the unknown type in any position, namely:

$$G ::= B \mid G \rightarrow G \mid \forall X. G \mid G \times G \mid X \mid \alpha \mid ?$$

Observe that static types $T$ are syntactically included in gradual types $G$.

The syntactic meaning of gradual types is straightforward: the unknown type represents any type, and a precise type (constructor) represents the equivalent static type (constructor). In other words, $\text{Int} \rightarrow ?$ denotes the set of all function types from $\text{Int}$ to any static type. Perhaps surprisingly, we can simply extend this syntactic approach to deal with universal types, type variables, and type names; the concretization function $C$ is defined in Figure 2. Note that the definition is purely syntactic and does not even consider well-formedness (? stands for any static type); notions built above concretization, such as consistency, will naturally embed the necessary restrictions (§5.2).

Following the abstract interpretation framework, the notion of precision is not subject to tailoring: precision coincides with set inclusion of the denoted static types (Garcia et al. 2016).

**Definition 5.1 (Type Precision).** $G_1 \subseteq G_2$ if and only if $C(G_1) \subseteq C(G_2)$.

**Proposition 5.2 (Precision, Inductively).** The inductive definition of type precision given in Figure 3 is equivalent to Definition 5.1.

Observe that both $\forall X. X \rightarrow ?$ and $\forall X. ? \rightarrow X$ are more precise than $\forall X. ? \rightarrow ?$, and less precise than $\forall X. X \rightarrow X$, thereby reflecting the original intuition about precision (Siek et al. 2015a). Also $\forall X. ? \rightarrow ?$ and $? \rightarrow ?$ are unrelated by precision, since they correspond to different constructors (and GSF is a language with explicit polymorphism); they are both more precise than $?$, of course.
Dual to concretization is abstraction, which produces a gradual type from a non-empty set of static types. The abstraction function $A$ is direct (Figure 2): it preserves type constructors and falls back on the unknown type whenever an heterogeneous set is abstracted. $A$ is both sound and optimal: it produces the most precise gradual type that over-approximates a given set of static types.

**Proposition 5.3 (Galois connection).** \( (C, A) \) is a Galois connection, i.e.: 
\( a) \) (Soundness) for any non-empty set of static types $S = \{ \overline{T} \}$, we have $S \subseteq C(A(S))$ 
\( b) \) (Optimality) for any gradual type $G$, we have $A(C(G)) \subseteq G$.

### 5.2 Lifting the Static Semantics

The key point of AGT is that once the meaning of gradual types is agreed upon, there is no space for ad hoc design in the static semantics of the language. The abstract interpretation framework provides us with the definitions of type predicates and functions over gradual types, for which we can then find equivalent inductive or algorithmic characterizations.

In particular, a predicate on static types induces a counterpart on gradual types through existential lifting. Our only predicate in SF is type equality, whose existential lifting is type consistency:

**Definition 5.4 (Consistency).** \( \Xi; \Delta \vdash G_1 \sim G_2 \) if and only if \( \Sigma; \Delta \vdash T_1 = T_2 \) for some \( \Sigma \in C(\Xi) \), \( T_1 \in C(G_1) \).

For closed types we write \( G_1 \sim G_2 \). This definition uses a gradual type name store \( \Xi \), which binds type names to gradual types. Its concretization is the pointwise concretization:

\[
C(\cdot) = \emptyset \\
C(\Xi, \alpha := G) = \{ \Sigma, \alpha := T \mid \Sigma \in C(\Xi), T \in C(G) \}
\]

Note that because consistency is the consistent lifting of static type equality, which does impose well-formedness, consistency is only defined on well-formed types (i.e. \( \vdash X \sim X \) does not hold).

**Proposition 5.5 (Consistency, Inductively).** The inductive definition of type consistency given in Figure 3 is equivalent to Definition 5.4.

Again, observe that the resulting definition of consistency relates any two types that only differ in unknown type components, without any restriction. Also, because of explicit polymorphism, top-level constructors must match, so \( ? \to ? \) is not consistent with \( \forall X.? \to ? \). However, in line with gradual typing, both are consistent with \( ? \), as expected. Therefore GSF does not treat \( ? \to ? \) as a special “quasi-polymorphic” type, unlike System $F_G$ (Igarashi et al. 2017a). Rather, consistency in GSF coincides with that of CSA (Xie et al. 2018).

Lifting type functions such as $\text{dom}$ requires abstraction: a lifted function is the abstraction of the results of applying the static function to all the denoted static types (Garcia et al. 2016):

**Definition 5.6 (Consistent lifting of functions).** Let $F_n$ be a function of type $\text{Type}^n \to \text{Type}$. Its consistent lifting $\overline{F}_n^\#$, of type $\text{GType}^n \to \text{GType}$, is defined as: $\overline{F}_n^\#(G) = A(\{ F_n(T) \mid T \in C(G) \})$

The abstract interpretation framework allows us to prove the following definitions:

**Proposition 5.7 (Consistent Type Functions).** The definitions of $\text{dom}^\#$, $\text{cod}^\#$, $\text{inst}^\#$, and $\text{proj}^\#$ given in Fig. 3 are consistent liftings, as per Def. 5.6, of the corresponding functions from Fig. 1.

The gradual typing rules of GSF (Figure 3) are obtained by replacing type predicates and functions with their corresponding liftings. Note that in (Gapp), the premise \( \Xi; \Delta \vdash \text{dom}^\#(G_1) \sim G_2 \) is a compositional lifting of the corresponding premise in (Tapp), as justified by Garcia et al. (2016).

Of particular interest here is the fact that a term of unknown type can be optimistically treated as a polymorphic term and hence instantiated, yielding \( ? \) as the result type of the type application.
\( x \in \text{VAR}, X \in \text{TypeVar}, \alpha \in \text{Name} \quad \Xi \in \text{Name} \quad \Gamma \in \text{Type} \quad \Delta \subset \text{TypeVar} \quad \tau \in \text{Var} \rightarrow \text{GType} \\
G \quad ::= \quad B \mid G \rightarrow G \mid \forall X.G \mid G \times G \mid X | \alpha | ? \quad \text{(gradual types)} \\
t \quad ::= \quad b \mid \lambda x : G.t \mid AX.t \mid \langle t, t \rangle | x | t : G | op(\bar{t}) | t t | t [G] | \pi_1(t) \quad \text{(gradual terms)} \\
\Xi ; \Delta ; \Gamma \vdash t : G \quad \text{Well-typed terms} \\
\begin{align*}
(Gb) & \quad t y(b) = B & \Xi ; \Delta ; \Gamma + b : B \\
(GA) & \quad \Xi ; \Delta, X ; \Gamma \vdash t : G & \Xi ; \Delta ; \Gamma + \Delta.t : \forall X.G \\
(Gx) & \quad x : G \in \Gamma & \Xi ; \Delta ; \Gamma + x : G \\
(Gop) & \quad \Xi ; \Delta, \Gamma + \bar{t} : \Xi & t y(op) = \Xi \rightarrow G \\
(Ggap) & \quad \Xi ; \Delta, \Gamma + \bar{t} : \Xi & \Xi ; \Delta, \Gamma + \bar{t} : \Xi \\
(GappG) & \quad \Xi ; \Delta, \Gamma \vdash t : G & \Xi ; \Delta, \Gamma \vdash \bar{t} : \Xi \\
\end{align*} \\
\begin{align*}
\text{dom}^\# : \text{GType} \rightarrow \text{GType} & \quad \text{cod}^\# : \text{GType} \rightarrow \text{GType} & \quad \text{inst}^\# : \text{GType} \rightarrow \text{GType} & \quad \text{proj}^\# : \text{GType} \rightarrow \text{GType} \\
\text{dom}^\#(G_1 \rightarrow G_2) = G_1 & \quad \text{cod}^\#(G_1 \rightarrow G_2) = G_2 & \quad \text{inst}^\#(\forall X.G, G') = G[G'/X] & \quad \text{proj}^\#(G_1 \times G_2) = G_i \\
\text{dom}^\#(\text{?}) = ? & \quad \text{cod}^\#(\text{?}) = ? & \quad \text{inst}^\#(\text{?}, G') = ? & \quad \text{proj}^\#(\text{?}) = ? \\
\text{dom}^\#(G) \text{ undefined o/w} & \quad \text{cod}^\#(G) \text{ undefined o/w} & \quad \text{inst}^\#(G, G') \text{ undefined o/w} & \quad \text{proj}^\#(G) \text{ undefined o/w} \\
\Xi ; \Delta + G \rightarrow G \\
\begin{align*}
\vdash \Xi & \quad \Xi ; \Delta + B \sim B & \Xi ; \Delta + X \sim X \\
\vdash \Xi ; \Delta + G_1 \sim G_1' & \Xi ; \Delta + G_2 \sim G_2' \\
\Xi ; \Delta + \forall X.G_1 \sim \forall X.G_2 & \Xi ; \Delta + G_1 \times G_2 \sim G_1' \times G_2' \\
\vdash \Xi & \quad \alpha \in \text{dom}(\Xi) \\
\Xi ; \Delta + \forall \alpha \sim \forall \alpha & \Xi ; \Delta + G \sim \Xi(\alpha) & \Xi ; \Delta + G & \Xi ; \Delta + G & \Xi ; \Delta + G \sim ? \\
\Xi ; \Delta + \forall \alpha \sim \forall \alpha & \Xi ; \Delta + G \sim ? \\
\Xi ; \Delta + G \sim ? \\
\end{align*} \\
G \sqsubseteq G \quad \text{Type precision} \\
\begin{align*}
B \sqsubseteq B & \quad X \sqsubseteq X & G_1 \sqsubseteq G_1' & G_2 \sqsubseteq G_2' \\
G_1 \rightarrow G_2 \sqsubseteq G_1' \rightarrow G_2' & \forall X.G_1 \sqsubseteq \forall X.G_2 \\
G_1 \times G_2 \sqsubseteq G_1' \times G_2' & \alpha \sqsubseteq \alpha & G \sqsubseteq ? \\
\end{align*} \\
Fig. 3. GSF: Syntax and Static Semantics
We now turn to the dynamic semantics of GSF. As anticipated, this is where the complexity of AGT provides effective (though incomplete) guidance for the dynamics. In this section, we first brieﬂy recall the main ingredients of the AGT approach to dynamic semantics, namely evidence for consistent transitivity and consistent judgments and consistent transitivity. We then describe the reduction rules of GSF by treating evidence as an abstract datatype. This allows us to clarify a number of key operational aspects before turning in §7 to the details of the representation and operations of evidence that enable GSF to satisfy parametricity while adequately tracking type instantiations.

6 GSF: EVIDENCE-BASED DYNAMICS

As established by Siek and Taha (2006) in the context of simple types, we can prove that the GSF type system is equivalent to the SF type system on fully-static terms. We say that a gradual type is static if the unknown type does not occur in it, and a term is static if it is fully annotated with static types. Let ⊢_S denote the typing judgment of SF.5

Proposition 5.8 (Static equivalence for static terms). Let t be a static term and G a static type (G = T). We have ⊢_S t : T if and only if ⊢ t : T

The second important property of the static semantics of a gradual language is the static gradual guarantee, which states that typeability is monotonic with respect to precision (Siek et al. 2015a).

Type precision (Def. 5.1) extends to term precision. A term t is more precise than a term t’ if they both have the same structure and t is more precisely annotated than t’ (§B.4). The static gradual guarantee ensures that removing type annotations does not introduce type errors (or dually, that gradual type errors cannot be ﬁxed by making types more precise).

Proposition 5.9 (Static gradual guarantee). Let t and t’ be closed GSF terms such that t ⊏ t’ and ⊢ t : G. Then ⊢ t’ : G’ and G ⊑ G’.

6.1 Background: Evidence-Based Semantics for Gradual Languages

For obtaining the dynamic semantics of a gradual language, AGT augments a consistent judgment (such as consistency or consistent subtyping) with the evidence of why such a judgment holds. Then, reduction mimics proof reduction of the type preservation argument of the static language, combining evidences through steps of consistent transitivity, which either yield more precise evidence, or fail if the evidences to combine are incompatible. A failure of consistent transitivity corresponds to a cast error in a traditional cast calculus (Garcia et al. 2016).

Consider the gradual typing derivation of (λx : ?.x + 1) false. In the inner typing derivation of the function, the consistent judgment ? ~ Int supports the addition expression, and at the top-level, the judgment Bool ~ ? supports the application of the function to false. When two types are involved in a consistent judgment, we learn something about each of these types, namely the justification of why the judgment holds. This justification can be captured by a pair of gradual types, ε = (G_1, G_2), which are at least as precise as the types involved in the judgment (Garcia et al. 2016).6 Formally:

ε ⊢ G_1 ~ G_2 ⇐⇒ ε ⊑ A^2(((T_1, T_2) | T_1 ∈ C(G_1), T_2 ∈ C(G_2), T_1 = T_2))

---

5As usual, the propositions here are stated over closed terms, but are proven as corollaries of statements over open terms.
6We use blue color for evidence ε to enhance readability of the structure of terms in the next section and beyond.
i.e. if evidence \( \langle G_1', G_2' \rangle \) justifies the consistency judgment \( G_1 \sim G_2 \), then \( G_1' \subseteq G_1 \) and \( G_2' \subseteq G_2 \). For instance, by knowing that \(? \sim \text{Int}\) holds, we learn that the first type can only possibly be \text{Int}, while we do not learn anything new about the right-hand side, which is already fully static. Therefore the evidence of that judgment is \( \varepsilon_1 = \langle \text{Int}, \text{Int} \rangle \). Similarly, the evidence for the second judgment is \( \varepsilon_2 = \langle \text{Bool}, \text{Bool} \rangle \). Types in evidence can be gradual, e.g. \( \langle ? \rightarrow ?, ? \rightarrow ? \rangle \) justifies that \( (? \rightarrow ?) \sim ? \). Note that with the lifting of simple static type equality, both components of the evidence always coincide, so evidence can be represented as a single gradual type. However, type equality in SF is more subtle (§4), so the general presentation of evidence as pairs is required.

At runtime, reduction rules need to combine evidences in order to either justify or refute a use of transitivity in the type preservation argument. In our example, we need to combine \( \varepsilon_1 \) and \( \varepsilon_2 \) in order to (try to) obtain a justification for the transitive judgment, namely that \( \text{Bool} \sim \text{Int} \). The combination operation, called consistent transitivity \( \circ \), determines whether two evidences support the transitivity: here, \( \varepsilon_2 \circ \varepsilon_1 = \langle \text{Bool}, \text{Bool} \rangle \circ \langle \text{Int}, \text{Int} \rangle \) is undefined, so a runtime error is raised.

The evidence approach is very general and scales to disciplines where consistent judgments are not symmetric, involve more complex reasoning, and even other evidence combination operations (Garcia et al. 2016; Lehmann and Tanter 2017). All the definitions involved are justified by the abstract interpretation framework. Also, both type safety and the dynamic gradual guarantee become straightforward to prove. In particular, the dynamic gradual guarantee follows directly from the monotonicity (in precision) of consistent transitivity. In fact, the generality of the approach even admits evidence to range over other abstract domains; for instance, for gradual security typing with references, evidence is defined with label intervals, not gradual labels (Toro et al. 2018).

6.2 Reduction for GSF

In order to denote reduction of (evidence-augmented) gradual typing derivations, Garcia et al. (2016) use intrinsic terms as a notational device; while appropriate, the resulting description is fairly hard to comprehend and unusual, and it does implicitly involve a (presentational) transformation from source terms to their intrinsic representation.

In this work, we simplify the exposition by avoiding the use of intrinsic terms; instead, we rely on a type-directed, straightforward translation that inserts explicit ascriptions everywhere consistency is used—very much in the spirit of the coercion-based semantics of subtyping (Pierce 2002). For instance, the small program of §6.1 above, \((\lambda x : . x + 1) \text{false} \), is translated to:

\[
(\varepsilon_2 \rightarrow \text{Int}(\lambda x : . , (\varepsilon_1 x :: \text{Int}) :: ? \rightarrow \text{Int}) (\varepsilon_2 (\text{false} :: \text{Bool}) :: ?)
\]

where \( \varepsilon_G \) is the evidence of the reflexive judgment \( \mathcal{G} \sim \mathcal{G} \) (e.g. \( \varepsilon_{\text{Int}} \) supports \( \text{Int} \sim \text{Int} \)). Evidences \( \varepsilon_1 \) and \( \varepsilon_2 \) are the ones from §6.1.\(^7\)

Despite this translation, we do preserve the essence of the AGT dynamics approach in which evidence and consistent transitivity drive the runtime monitoring aspect of gradual typing. Furthermore, by making the translation explicitly ascribe all base values to their base type, we can present a uniform syntax and greatly simplify reduction rules compared to the original AGT exposition. This presentation also streamlines the proofs by reducing the number of cases to consider.

Figure 4 presents the syntax and semantics of GSF\(\varepsilon\), a simple variant of GSF in which all values are ascribed, and ascriptions carry evidence. Key changes with respect to Figure 3 are highlighted in gray. Here, we treat evidence as a pair of elements of an abstract datatype; we define its actual representation (and operations) in the next section.

\(^7\)Such initial evidences are computed by means of an interior function, given by the abstract interpretation framework (Garcia et al. 2016). The definition of interior (§C.2) and the type-preserving translation (§C.5) are direct.
Well-typed terms (for conciseness, s ranges over both t and u)

\( \Xi ; \Delta ; \Gamma \vdash s : G \)

- (E) \( ty(b) = B \quad \Xi ; \Delta ; \Gamma \vdash b : B \)
- (EA) \( \Xi ; \Delta , X \vdash t : G \quad \Xi ; \Delta \vdash G \)
- (Ex) \( x : G \in \Gamma \quad \Xi ; \Delta \vdash \Gamma \)
- (Eop) \( \Xi ; \Delta ; \Gamma \vdash \top : \Xi \)
- (Eapp) \( \Xi ; \Delta ; \Gamma \vdash t : \forall X . G \quad \Xi ; \Delta \vdash G' \)
- (EappG) \( \Xi ; \Delta ; \Gamma \vdash t : G \quad \Xi ; \Delta ; \Gamma \vdash G'[X] : G[G'/X] \)

Notion of reduction

\( \Xi \triangleright t \rightarrow \Xi \triangleright \text{error} \)

- (Rasc) \( \Xi \triangleright t \rightarrow \Xi \triangleright (\epsilon_1 u :: G_1) :: G_2 \)
- (Rop) \( \Xi \triangleright \epsilon_2 u :: G \)
- (Rapp) \( \Xi \triangleright (\epsilon_1 (\lambda x : G_{11} . t) :: G_1) \rightarrow t' \)
- (Rproji) \( \Xi \triangleright t \rightarrow \pi_i (\epsilon_i (u_1 , u_2) :: G_{11} \times G_{12}) \)
- (RappG) \( \Xi \triangleright (\epsilon \lambda X . t : \forall X . G) [G'] \rightarrow \Xi \triangleright t \rightarrow \Xi \triangleright \text{error} \)

Evaluation frames and reduction

\( \epsilon \square :: G | op(\bar{\epsilon}, \square, \bar{t}) | \square t | \epsilon \square | G \)

- (Rf) \( \Xi \triangleright t \rightarrow \Xi \triangleright f[t] \rightarrow \Xi \triangleright f[t'] \)
- (Rerr) \( \Xi \triangleright t \rightarrow \Xi \triangleright \text{error} \)

Fig. 4. GSF\( \epsilon \): Syntax, Static and Dynamic Semantics
types to match exactly; the translation inserts ascriptions to ensure that top-level constructors match in every elimination form.

The notion of reduction for GSF terms deals with evidence propagation and composition with consistent transitivity. Rule (Rasc) specifies how an ascription around an ascribed value reduces to a single value if consistent transitivity holds: \( \varepsilon_1 \) justifies that \( G_u \sim G_1 \), where \( G_u \) is the type of the underlying simple value \( u \), and \( \varepsilon_2 \) is evidence that \( G_1 \sim G_2 \). The composition via consistent transitivity, if defined, justifies that \( G_u \sim G_2 \); if undefined, reduction steps to error. Rule (Rop) simply strips the underlying simple values, applies the primitive operation, and then wraps the result in an ascription, using a canonical base evidence \( \varepsilon_B \) (which trivially justifies that \( B \sim B \)). Rule (Rapp) combines the evidence from the argument value \( \varepsilon_2 \) with the domain evidence of the function value \( \text{dom}(\varepsilon_1) \) in an attempt to transitivity justify that \( G_u \sim G_{11} \). Failure to justify that judgment, as in our example in §6.1, produces error. The return value is ascribed to the expected return type, using the codomain evidence of the function \( \text{cod}(\varepsilon_1) \). Rule (Rpair) produces a pair value when the subterms of a pair have been reduced to values themselves, using the product operator on evidences \( \varepsilon_1 \times \varepsilon_2 \). This rule is necessary to enforce a uniform presentation of all values as ascribed values, which simplifies technicalities. Dually, Rule (Rproji) extracts a component of a pair and ascribes it to the projected type, using the corresponding evidence obtained with \( p_1(\varepsilon) \).

Apart from the presentational details, the above rules are standard for an evidence-based reduction semantics. Rule (RappG) is the rule that specifically deals with parametric polymorphism, reducing a type application. This is where most of the complexity of gradual parametricity concentrates. Observe that there are two ascriptions in the produced term:

- The inner ascription (to \( G[\alpha/X] \)) is for the body of the polymorphic term, asserting that substituting a fresh type name \( \alpha \) for the type variable \( X \) preserves typing. The associated evidence \( \varepsilon[\hat{\alpha}] \) is the result of instantiating \( \varepsilon \) (which justifies that the actual type of \( \Lambda X.t \) is consistent with \( \forall X.G \)) with the fresh type name, hence justifying that the body after substitution is consistent with \( G[\alpha/X] \).

- The outer ascription asserts that \( G[\alpha/X] \) is consistent with \( G[G'/X] \), witnessed by evidence \( \varepsilon_{out} \). This evidence plays a key role in avoiding unjustified failures as described in §2.4. We define \( \varepsilon_{out} \) in §7.2 below, once the representation of evidence is introduced.

The use of \( \hat{\alpha} \) is a technicality: because so far we treat evidence as an abstract datatype from an as-yet-unspecified domain, say pairs of ETYYPE, we cannot directly use gradual types (GTYPE) inside evidences. The connection between GTYPE and ETYYPE is specified by lifting operations, \( \text{lift}_G : \text{GTYPE} \rightarrow \text{ETYYPE} \) and \( \text{unlift} : \text{ETYYPE} \rightarrow \text{GTYPE} \). Because type names have meaning related to a store, the lifting is parameterized by the store \( \Sigma \). Term substitution is mostly standard: it uses \( \text{unlift} \) to recover \( \alpha \), and is extended to substitute recursively in evidences. Substitution in evidence, also triggered by evidence instantiation, is simply component-wise substitution on evidence types.

Finally, the evaluation frames and associated reduction rules in Figure 4 are straightforward; in particular (Rerr) and (Rferr) propagate error to the top-level.

7 EVIDENCE FOR GRADUAL PARAMETRICITY

We now turn to the actual representation of evidence. We first explain in §7.1 why the standard representation of evidence as pair of gradual types is insufficient for gradual parametricity. We then introduce the refined representation of evidence to enforce parametricity (§7.2), and basic properties of the language. Richer properties of GSF are discussed in §8, §9 and §10.

---

8We use \( p_1(\varepsilon) \) to avoid confusion with \( \pi_1(\varepsilon) \), which refers to the first projection of evidence (itself a metalanguage pair).

9In standard AGT (Garcia et al. 2016) the lifting is simply the identity, i.e. ETYYPE = GTYPE.
7.1 Simple Evidence, and Why It Fails

In standard AGT (Garcia et al. 2016), evidence is simply represented as a pair of gradual types, \( i.e. \) \( \text{ETYPE} = \text{GTYPE} \). Consistent transitivity is defined through the abstract interpretation framework. The definition for simple types is as follows: \( (\varepsilon \vdash J) \) means \( \varepsilon \) supports the consistent judgment \( J \):

**Definition 7.1 (Consistent transitivity).** Suppose \( \varepsilon_{ab} \vdash G_a \sim G_b \) and \( \varepsilon_{bc} \vdash G_b \sim G_c \). Evidence for consistent transitivity is deduced as \( (\varepsilon_{ab} \circ \varepsilon_{bc}) \vdash G_a \sim G_c \), where:

\[
\langle G_1, G_{21} \rangle \circ \langle G_{22}, G_3 \rangle = A^2(\{\langle T_1, T_3 \rangle \in C(G_1) \times C(G_3) | \exists T_2 \in C(G_{21}) \cap C(G_{22}), T_1 = T_2 \land T_2 = T_3 \})
\]

In words, if defined, the evidence that supports the transitive judgment is obtained by abstracting over the pairs of static types denoted by the outer evidence types \( (G_1 \text{ and } G_3) \text{ such that } \) they are connected through a static type common to both middle evidence types \( (G_{21} \text{ and } G_{22}) \). This definition can be proven to be equivalent to an inductive definition that proceeds in a syntax-directed manner on the structure of types (Garcia et al. 2016).

Consistent transitivity satisfies some important properties. First, it is associative. Second, the resulting evidence is more precise than the outer evidence types, reflecting that during evaluation, typing justification only gets more precise (or fails). Violating this property breaks type safety. The third property is key for establishing the dynamic gradual guarantee (Garcia et al. 2016).

**Lemma 7.2.** (Properties of consistent transitivity).

(a) Associativity. \( (\varepsilon_1 \circ \varepsilon_2) \circ \varepsilon_3 = \varepsilon_1 \circ (\varepsilon_2 \circ \varepsilon_3) \), or both are undefined.

(b) Optimality. If \( \varepsilon = \varepsilon_1 \circ \varepsilon_2 \) is defined, then \( \pi_1(\varepsilon) \subseteq \pi_1(\varepsilon_1) \) and \( \pi_2(\varepsilon) \subseteq \pi_2(\varepsilon_2) \).

(c) Monotonicity. If \( \varepsilon_1 \subseteq \varepsilon_1' \) and \( \varepsilon_2 \subseteq \varepsilon_2' \) and \( \varepsilon_1 \circ \varepsilon_2 \) is defined, then \( \varepsilon_1 \circ \varepsilon_2 \subseteq \varepsilon_1' \circ \varepsilon_2' \).

Unfortunately, adopting gradual types for evidence types and simply extending the consistent transitivity definition to deal with GSF types and consistency judgments yields a gradual language that breaks parametricity.\(^{10}\) To illustrate, consider this simple program:

\[
\begin{array}{l}
\langle A X. \langle \lambda x : X. \text{let} y : ? = x \text{ in} \text{let} z : ? = y \text{ in} z + 1 \rangle \rangle \ [\text{Int}] 1
\end{array}
\]

The function is not parametric because it ends up adding 1 to its argument, although it does so after two intermediate bindings, typed as \( ? \). Without further precaution, the parametricity violation of this program would not be detected at runtime. Assume that the type application generates the fresh name \( \alpha \), bound to \text{Int} in the store. For justifying that \( x \) can flow to \( y \) (the let-binding is equivalent to a function application), we need evidence for \( \text{Int} \sim ? \) by consistent transitivity between the evidence \( \{(\text{Int}, \alpha)\} \), which justifies \( \text{Int} \sim \alpha \)\(^{11}\), and \( \langle \alpha, \alpha \rangle \), which justifies \( \alpha \sim ? \).\(^{12}\) Using the definition of consistent transitivity (Def. 7.1), \( \langle \text{Int}, \alpha \rangle \circ \langle \alpha, \alpha \rangle = \langle \text{Int}, \alpha \rangle \). Similarly, for justifying the flow of \( y \) to \( z \), the previous evidence must be combined with \( \langle ?, ? \rangle \), which justifies \( ? \sim ? \). By Def. 7.1, \( \langle \text{Int}, \alpha \rangle \circ \langle ?, ? \rangle = A^2(\{\langle \text{Int}, \alpha \rangle, \langle \text{Int}, \alpha \rangle\}) = \langle \text{Int}, ? \rangle \). This evidence can subsequently be used to produce evidence to justify that the addition is well-typed, since \( \langle \text{Int}, ? \rangle \circ \langle \text{Int}, \text{Int} \rangle = \langle \text{Int}, \text{Int} \rangle \). Therefore the program produces 2, without errors: parametricity is violated.

### 7.2 Refining Evidence

For gradual parametricity, evidence must do more than just ensure type safety. It needs to safeguard the sealing that type variables are meant to represent, also taking care of unsealing as necessary. First of all, we need to define evidence to adequately represent consistency judgments of GSF.

---

\(^{10}\)The obtained language is type safe, and satisfies the dynamic gradual guarantee. This novel design could make sense to gradualize impure polymorphic languages, which do not enforce parametricity. Exploring this perspective is future work.

\(^{11}\)Note that conversely to the simply-typed setting, both components of evidence are not necessarily equal, as in this case.

\(^{12}\)This evidence is obtained by substituting \( \alpha \) for \( X \) in the initial evidence \( \langle X, X \rangle \) for \( X \sim ? \).
Evidence Types. We define evidence types, \( E \in \text{ETYPE} \), to be an enriched version of gradual types:

\[
E ::= B \mid E \to E \mid \forall X.E \mid E \times E \mid \alpha^{E} \mid X \mid ?
\]

SF equality judgments, and hence GSF consistency judgments, are relative to a store. It is therefore not enough to use type names in evidence: we need to keep track of their associated types in the store. An evidence type name \( \alpha^{E} \) therefore captures the type associated to the type name \( \alpha \) through the store. For instance, evidence that a variable has a polymorphic type \( X \) is initially \( \langle X, X \rangle \). When \( X \) is instantiated, say to \( \text{Int} \), and a fresh type name \( \alpha \) is introduced, the evidence becomes \( \langle \alpha^{\text{Int}}, \alpha^{\text{Int}} \rangle \).

An evidence type name does not only record the end type to which it is bound, but the whole path. For instance, \( \alpha^{\text{Int}} \) is a valid evidence type name that embeds the fact that \( \alpha \) is bound to \( \beta \), which is itself bound to \( \text{Int} \).

Note that as a program reduces, evidence can not only become more precise than statically-used types, but also than the global store. For instance, it can be the case that \( \alpha ::= \ ? \) in the global store \( \Xi \), but that locally, the evidence for \( \alpha \) has gotten more precise, such as \( \alpha^{\text{Int}} \). Formally, a type name is enriched with its transitive bindings in the store, \( \text{lift}_{\Xi}(\alpha) = \alpha^{\text{lift}_{\Xi}(\Xi(\alpha))} \). Unlifting simply forgets the additional information: \( \text{unlift}_{\Xi}(\alpha^{E}) = \alpha \). In all other cases, both operations recur structurally.

It is crucial to understand the intuition behind the position of type names in a given evidence. The position of \( \alpha^{E} \) in an evidence can correspond to a sealing, an unsealing, or neither. If \( \alpha^{E} \) is only on the right side, e.g. \( \langle \text{Int}, \alpha^{\text{Int}} \rangle \), then the evidence is a sealing (here, of \( \text{Int} \) with \( \alpha \)). Dually, if \( \alpha^{E} \) is only on the left side, e.g. \( \langle \alpha^{\text{Int}}, \text{Int} \rangle \), the evidence is an unsealing (here, of \( \text{Int} \) from \( \alpha \)). Sealing and unsealing evidences arise through reduction, as will be illustrated later in this section.

Consistent Transitivity. With this syntactic enrichment, consistent transitivity can be strengthened to account for sealing and unsealing, ensuring parametricity. Consistent transitivity is defined inductively; selected rules are presented in Figure 5.

Rule (unsl) specifies that when a sealing and an unsealing of the same type name meet in the middle positions of a consistent transitivity step, the type name can be eliminated in order to calculate the resulting evidence. For instance, \( \langle \text{Int}, \alpha^{\text{Int}} \rangle \circ \langle \alpha^{\text{Int}}, \text{Int} \rangle \circ \langle ?, ?, \text{Int} \rangle = \langle \text{Int}, \text{Int} \rangle \).

As shown in §7.1, it is important for consistent transitivity to not lose precision when combining an evidence with an unknown evidence. To this end, rule (identL) in Fig. 5 preserves the left evidence. Going back to the example of §7.1, we now have \( \langle \text{Int}, \alpha^{\text{Int}} \rangle \circ \langle ?, ?, \text{Int} \rangle = \langle \text{Int}, \alpha^{\text{Int}} \rangle \), instead of \( \langle \text{Int}, ?, \text{Int} \rangle \). Because \( \langle \text{Int}, \alpha^{\text{Int}} \rangle \circ \langle \text{Int}, \text{Int} \rangle \) is undefined, reduction steps to \textbf{error} as desired.

Rule (seall) shows that when an evidence is combined with a sealing, the resulting evidence is also a sealing. This sealing can be more precise, e.g. \( \langle \text{Int}, \text{Int} \rangle \circ \langle ?, \alpha^{\text{Int}} \rangle = \langle \text{Int}, \alpha^{\text{Int}} \rangle \).
Figure 5 only shows one structurally-recursive rule, corresponding to the function case (func); consistent transitivity is computed recursively with the domain and codomain evidences. To combine a function evidence with unknown evidence, the unknown evidence is first “expanded” to match the type constructor (func?L). There are similar rules for the other type constructors. Also, there are symmetric variants of the above rules—such as (identR) and (sealR)—in which every evidence and every evidence type is swapped. The complete definition is provided in §C.3.

Importantly, this refined definition of consistent transitivity preserves associativity and optimality, based on a natural notion of precision for evidence types (§C.1). It does however break monotonicity, and hence the dynamic gradual guarantee. In §9, we give a semantic argument establishing that the dynamic gradual guarantee is fundamentally incompatible with parametricity anyway, independently of this refinement.

**Outer Evidence.** The reduction rule of a type application (RappG) produces an outer evidence \( \epsilon_{\text{out}} \) that justifies that \( G[\alpha/\lambda] \) is consistent with \( G[G'/\lambda] \). The precise definition of this evidence is delicate, addressing a subtle tension between the precision required for justifying unsealing when possible, and the imprecision required for parametricity.

\[
\epsilon_{\text{out}} \triangleq \langle E_\ast[\alpha^E], E_\ast[E'] \rangle \quad \text{where } E_\ast = \text{lift}_\zeta(\text{unlift}(\pi_2(\epsilon))), \alpha^E = \text{lift}_\zeta(\alpha), E' = \text{lift}_\zeta(G')
\]

In this definition, \( \epsilon, \alpha, G', \Xi, \) and \( \Xi' \) come from rule (RappG). Determining \( E_\ast \) is the key challenge. The second evidence type of \( \epsilon \) refines \( \forall X.G \) by exploiting the fact that the underlying polymorphic value \( \Lambda X.t \) is consistent with it; this extra precision is crucial for unsealing. The roundtrip unlift/lift “resets” the sealing information of evidence type names to that contained in the store; this relaxation is crucial for parametricity (to prove the compositionality lemma—§8).

Note that \( \epsilon_{\text{out}} \) will never cause a runtime error when combined with the resulting evidence of the parametric term result, because both are necessarily related by precision.

**Illustration.** The following reduction trace illustrates all the important aspects of reduction:

\[
\begin{align*}
&\text{(RappG)} \quad \text{initial evidence}\quad \langle (\alpha^\text{Int} \to \alpha^\text{Int}), \text{Int} \to \text{Int} \rangle \langle \epsilon_{\text{out}} \rangle \\
&\quad \langle \alpha^\text{Int} \to \alpha^\text{Int}, \text{Int} \to \text{Int} \rangle \langle \epsilon_{\text{out}} \rangle \\
&\quad \langle (\alpha^\text{Int} \to \alpha^\text{Int}, \text{Int} \to \text{Int}), (\lambda x : \alpha. x) : \alpha \to ? \rangle \\
&\quad \langle \text{Int} \to \text{Int} \rangle \langle \lambda x : \alpha. x : \alpha \to ? \rangle \\
&\quad \langle \text{Int} \to \text{Int} \rangle \langle \lambda x : \alpha. x : \alpha \to ? \rangle \\
&\quad \langle (\text{Int} \to \text{Int}), (\lambda x : \alpha. x) : \alpha \to ? \rangle \\
&\quad \text{argument is sealed}
\end{align*}
\]

Crucially, the initial evidence of the identity function is fully precise, even though it is ascribed an imprecise type. Consequently, in the first reduction step above, \( \epsilon_{\text{out}} \) is calculated as:

\[
\epsilon_{\text{out}} \triangleq \langle E_\ast[\alpha^E], E_\ast[E'] \rangle = \langle (\forall X. X \to ?), (\forall X. X \to ?) \text{Int} \rangle = \langle \alpha^\text{Int} \to \alpha^\text{Int}, \text{Int} \to \text{Int} \rangle
\]

The application step (Rapp) then gives rise to sealing and unsealing evidences after deconstructing \( \epsilon_{\text{out}} \): the inner evidence \( \langle \text{Int}, \alpha^\text{Int} \rangle \) seals the number 1 at type \( \alpha \), while the outer evidence \( \langle \alpha^\text{Int}, \text{Int} \rangle \) allows the subsequent unsealing in the ascription step (Rasc). As a result, the ascribed identity function yields usable values, because the outer evidence subsequently takes care of unsealing. This addresses the excess of failure reported with \( \lambda B \) and System F\(_C\) in §2.4. Note that if the function were not intrinsically precise on its return type, e.g. \( \Lambda X. \lambda x : X. (x :: ?) \), then initial evidence would likewise be imprecise, and deconstructing \( \epsilon_{\text{out}} \) would not justify unsealing the result anymore.

\[\text{For instance, consider } \langle \text{Int}, \alpha^\text{Int} \rangle \not\subseteq \langle \text{Int}, \alpha^\text{Int} \rangle \text{ and } \langle \alpha^\text{Int}, \text{Int} \rangle \not\subseteq \langle ?, ? \rangle. \text{ By consistent transitivity, } \langle \alpha^\text{Int}, \text{Int} \rangle \circ \langle \alpha^\text{Int}, \text{Int} \rangle = \langle \text{Int}, \text{Int} \rangle \text{ (rule unsl)}, \text{ and } \langle \text{Int}, \alpha^\text{Int} \rangle \circ \langle ?, ? \rangle = \langle \text{Int}, \alpha^\text{Int} \rangle \text{ (rule idl)}, \text{ but } \langle \text{Int}, \text{Int} \rangle \not\subseteq \langle \text{Int}, \alpha^\text{Int} \rangle.\]

Matias Toro, Elizabeth Labrada, and Éric Tanter
7.3 Basic Properties of GSF Evaluation

The runtime semantics of a GSF term are given by first translating the term to GSFℓ (noted \( \vdash t \leadsto \varepsilon \vdash t : G \)) and then reducing the GSFℓ term. We write \( t \Downarrow \Xi \vdash v \) (resp. \( t \Downarrow \varepsilon \downarrow \text{error} \)) if \( t \leadsto \varepsilon \vdash t : G \) and \( \Delta \cdot t \varepsilon \leadsto \varepsilon \vdash v \) (resp. \( \Delta \cdot t \varepsilon \leadsto \varepsilon \vdash \varepsilon \downarrow \text{error} \)) for some resulting store \( \Xi \). We write \( \Xi \vdash v : G \) for \( \Xi ; \cdot \vdash v : G \). We write \( t \Downarrow \) if the translation of \( t \) diverges, and \( t \Downarrow v \) when the store is irrelevant.

The properties of GSF follow from the same properties of GSFℓ, expressed using the small-step reduction relation, due to the fact that the translation \( \leadsto \) preserves typing (§C.5). In particular, GSF terms do not get stuck, although they might produce error or diverge:

**Proposition 7.3 (Type Safety).** If \( \vdash t : G \) then either \( t \Downarrow \Xi \vdash v \) with \( \Xi \vdash v : G \), \( t \Downarrow \varepsilon \downarrow \text{error} \), or \( t \Downarrow \).

Proposition 5.8 established that GSF typing coincides with SF typing on static terms. A similar result holds considering the dynamic semantics. In particular, static GSF terms never produce error:

**Proposition 7.4 (Static terms do not fail).** Let \( t \) be a static term. If \( \vdash t : T \) then \( \neg(t \Downarrow \varepsilon \downarrow \text{error}) \).

This result follows from the fact that all evidences in a static program are static, hence never gain precision; the initial type checking ensures that combination through transitivity never fails. As we will see in §10, a static term is also guaranteed to terminate.

8 GSF: Parametricity

We establish parametricity for GSF by proving parametricity for GSFℓ. Specifically, we define a step-indexed logical relation for GSFℓ terms, closely following the relation for \( \lambda B \) (Ahmed et al. 2017). In the following, we only go briefly over the definition of the relation (Figure 6), and focus on the few differences with the \( \lambda B \) relation, essentially dealing with evidences.

The relation is defined on tuples \((W, t_1, t_2)\) that denote two related terms \( t_1, t_2 \) in a world \( W \). A world is composed of a step index \( j \), two stores \( \Xi_1 \) and \( \Xi_2 \) used to typecheck and evaluate the related terms, and a mapping \( \kappa \), which maps type names to relations \( R \), used to relate sealed values. The components of a world are accessed through a dot notation, e.g. \( W.j \) for the step index.

The interpretations of values, terms, stores, name environments, and type environments are mutually defined, using the auxiliary definitions at the bottom of Figure 6. As usual, the value and term interpretations are indexed by a type and a type substitution \( \rho \). We use \( \text{Atom}_n[G_1, G_2] \) to denote a set of pair of terms of type \( G_1 \) and \( G_2 \), and worlds with a step index less than \( n \). We write \( \text{Atom}_n^{\text{val}}[G_1, G_2] \) to restrict that set to values, and \( \text{Atom}_n[G] \) to denote a set of terms of the same type after substitution. The \( \text{Atom}_n^{\text{val}}[G] \) variant is similar to \( \text{Atom}_n^{\text{val}}[G_1, G_2] \) but restricts the set to values that have, after substitution, equally precise evidences (the equality is after unlifting because two sealed values may be related under different instantiations). \( \text{Rel}_n[G_1, G_2] \) defines the set of relations of values of type \( G_1 \) and \( G_2 \). We use \( [R]_n \) and \( [\kappa]_n \) to restrict the step index of the worlds to less than \( n \). Finally, \( \kappa' \geq \kappa \) specifies that \( \kappa' \) is a future relation mapping of \( \kappa \) (and extension), and similarly \( W' \geq W \) expresses that \( W' \) is a future world of \( W \). The \( \downarrow \) operator lowers the step index of a world by 1.

The logical interpretation of terms of a given type enforces a “termination-sensitive” view of parametricity: if the first term yields a value, the second must produce a related value at that type; if the first term fails, so must the second. Note that \( \text{Atom}_n^{\text{val}}[G] \) requires the second component of the evidence of each value to have the same precision in order to enforce such sensitivity. Indeed, if one is allowed to be more precise than the other, then when later combined in the same context, the more precise value may induce failure while the other does not.

Two base values are related if they are equal. Two functions are related if their application to related values yields related results. Two type abstractions are related if given any two types
\(\forall \rho \exists \alpha : \exists \beta \in \text{World}\)
and any relation between them, the instantiated terms (without their unsealing evidence) are also related in a world extended \((\Xi)\) with \(\alpha\), the two instantiation types \(G_1\) and \(G_2\) and the chosen relation \(R\) between sealed values. Note that the step index of this extended world is decreased by one, because we take a reduction step. Two pairs are related if their components are pointwise related. Two sealed values are related at a type name \(\alpha\) if, after unsealing, the resulting values are in the relation corresponding to \(\alpha\) in the current world, \(W, \kappa(\alpha)\).

Finally, two values are related at type \(\gamma\) if they are related at the least-precise type with the same top-level constructor as the second component of the evidence, \(\text{const}(\pi_2(\epsilon_i))\).\(^{14}\) The intuition is that to be able to relate these unknown values we must take a step towards relating their actual content; evidence necessarily captures at least the top-level constructor (e.g. if a value is a function, the second evidence type is no less precise than \(\rightarrow \), i.e. \(\text{const}(E_1 \rightarrow E_2)\)).

The logical relation is well-founded for two reasons: (i) in the \(\rightarrow\) case, \(\text{const}(\pi_2(\epsilon_i))\) cannot itself be \(\rightarrow\), as just explained; (ii) in each other recursive cases, the step index is lowered: for functions and pairs, the relation is between reducible expressions (applications, projections) that either take a step or fail; for type abstractions, the relation is with respect to a world whose indexed is lowered.

The interpretations of stores, type name environments and type environments are straightforward (Figure 6). The logical relation allows us to define logical approximation, whose symmetric extension is logical equivalence. Any well-typed GSFCr term is related to itself at its type:

**Theorem 8.1 (Fundamental Property).** If \(\Xi; \Delta; \Gamma \vdash t : G\) then \(\Xi; \Delta; \Gamma \vdash t \leq t : G\).

As standard, the proof of the fundamental property uses compatibility lemmas for each term constructor and the compositionality lemma (§E.2). Almost every compatibility lemma relies on the fact that the ascription of two related values yield related terms.

**Lemma 8.2 (Ascriptions Preserve Relations).** If \(W, v_1, v_2 \in \mathcal{V}_\rho[G], \epsilon \vdash \Xi; \Delta \vdash G \rightarrow G', W \in S[\Xi], \text{and } (W, \rho) \in D[\Delta],\) then \((W, \rho_1(\epsilon)v_1 :: \rho(G'), \rho_2(\epsilon)v_2 :: \rho(G')) \in T_\rho[G']\).

Note that type substitution on evidences takes as parameter the corresponding store: \(\rho_1(\epsilon)\) is syntactic sugar for \(\rho(W, \Xi, _i, \epsilon)\), lifting each substituted type name in the process, e.g. if \(\rho(X) = \alpha, W.\Xi_1(\alpha) = \text{Int}\), and \(W.\Xi_2(\alpha) = \text{Bool}\), then \(\rho_1((X, X)) = ⟨\alpha^\text{Int}, \alpha^\text{Int}\rangle\), and \(\rho_2((X, X)) = ⟨\alpha^\text{Bool}, \alpha^\text{Bool}\rangle\).

## 9 PARAMETRICITY VS. DYNAMIC GRADUAL GUARANTEE

We now turn to the dynamic gradual guarantee (Siek et al. 2015a). In a big-step setting, this guarantee essentially says that if \(\vdash t : G \text{ and } G \vdash v\), then for any \(t'\) such that \(t \leq t'\), we have \(t' \parallel v'\) for some \(v'\) such that \(v \equiv v'\). We show that parametricity as defined in §8 is however incompatible with this guarantee. First, we can prove the following lemma:

**Lemma 9.1.** For any \(\vdash v : \gamma \text{ and } G, \) we have \((\lambda\mathbf{x}.\lambda\mathbf{x} : \gamma : x :: X)[G] \parallel v\) \textbf{error}.

**Proof.** Let \(v' = (\lambda\mathbf{x}.\lambda\mathbf{x} : \gamma : x :: X), v' \rightarrow v' \rightarrow v' : \forall X.\gamma \rightarrow X\), and \(v' \vdash v' \equiv v' : \gamma\).

By the fundamental property (Th. 8.1), \(v' \leq v' \vdash v' : \forall X.\gamma \rightarrow X\) so for any \(W_0 \in S[\gamma], (W_0, v', v') \in T_0[\forall X.\gamma \rightarrow X]\). Because \(v'\) is a value, \((W_0, v', v') \in \mathcal{V}_0[\forall X.\gamma \rightarrow X]\). By reduction, \(\vdash v' \in \mathcal{V}_0[G_i] \rightarrow \mathcal{V}'_i \leftarrow (\lambda\mathbf{x} : \gamma \rightarrow \alpha) \leftarrow \alpha\). We can instantiate the definition of \(\mathcal{V}_0[\forall X.\gamma \rightarrow X]\) with \(W_0, \alpha = G_1\) and \(G_2\) structurally different (and different from \(\gamma\)), some \(R \in \text{REL}_{W_0, G}[G_1, G_2]\), \(v_1, v_2, \epsilon'_1\) and \(\epsilon'_2\), then we have that \((W_1, \epsilon_1, v_1) \in \mathcal{V}_{\mathbf{X} \mapsto R}[\gamma \rightarrow X]\), where \(W_i = (\downarrow (W_0 \boxslash (\alpha, G_1, G_2, R))\). As \(v_1\) and \(v_2\) are values, \((W_1, \epsilon_1, v_1) \in \mathcal{V}_{\mathbf{X} \mapsto R}[\gamma \rightarrow X]\). Also, by associativity of consistent transitivity, the reduction of \(\mathcal{V}'_1 \rightarrow (\epsilon'_1v_1 : \gamma \rightarrow G_1) \rightarrow v_1\) is equivalent to that of \(\mathcal{V}'_1 \rightarrow \text{cod}(\epsilon'_1)(v_1 (\text{dom}(\epsilon'_1)v_2 : ?)) : G_i\).

\(^{14}\)\text{const} extracts the top-level constructor of an evidence type, e.g. \(\text{const}(E_1 \rightarrow E_2) = \equiv \rightarrow \) and \(\text{const}(\forall X. E) = \forall X.\gamma\).
By the fundamental property (Th. 8.1) we know that \( \vdash \nu_1 < \nu_2 : \gamma \); we can instantiate this definition with some \( W_2 \geq W_1 \), and we have that \( (W_2, \nu_1, \nu_2) \in \mathcal{J}_0[\gamma] \). Since \( \nu_2 \) is a value, \( (W_2, \nu_1, \nu_2) \in \mathcal{V}_{X \rightarrow \alpha}[\gamma] \). By the ascription lemma (8.2), \( (W_2, \text{dom}(\epsilon'_i) \nu_2 \vdash \gamma, \text{dom}(\epsilon'_i) \nu_2 \vdash \gamma) \in \mathcal{J}_p[\gamma] \). If \( \text{dom}(\epsilon'_i) \nu_2 \vdash \gamma \) reduces to \text{error} then the result follows immediately. Otherwise, \( \exists \epsilon'_i \cdot \text{dom}(\epsilon'_i) \nu_2 \vdash \gamma \mapsto \gamma \exists \epsilon'_i \cdot \nu'_2 \), and \( (W_2, \nu''_1, \nu''_2) \in \mathcal{V}_p[\gamma] \), where \( W_2 = W_2, \) and some \( \nu''_1 \) and \( \nu''_2 \). We can instantiate the definition of \( \mathcal{V}_{X \rightarrow \alpha}[\gamma] \) with \( W_2, \nu''_1 \) and \( \nu''_2 \), obtaining that \( (W_3, \nu_1 \nu'_2, \nu_2 \nu'_2) \in \mathcal{J}_{X \rightarrow \alpha}[X] \). We then proceed by contradiction. Suppose that \( \exists \epsilon'_i \cdot \nu_1 \nu'_2 \mapsto \epsilon \exists \epsilon'_i \cdot \nu'_2 \) (for a big-enough step index). If \( \nu'_i = \epsilon'_i u : \gamma \), then by evaluation \( \nu'_i = \epsilon'_i u : \alpha \), for some \( \epsilon'_i u \). But by definition of \( \mathcal{V}_{X \rightarrow \alpha}[X] \), it must be the case that for some \( W_4 \geq W_3, (W_4, \epsilon'_i u : G_1, \epsilon'_i u : G_2) \in R \), which is impossible because \( u \) cannot be ascribed to structurally different types \( G_1 \) and \( G_2 \). Therefore \( \nu_1 \nu'_2 \) cannot reduce to a value, and hence the term \( \nu_1 [G] \nu'_2 \) cannot reduce to a value either. Because \( \nu_1 \) is non-diverging, its application must produce \text{error}.

Consequently, the dynamic gradual guarantee is violated:

**Corollary 9.2.** There exist \( t_1 : G \) and \( t_2 : \gamma \) such that \( t_1 \Downarrow \nu \) and \( t_2 \Downarrow \text{error} \).

**Proof.** Let \( \text{id}_X \triangleq \Lambda X. \lambda x : X. x :: X \), and \( \text{id}_G \triangleq \Lambda X. \lambda x : \gamma. x :: X \). By definition of precision, we have \( \text{id}_X \subseteq \text{id}_G \). Let \( \vdash \nu : G \) and \( \vdash \nu' : \gamma \), such that \( \nu \subseteq \nu' \). Pose \( t_1 \triangleq \text{id}_X \gamma \nu \) and \( t_2 \triangleq \text{id}_G \nu' \). By definition of precision, we have \( t_1 \subseteq t_2 \). By evaluation, \( t_1 \Downarrow \nu \). But by Lemma 9.1, \( t_2 \Downarrow \text{error} \).

Interestingly, Lemma 9.1 holds irrespective of the actual choices for representing evidence in GSF. The key element is the (standard) logical interpretation of \( V X G \). Therefore the incompatibility described here does not apply only to GSF: in fact, we have been able to prove that Lemma 9.1 also holds in \( \lambda B \) (Ahmed et al. 2017), whose notion of parametricity is essentially the same as GSF.

By sticking to this standard notion of parametricity, one way to accommodate the dynamic gradual guarantee is to change the definition of precision, as done by Igarashi et al. (2017a) (denying that \( t_1 \subseteq t_2 \) in the proof of Corollary 9.2). We believe this is questionable, because precision is a syntactic and intuitive notion describing "how static a type is", and replacing parts of a type with \( ? \) is clearly making it "less static" (recall §2.3). Dually, if one sticks to the natural notion of precision, as adopted by both GSF and CSA, and justified by the AGT interpretation, reconciliation might come from considering other forms of parametricity, or perhaps less flexible gradual language designs (Devriese et al. 2018). Currently, it seems that the incompatibility of the dynamic gradual guarantee with parametricity has to be understood, in conjunction with a similar observation regarding noninterference (Toro et al. 2018), as hinting towards further refined criteria for semantically-rich gradual typing. In particular, weaker forms of the dynamic gradual guarantee might still be useful, as explored next.

## 10 GRADUAL FREE THEOREMS IN GSF

The parametricity logical relation (§8) allows us to define notions of logical approximation (\( \approx \)) and equivalence (\( \simeq \)) that are sound with respect to contextual approximation (\( \simeq_{\text{ctx}} \)) and equivalence (\( \simeq_{\text{ref}} \)), and hence can be used to derive free theorems about well-typed GSF terms (Ahmed et al. 2017; Wadler 1989). The definitions of contextual approximation and equivalence, and the soundness of the logical relation, are fairly standard and left to §E.3.

As shown by Ahmed et al. (2017), in a gradual setting, the free theorems that hold for System F are weaker, as they have to be understood "modulo errors and divergence". Ahmed et al. (2017) prove two such free theorems in \( \lambda B \). However, these free theorems only concern fully static type signatures. This leaves unanswered the question of what imprecise free theorems are enabled by gradual parametricity. To the best of our knowledge, this topic has not been formally developed in the literature so far, despite several claims about expected theorems, exposed hereafter.
whose statement declare that parametricity dictates that any value of type \(1\). As usual, the predicates for values and terms carry a type environment and thus \(\text{ImpSV}_\rho[T \sqsubseteq G]\). \[\begin{align*}
N_\rho^\Sigma[V] & = \{v = \epsilon b :: G \mid v \in \text{ImpSV}_\rho[V] \sqsubseteq G\} \\
N_\rho^\Sigma[T_1 \rightarrow T_2 \sqsubseteq G] & = \{v = \epsilon u :: \rho(G) \mid v \in \text{ImpSV}_\rho[T_1 \rightarrow T_2 \sqsubseteq G] \land \forall v' \in N_\rho^\Sigma[T_1 \sqsubseteq \text{dom}_\rho\Sigma(G)]\} \\
N_\rho^\Sigma[\forall X.T \sqsubseteq G] & = \{v = \epsilon u :: \rho(G) \mid v \in \text{ImpSV}_\rho[\forall X.T \sqsubseteq G] \land (\forall T'. \Sigma \vdash T'. \Sigma \triangleright (\epsilon u :: \forall X.\text{schm}_\rho\Sigma(\rho(G)))[T']\} \\
N_\rho^\Sigma[\forall T_1 \times T_2 \sqsubseteq G] & = \{v = \epsilon u :: \rho(G) \mid v \in \text{ImpSV}_\rho[\forall T_1 \times T_2 \sqsubseteq G] \land (p_1(\epsilon) \pi_1(u) :: \text{pro}_{\rho}F(\rho(G))) \in N_\rho^\Sigma[T_1 \sqsubseteq \text{pro}_{\rho}F(G)]\} \\
N_\rho^\Sigma[X \sqsubseteq G] & = \{v = \epsilon \pi_1(\rho(G)) \mid v \in \text{ImpSV}_\rho[X \sqsubseteq G] \land (\forall G'. \Sigma \vdash G'. (\pi_1(\epsilon), \text{lift}_\Sigma(T)) u :: \rho(G')) \in C_\rho^\Sigma[T \sqsubseteq G'. T']\} \\
N_\rho^\Sigma[\alpha \sqsubseteq G] & = \{v = \epsilon u :: \rho(G) \mid \text{static}(u) \land \pi_2(\epsilon) = \text{lift}_\Sigma(\rho(T)) \land \Sigma; \vdash t :: \rho(G)\} \\
N_\rho^\Sigma[\Delta; \Gamma \vdash t :: G \triangleq \Sigma; \Delta; \Gamma \vdash t :: G \land \forall \rho \in D^\Sigma[\Delta], \forall y \in G_\rho^\Sigma[\Gamma], \rho(y(t)) \in C_\rho^\Sigma[T \sqsubseteq G]\} \\
N_\rho^\Sigma[\gamma :: T \sqsubseteq G] & = \{v = \epsilon u :: \rho(G) \mid \forall \rho \in D^\Sigma[\Delta], \forall y \in G_\rho^\Sigma[\Gamma], \rho(y(t)) \in C_\rho^\Sigma[T \sqsubseteq G]\} \\
N_\rho^\Sigma[\gamma :: T \sqsubseteq G] & = \{v = \epsilon u :: \rho(G) \mid \forall \rho \in D^\Sigma[\Delta], \forall y \in G_\rho^\Sigma[\Gamma], \rho(y(t)) \in C_\rho^\Sigma[T \sqsubseteq G]\}
\end{align*}\]

Igarashi et al. (2017a) report that the System F polymorphic identity function, if allowed to be cast to \(\forall X.\Sigma \rightarrow X\), would always trigger a runtime error when applied, suggesting that functions of type \(\forall X.\Sigma \rightarrow X\) are always failing. Consequently, System F\(_G\) rejects such a cast by tweaking the precision relation (§2.3). But the corresponding free theorem is not proven. Also, Ahmed et al. (2011) declare that parametricity dictates that any value of type \(\forall X.\Sigma \rightarrow ?\) is either constant or always failing or diverging (p.7). This gradual free theorem is not proven either. In fact, in both an older system (Ahmed et al. 2009) and its newest version (Ahmed et al. 2017), as well as in System F\(_G\), casting the identity function to \(\forall X.\Sigma \rightarrow ?\) yields a function that returns without errors, though the returned value is still sealed, and as such unusable (§2.4). Considering that the underlying function is intrinsically parametric, why shall we expect it to fail or return unusable values? In fact, while the specific choice of runtime semantics may decree failure, such behavior is not imposed by the parametricity relation per se. Parametricity only imposes uniformity of behavior, including failure, of polymorphic terms, which leaves some freedom regarding when to fail.

Disproving Gradual Free Claims. We uncover a novel property of GSF: it preserves the strong normalization property of System F terms even as they are ascribed to less precise types, as long as they are used with similarly-terminating terms, and instantiated at static types.

We establish this result using a logical predicate, named imprecise termination (Figure 7\(^{15}\)), whose statement \(\models t :: T \sqsubseteq G\) expresses that \(t\) is a static term of type \(T\) that has been ascribed a less precise type \(G\). As usual, the predicates for values and terms carry a type environment and type name store; we do not need step indexing because the logical relation is defined inductively on the structure of \(T\) (not \(G\)). At the function type, the predicate specifies that when applied to an imprecisely-terminating argument, the application terminates and yields an imprecisely-terminating result. For type application, only static type instantiations are considered. The predicate

\[\text{schm}_\rho\Sigma(\forall X.\Sigma) = G, \text{schm}_\rho\Sigma(?) = ?, \text{undefined o/w.}\]

\(^{15}\) \text{schm}_\rho\Sigma(\forall X.\Sigma) = G, \text{schm}_\rho\Sigma(?) = ?, \text{undefined o/w.}\]
ImpSV\(\frak{ρ}\)[\(T \subseteq G\)] characterizes imprecisely-ascribed static values. The rest of the definitions are essentially administrative ascriptions to align types as required by GSFε.

Static terms satisfy the imprecise termination predicate, and are hence hereditarily terminating:

**Lemma 10.1.** Let \(t\) be a static term. If \(\vdash t : T\) and \(T \sqsubseteq G\), then \(\vdash (t :: G) \leadsto t' : T \subseteq G\).

This property is related to, but weaker than the dynamic gradual guarantee. Nevertheless, it is powerful enough to disprove the claims from the literature about \(\forall X. ? \rightarrow X\) and \(\forall X. X \rightarrow ?\): both types admit the ascribed System F identity function, among many others,\(^{16}\) as a non-constant, non-failing, parametricity-preserving inhabitant. We believe this result constitutes a valuable compositionality guarantee when embedding fully-static (System F) terms in a gradual world. Another corollary is that closed static terms always terminate (by \(\mid = t : T \sqsubseteq T\)), hence superseding Proposition 7.4.

**Cheap Theorems.** The intuition of \(\forall X. ? \rightarrow X\) denoting always-failing functions is not entirely misguided: this result does hold for a subset of the terms of that type. This leads us to observe that we can derive “cheap theorems” with gradual parametricity: obtained not by looking only at the type, but by also considering the head constructors of a term. For instance:

**Theorem 10.2.** Let \(\nu \triangleq \Lambda X. \lambda x : ? . t\) for some \(t\), such that \(\vdash \nu : \forall X. ? \rightarrow X\). Then for any \(\vdash \nu' : G\), we either have \(\nu \mid G \nu' \downarrow \text{error}\) or \(\nu \mid G \nu' \uparrow \|\).

This result holds independently of the body \(t\), therefore without having to analyze the whole term. Not as good as a free theorem, but cheap.

11 RELATED WORK

We have already discussed at length related work on gradual parametricity, especially the most recent developments (Ahmed et al. 2017; Igarashi et al. 2017a; Xie et al. 2018). In addition to static semantics issues in \(\lambda B\) and System F\(_G\), all theses languages suffer from dynamic semantics that do not accurately track type instantiations (§2.4). Note that, conversely to \(\lambda B\), GSF does not impose any syntactic value restriction on polymorphic terms; such a restriction might be necessary when exploring the extension of GSF with implicit polymorphism. Finally, instead of leaving the dynamic gradual guarantee as a conjecture, we show that it is incompatible with parametricity, at least given the standard definitions of both notions. Note that some language features are also known to break the dynamic gradual guarantee, such as structural type tests and object identity (Siek et al. 2015a), as well as method overloading and extension methods (Muehlboeck and Tate 2017).

The relation between parametric polymorphism in general and dynamic typing much predates the work on gradual typing. Abadi et al. (1991) first note that without further precaution, type abstraction might be violated. Subsequent work explored different approaches to protect parametricity, especially runtime-type generation (RTG) (Abadi et al. 1995; Leroy and Mauny 1991; Rossberg 2003). Pierce and Sumii (2000) and Guha et al. (2007) use dynamic sealing, originally proposed by Morris (1973), in order to dynamically enforce type abstraction. Matthews and Ahmed (2008) also use RTG in order to protect polymorphic functions in an integration of Scheme and ML. This line of work eventually led to the polymorphic blame calculus (Ahmed et al. 2011) and its most recent version with the proof of parametricity by Ahmed et al. (2017). We adapt their logical relation to the evidence-based semantics of GSF.

Hou et al. (2016) prove the correctness of compiling polymorphism to dynamic typing with embeddings and partial projections; the compilation setting however differs significantly from gradual typing. New and Ahmed (2018) use embedding-projection pairs to formulate a semantic\(^{16}\) e.g. \(\Lambda X. \lambda x : X . \lambda f : X \rightarrow X . f x\) of type \(\forall X. X \rightarrow (X \rightarrow X) \rightarrow X\) can also be ascribed to \(\forall X. X \rightarrow ?\).
account of the dynamic gradual guarantee, coined graduality, in a language with explicit casts. It would be interesting to extend their simply-typed setting to parametric polymorphism, and study the interplay of parametricity and graduality when casts, and possibly seals, are explicit as in the work of Neis et al. (2009) on parametricity in a non-parametric language.

Devriese et al. (2018) disprove a conjecture by Pierce and Sumii (2000) according to which the compilation of System F to an untyped language with dynamic sealing is fully abstract, i.e. preserves contextual equivalences. They show that, for similar reasons, the embedding of System F in current polymorphic blame calculi is not fully abstract; their observation also applies to GSF. Full abstraction might be too strong a criteria for gradual typing: already in the simply-typed setting, embedding typed terms in gradual contexts is not fully abstract, because gradual types admit non-terminating terms. Imprecise termination (§10) is a weaker, yet useful result that sheds light on gradual free theorems about imprecise type signatures. It should be possible to generalize this result to account for the harmless content of imprecise ascriptions; we leave this perspective for future work.

This work is generally related to gradualization of advanced typing disciplines, in particular to gradual information-flow security typing (Disney and Flanagan 2011; Fennell and Thiemann 2013, 2016; Garcia and Tanter 2015; Toro et al. 2018). In these systems, one aims at preserving noninterference, i.e. that private values do not affect public outputs. Both parametricity and noninterference are 2-safety properties, expressed as a relation of two program executions. While Garcia and Tanter (2015) show that one can derive a pure security language with AGT that satisfies both noninterference and the dynamic gradual guarantee, Toro et al. (2018) find that in presence of mutable references, one can have either the dynamic gradual guarantee, or noninterference, but not both. Also similarly to this work, AGT for security typing needs a more precise abstraction for evidence types (based on security label intervals) in order to enforce noninterference. Together, these results suggest that new criteria are needed to characterize the spectrum of type-based reasoning that gradual typing supports when applied to semantically-rich disciplines.

12 CONCLUSION

We uncover design flaws in prior work on gradual parametric languages that enforce relational parametricity. We exploit the Abstracting Gradual Typing (AGT) methodology to design a new gradual language with explicit parametric polymorphism, GSF. We find that AGT greatly streamlines the static semantics of GSF, but does not yield a language that respects parametricity by default; non-trivial exploration was necessary to uncover how to strengthen the structure and treatment of runtime evidence in order to recover parametricity. We show that parametricity is, like noninterference (Toro et al. 2018), incompatible with the dynamic gradual guarantee laid forth by Siek et al. (2015a). We nevertheless establish a novel, weaker property of GSF regarding the embedding of System F terms at less precise types, which allows us to disprove some claims from the literature about gradual free theorems.

Future work also includes extending GSF and its associated reasoning with existential types, both in terms of their encoding, and as primitives in the language. We shall also study the integration of implicit polymorphism on top of GSF, most likely following the approach of Xie et al. (2018). Finally, it would be interesting to understand whether the evidence-based runtime semantics presented here can be used to derive a cast calculus akin to $\lambda B$, and then address efficiency considerations.

ACKNOWLEDGMENTS

We thank Amal Ahmed, Dominique Devriese, Kenji Maillard, Gabriel Scherer, the attendees of various oral presentations of this work, and the anonymous reviewers, for useful feedback and suggestions that improved both the presentation and our understanding of this work.
REFERENCES

Martin Abadi, Luca Cardelli, Benjamin Pierce, and Gordon Plotkin. 1991. Dynamic typing in a statically typed language. *ACM Transactions on Programming Languages and Systems* 13, 2 (April 1991), 237–268.

Martin Abadi, Luca Cardelli, Benjamin Pierce, and Didier Rémy. 1995. Dynamic typing in polymorphic languages. *Journal of Functional Programming* 5, 1 (1995), 111–130.

Amal Ahmed, Robert Bruce Findler, Jacob Matthews, and Philip Wadler. 2009. Blame for All. In *Workshop on Script to Program Evolution (STOP)*, Genova, Italy.

Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek, and Philip Wadler. 2011. Blame for All. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2011)*. ACM Press, Austin, Texas, USA, 201–214.

Amal Ahmed, Dustin Jamner, Jeremy G. Siek, and Philip Wadler. 2017. Theorems for Free for Free: Parametricity, with and Without Types. See(ICFP 2017 2017), 39:1–39:28.

Johannes Bader, Jonathan Aldrich, and Éric Tanter. 2018. Gradual Program Verification. In *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2018)* (Lecture Notes in Computer Science), Isil Dillig and Jens Palsberg (Eds.), Vol. 10747. Springer-Verlag, Los Angeles, CA, USA, 25–46.

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

Felipe Bañados Schwerter, Ronald Garcia, and Éric Tanter. 2016. Gradual Type-and-Effect Systems. *Journal of Functional Programming* 26 (Sept. 2016), 19:1–19:69.

Haskell B. Curry, J. Roger Hindley, and J. P. Seldin. 1972. *Combinatory Logic, Volume II*. Studies in logic and the foundations of mathematics, Vol. 65. North-Holland Pub. Co.

Dominique Devriese, Marco Patrignani, and Frank Piessens. 2018. Parametricity versus the universal type. *Proceedings of the ACM on Programming Languages* 2, POPL (Jan. 2018), 38:1–38:23.

Tim Disney and Cormac Flanagan. 2011. Gradual information flow typing. In *International Workshop on Scripts to Programs*. Luminous Fennell and Peter Thiemann. 2013. Gradual Security Typing with References. In *Proceedings of the 26th Computer Security Foundations Symposium (CSF)*, 224–239.

Ronald Garcia, Alison M. Clark, and Éric Tanter. 2016. Abstracting Gradual Typing. In *Proceedings of the 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2016)*. ACM Press, St Petersburg, FL, USA, 429–442.

Ronald Garcia and Éric Tanter. 2015. Deriving a Simple Gradual Security Language. eprint arXiv:1511.01399.

Ronald Garcia, Éric Tanter, Roger Wolff, and Jonathan Aldrich. 2014. Foundations of Typestate-Oriented Programming. *ACM Transactions on Programming Languages and Systems* 36, 4, Article 12 (Oct. 2014), 12:1–12:44 pages.

Jean-Yves Girard. 1972. *Interprétation Fonctionnelle et Élimination des Coupures de l’Arithmétique d’Ordre Supérieur*. Ph.D. Dissertation. Université de Paris VII, Paris, France.

Arjun Guha, Jacob Matthews, Robert Bruce Findler, and Shriram Krishnamurthi. 2007. Relationally-parametric polymorphic contracts. In *Proceedings of the ACM Dynamic Languages Symposium (DLS 2007)*. ACM Press, Montreal, Canada, 29–40.

Jean-Yves Girard. 1972. *Interprétation Fonctionnelle et Élimination des Coupures de l’Arithmétique d’Ordre Supérieur*. Ph.D. Dissertation. Université de Paris VII, Paris, France.

Arjun Guha, Jacob Matthews, Robert Bruce Findler, and Shriram Krishnamurthi. 2007. Relationally-parametric polymorphic contracts. In *Proceedings of the ACM Dynamic Languages Symposium (DLS 2007)*. ACM Press, Montreal, Canada, 29–40.

David Herman, Aaron Tomb, and Cormac Flanagan. 2010. Space-efficient gradual typing. *Higher-Order and Symbolic Computation* 23, 2 (June 2010), 167–189.

Kuen-Bang Hou, Nick Benton, and Robert Harper. 2016. Correctness of Compiling Polymorphism to Dynamic Typing. *Journal of Functional Programming* 27 (2016), 1:1–1:24.

Kuen-Bang Hou, Nick Benton, and Robert Harper. 2016. Correctness of Compiling Polymorphism to Dynamic Typing. *Journal of Functional Programming* 27 (2016), 1:1–1:24.

ICFP 2017 2017.

Atsushi Igarashi, Peter Thiemann, Vasco T. Vasconcelos, and Philip Wadler. 2017b. Gradual Session Types. See(ICFP 2017 2017), 38:1–38:28.

Yuu Igarashi, Taro Sekiyama, and Atsushi Igarashi. 2017a. On Polymorphic Gradual Typing. See(ICFP 2017 2017), 40:1–40:29.
Lintaro Ina and Atsushi Igarashi. 2011. Gradual typing for generics. In Proceedings of the 26th ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2011). ACM Press, Portland, Oregon, USA, 609–624.

Nico Lehmann and Éric Tanter. 2017. Gradual Refinement Types. In Proceedings of the 44th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2017). ACM Press, Paris, France, 775–788.

Xavier Leroy and Michel Mauny. 1991. Dynamics in ML. In Proceedings of the Conference on Functional Programming Languages and Computer Architecture (FPCA 1991) (Lecture Notes in Computer Science), Vol. 523. Springer-Verlag, 406–426.

Jacob Matthews and Amal Ahmed. 2008. Parametric Polymorphism Through Run-Time Sealing, or, Theorems for Low, Low Prices!. In Proceedings of the 17th European Symposium on Programming Languages and Systems (ESOP 2008) (Lecture Notes in Computer Science), Sophia Drossopoulou (Ed.), Vol. 4960. Springer-Verlag, Budapest, Hungary, 16–31.

Jacob Matthews and Robert Bruce Findler. 2007. Operational Semantics for Multi-language programs. In Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2007). ACM Press, Nice, France, 3–10.

John C. Mitchell. 1988. Polymorphic Type Inference and Containment. Information and Computation 76, 2-3 (Feb. 1988), 211–249.

James H. Morris. 1973. Protection in Programming Languages. Commun. ACM 16, 1 (Jan. 1973), 15–21.

Fabian Muehlboeck and Ross Tate. 2017. Sound gradual typing is nominally alive and well. , 56:1–56:30 pages.

Georg Neis, Derek Dryer, and Andreas Rossberg. 2009. Non-Parametric Parametricity. In Proceedings of the 14th ACM SIGPLAN Conference on Functional Programming (ICFP 2009). ACM Press, Edinburgh, Scotland, UK, 135–148.

Max S. New and Amal Ahmed. 2018. Graduality from Embedding-Projection Pairs. , 73:1–73:30 pages.

Benjamin C. Pierce. 2002. Types and programming languages. MIT Press, Cambridge, MA, USA.

Benjamin C. Pierce and Eijiro Sumii. 2000. Relating Cryptography and Polymorphism. Manuscript.

Aseem Rastogi, Avik Chaudhuri, and Basil Hosmer. 2012. The ins and outs of gradual type inference. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2012). ACM Press, Philadelphia, USA, 481–494.

Martin Odersky and Konstantin Läufer. 1996. Putting Type Annotations to Work. In Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 96). ACM Press, St. Petersburg Beach, Florida, USA, 54–67.

Benjamin Pierce and Eijiro Sumii. 2000. Relating Cryptography and Polymorphism. Manuscript.

Benjamin C. Pierce. 2002. Types and programming languages. MIT Press, Cambridge, MA, USA.

Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. 2015a. Refined Criteria for Gradual Typing. In Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2016). ACM Press, Portland, Oregon, USA, 964–974.

Matías Toro, Ronald Garcia, and Eric Tanter. 2018. Type-Driven Gradual Security with References. ACM Transactions on Programming Languages and Systems 40, 4 (Nov. 2018), 16:1–16:55.

Matías Toro, Elizabeth Labrada, and Eric Tanter. 2019. Gradual Parametricity, Revisited. Proceedings of the ACM on Programming Languages 3, POPL (Jan. 2019), 17:1–17:30.
Matías Toro and Éric Tanter. 2017. A Gradual Interpretation of Union Types. In *Proceedings of the 24th Static Analysis Symposium (SAS 2017) (Lecture Notes in Computer Science)*, Vol. 10422. Springer-Verlag, New York City, NY, USA, 382–404.

Philip Wadler. 1989. Theorems for Free!. In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture (FP'CA '89)*. ACM, London, United Kingdom, 347–359.

Roger Wolff, Ronald Garcia, Éric Tanter, and Jonathan Aldrich. 2011. Gradual Typestate. In *Proceedings of the 25th European Conference on Object-oriented Programming (ECOOP 2011) (Lecture Notes in Computer Science)*, Mira Mezini (Ed.), Vol. 6813. Springer-Verlag, Lancaster, UK, 459–483.

Ningning Xie, Xuan Bi, and Bruno C. d. S. Oliveira. 2018. Consistent Subtyping for All. In *Proceedings of the 27th European Symposium on Programming Languages and Systems (ESOP 2018) (Lecture Notes in Computer Science)*, Amal Ahmed (Ed.), Vol. 10801. Springer-Verlag, Thessaloniki, Greece, 3–30.
Appendix

CONTENTS

Contents 31

A SF: Well-formedness 32

B GSF: Statics 33
  B.1 Syntax and Syntactic Meaning of Gradual Types 33
  B.2 Lifting the Static Semantics 34
  B.3 Well-formedness 37
  B.4 Static Properties 37

GSF: Dynamics 43
  C.1 Evidence Type Precision 43
  C.2 Initial Evidence 43
  C.3 Consistent Transitivity 44
  C.4 GSF\(\varepsilon\): Dynamic Semantics 44
  C.5 Translation from GSF to GSF\(\varepsilon\) 45

D GSF: Properties 47
  D.1 Type Safety 47
  D.2 Static Terms Do Not Fail 51

E GSF: Parametricity 53
  E.1 Auxiliary Definitions 53
  E.2 Fundamental Property 53
  E.3 Contextual Equivalence 81

F GSF: Imprecise Termination 83

G A Cheap Theorem in GSF 90
In this section we present auxiliary definitions for well-formedness of type name stores, and well-formedness of types.

**Definition A.1 (Well-formedness of the type name store).**

\[
\begin{align*}
\Gamma & \vdash \cdot & \alpha \notin \Sigma & \Sigma \vdash T \\
\Gamma & \vdash \cdot & \Sigma \vdash \alpha : T
\end{align*}
\]

**Definition A.2 (Well-formedness of types).**

\[
\begin{align*}
\Gamma & \vdash \Sigma & \Sigma ; \Delta \vdash T_1 & \Sigma ; \Delta \vdash T_2 \\
\Sigma ; \Delta \vdash T_1 \rightarrow T_2 & \\
\Sigma ; \Delta , X \vdash T & \\
\Sigma ; \Delta \vdash \forall X . T & \\
\Sigma ; \Delta \vdash T_1 \times T_2 & \\
\Gamma & \vdash \Sigma & \Sigma ; \Delta \vdash X \\
\Sigma ; \Delta \vdash X & \\
\Gamma & \vdash \Sigma & \alpha : T \in \Sigma \\
\Sigma ; \Delta \vdash \alpha
\end{align*}
\]
B  GSF: STATICS

In this section we present auxiliary definitions and proofs of the statics semantics of GSF not presented in the paper.

B.1 Syntax and Syntactic Meaning of Gradual Types

Proposition B.1 (Precision, inductively). The inductive definition of type precision given in Figure 3 is equivalent to Definition 5.1.

Proof. Direct by induction on the type structure of $G_1$ and $G_2$. We only present representative cases to illustrate the reasoning used in the proof. We prove first that $C(G_1) \subseteq C(G_2) \Rightarrow G_1 \subseteq G_2$, where $G_1 \subseteq G_2$ stands for the inductive definition given in Figure 3.

Case ($G_1 = B, G_2 = B$). Then $\{ B \} \subseteq \{ B \}$, but we already know that $B \subseteq B$ and the result holds.

Case ($G_1 = G, G_2 = ?$). Then $C(G) \subseteq C(?)$ = TYPE, but $G \subseteq ?$ is an axiom and the result holds.

Case ($G_1 = \forall X.G'_1, G_2 = \forall X.G'_2$). Then we know that $\{ \forall X.T \mid T \in C(G'_1) \} \subseteq \{ \forall X.T \mid T \in C(G'_2) \}$, then it must be the case that $C(G'_1) \subseteq C(G'_2)$. Then by induction hypothesis $G_1 \subseteq G_2$, then by inductive definition of precision for type abstractions, $\forall X.G_1 \subseteq \forall X.G_2$ and the result holds.

Then we prove the other direction, i.e. $G_1 \subseteq G_2 \Rightarrow C(G_1) \subseteq C(G_2)$.

Case ($G_1 = B, G_2 = B$). Then $B \subseteq B$, but we already know that $\{ B \} \subseteq \{ B \}$ and the result holds.

Case ($G_1 = G, G_2 = ?$). Then $G \subseteq ?$, but $C(G) \subseteq C(?)$ = TYPE and the result holds.

Case ($G_1 = \forall X.G'_1, G_2 = \forall X.G'_2$). Then we know that $\forall X.G_1 \subseteq \forall X.G_2$, then by looking at the premise of the corresponding definition, $G'_1 \subseteq G'_2$. Then by induction hypothesis $C(G'_1) \subseteq C(G'_2)$. But we have to prove that $\{ \forall X.T \mid T \in C(G'_1) \} \subseteq \{ \forall X.T \mid T \in C(G'_2) \}$, which is direct from $C(G'_1) \subseteq C(G'_2)$.

□

Proposition B.2 (Galois connection). $(C, A)$ is a Galois connection, i.e.:

a) (Soundness) for any non-empty set of static types $S = \{ T \}$, we have $S \subseteq C(A(S))$

b) (Optimality) for any gradual type $G$, we have $A(C(G)) \subseteq G$.

Proof. We first proceed to prove a) by induction on the structure of the non-empty set $S$.

Case ($\{ B \}$). Then $A(\{ B \}) = B$. But $C(B) = \{ B \}$ and the result holds.

Case ($\{ T_{i1} \rightarrow T_{i2} \}$). Then $A(\{ T_{i1} \rightarrow T_{i2} \}) = A(\{ T_{i1} \}) \rightarrow A(\{ T_{i2} \})$. But by definition of $C$, $C(A(\{ T_{i1} \})) \rightarrow A(\{ T_{i2} \}) = \{ T_{i1} \rightarrow T_{i2} \mid T_{i1} \in C(A(\{ T_{i1} \})) \}$. By induction hypotheses, $\{ T_{i1} \} \subseteq C(A(\{ T_{i1} \}))$ and $\{ T_{i2} \} \subseteq C(A(\{ T_{i2} \}))$, therefore $\{ T_{i1} \rightarrow T_{i2} \} \subseteq \{ T_{i1} \rightarrow T_{i2} \mid T_{i1} \in C(A(\{ T_{i1} \})) \} \subseteq \{ T_{i1} \rightarrow T_{i2} \mid T_{i1} \in C(A(\{ T_{i1} \})) \}$ and the result holds.

Case ($\{ T_{i1} \times T_{i2} \}$). We proceed analogous to case $\{ T_{i1} \rightarrow T_{i2} \}$.

Case ($\{ X \}, \{ \alpha \}$). We proceed analogous to case $\{ B \}$.

Case ($\{ \forall X.T \}$). Then $A(\{ \forall X.T \}) = \forall X.A(\{ T \})$. But by definition of $C$, $C(\forall X.A(\{ T \})) = \{ \forall X.T \mid T \in C(A(\{ T \})) \}$. By induction hypothesis, $\{ T \} \subseteq C(A(\{ T \}))$, therefore $\forall X.T = \{ \forall X.T \mid T \in C(A(\{ T \})) \}$ and the result holds.
Case (\{ T_i \} heterogeneous). Then \( A(\{ T_i \}) = ? \) and therefore \( C(A(\{ T_i \})) = \text{TYPE} \), but \( \{ T_i \} \subseteq \text{TYPE} \) and the result holds.

Now let us proceed to prove b) by induction on gradual type \( G \).

Case (\( B \)). Trivial because \( C(B) = \{ B \} \), and \( A(\{ B \}) = B \).

Case \( (G_1 \rightarrow G_2) \). We have to prove that \( A(C(G_1 \rightarrow G_2)) \subseteq G_1 \rightarrow G_2 \), which is equivalent to prove that \( C(A(T)) \subseteq T \), where \( \hat{T} = C(G_1 \rightarrow G_2) = \{ T_1 \rightarrow T_2 \mid T_1 \in C(G_1), T_2 \in C(G_2) \} \). Then \( \hat{T} \) has the form \( \{ T_{i1} \rightarrow T_{i2} \} \), such that \( \forall i, T_{i1} \in C(G_1) \) and \( T_{i2} \in C(G_2) \). Also note that \( \{ T_{i1} \} = C(G_1) \) and \( \{ T_{i2} \} = C(G_2) \). But by definition of \( A \), \( A(\{ T_{i1} \rightarrow T_{i2} \}) = A(\{ T_{i1} \}) \rightarrow A(\{ T_{i2} \}) \) and therefore \( C(A(\{ T_{i1} \})) \rightarrow A(\{ T_{i2} \}) = \{ T_1 \rightarrow T_2 \mid T_1 \in C(A(\{ T_{i1} \})), T_2 \in C(A(\{ T_{i2} \})) \} \). But by induction hypothesis \( C(A(\{ T_{i1} \})) \subseteq C(G_1) \) and \( C(A(\{ T_{i2} \})) \subseteq C(G_2) \) and the result holds.

Case \( (G_1 \times G_2) \). We proceed analogous to case \( G_1 \rightarrow G_2 \).

Case \( (X, \alpha) \). We proceed analogous to case \( B \).

Case \( (\forall X.G) \). We have to prove that \( A(C(\forall X.G)) \subseteq \forall X.G \), which is equivalent to prove that \( C(A(T)) \subseteq T \), where \( \hat{T} = C(G) = \{ X.T \mid T \in C(G) \} \). Then \( \hat{T} \) has the form \( \{ \forall X.T_i \} \), such that \( \forall i, T_i \in C(G) \). Also note that \( \{ T_i \} = C(G) \). But by definition of \( A \), \( A(\{ \forall X.T_i \}) = \forall X.A(\{ T_i \}) \) and therefore \( C(X.A(\{ T_i \})) = \{ X.T \mid T \in C(A(\{ T_i \})) \} \). But by induction hypothesis \( C(A(\{ T_i \})) \subseteq C(G) \) and the result holds.

Case (?). Then we have to prove that \( C(A(?)) \subseteq C(?) = \text{TYPE} \), but this is always true and the result holds immediately.

\[\Box\]

B.2 Lifting the Static Semantics

Definition B.3 (Store precision). \( \Xi_1 \subseteq \Xi_2 \) if and only if \( \text{dom}(\Xi_1) = \text{dom}(\Xi_2) \) and \( \forall \alpha \in \text{dom}(\Xi_1), \Xi_1(\alpha) \subseteq \Xi_2(\alpha) \).

Lemma B.4. If \( \Xi_1 \subseteq \Xi_2, \vdash \Xi, G_1 \subseteq G_2, \) and \( \Xi_1; \Delta \vdash G_1, \) then \( \Xi_2; \Delta \vdash G_2. \)

Proof. Straightforward induction on relation \( G_1 \subseteq G_2 \). We only present interesting cases.

Case \( (G_1 = \forall X.G_1, G_2 = \forall X.G_2) \). By definition of precision \( G_1' \subseteq G_2' \). By definition of well-formedness of types, \( \Xi_1; X \vdash G_1' \) and then by induction hypothesis \( \Xi_2; \Delta, X \vdash G_2 \). Then by definition of well-formedness of types \( \Xi_2; \Delta \vdash \forall X.G_2' \) and the result holds.

Case \( (G_2 = ?) \). This is trivial because as \( \vdash \Xi, \) then \( \Xi_2; \Delta \vdash ? \).

Case \( (G_1 = \alpha, G_2 = \alpha) \). Trivial by definition of \( \Xi_1 \subseteq \Xi_2, \alpha \in \text{dom}(\Xi_2) \), therefore \( \alpha : G_2' \in \Xi_2 \) and then \( \Xi_2; \Delta \vdash \alpha \).

\[\Box\]

Lemma B.5. Let \( \Xi_1 \subseteq \Xi_2 \), then \( \vdash \Xi_1 \Rightarrow \Xi_2 \).

Proof. By induction on relation \( \Xi_1 \subseteq \Xi_2 \).

Case (\( \cdot \subseteq \cdot \)). Trivial as \( \vdash \cdot \).

Case \( (\Xi_1', \alpha : G_1 \subseteq \Xi_2', \alpha : G_2) \). By definition of store precision we know that \( \Xi_1' \subseteq \Xi_2' \) and that \( G_1 \subseteq G_2 \). By definition of well-formedness, \( \vdash \Xi_1', \alpha : G_1 \Rightarrow \Xi_2' \), therefore by induction hypothesis \( \vdash \Xi_2' \). We only have left to prove is that \( \Xi_2' \vdash \Xi_2 \), which follows directly from Lemma B.4.
Lemma B.6. If $\Sigma \in C(\Xi)$ and $\vdash \Sigma$, then $\vdash \Xi$.

Proof. Corollary of Lemma B.5 as $\Sigma \subseteq \Xi$.

Lemma B.7. If $\Sigma; \Delta \vdash T_1 = T_2$, then $\Sigma; \Delta \vdash T_1$ and $\Sigma; \Delta \vdash T_2$.

Proof. By induction on relation $\Sigma; \Delta \vdash T_1 = T_2$. Most cases are straightforward, so we present only the interesting cases.

Case $(T_1 = \forall X. T'_1, T_2 = \forall X. T'_2)$. As $\Sigma; \Delta \vdash \forall X. T'_1 = \forall X. T'_2$, by inspection of the derivation rule, $\Sigma; \Delta, X \vdash T'_1 = T'_2$. By induction hypotheses we know that $\Sigma; \Delta, X \vdash T'_1$, and that $\Sigma; \Delta, X \vdash T'_2$. Therefore by well-formedness of types we know that $\Sigma; \Delta \vdash \forall X. T'_1$ and that $\Sigma; \Delta \vdash \forall X. T'_2$ and the result holds.

Case $(T_1 = X, T_2 = X)$. As $\Sigma; \Delta \vdash X = X$, then we know by inspection of the derivation rule that $\vdash \Sigma$ and that $X \in \Delta$. Then as $\vdash \Sigma$ and that $X \in \Delta; \Delta \vdash X$ and the result holds.

Proposition B.8 (Consistency, Inductively). The inductive definition of type consistency given in Figure 3 is equivalent to Definition 5.4.

Proof. First we prove that $\Sigma; \Delta \vdash T_1 = T_2$ for some $\Sigma \in C(\Xi)$, $T_1 \in C(G_i)$ implies that $\Xi; \Delta \vdash G_1 \sim G_2$, where $\Xi; \Delta \vdash G_1 \sim G_2$ stands for the inductive definition of consistency. We proceed by straightforward induction on $G_i$ such that the predicate holds (we only show interesting cases). By Lemma B.6 we know that if $\vdash \Sigma$, then $\vdash \Xi$, which will be assumed to be true whenever is needed.

Case $(G_1 = B, G_2 = B)$. Then $\Sigma; \Delta \vdash B = B$, but we already know that $\Xi \vdash B \sim B$ and the result holds.

Case $(G_1 = G, G_2 = ?)$. We know that $\Sigma; \Delta \vdash T_1 = T_2$ for some $T_1 \in C(G)$ and $T_2 \in C(?)$. Then by Lemma B.7, $\Sigma; \Delta \vdash T_1$, and as $\Sigma \subseteq \Xi$ and $T_1 \in G$, by Lemma B.4, $\Xi; \Delta \vdash G$. Then as $\Xi; \Delta \vdash G$, $G \sim ? = \text{Type}$ and the result holds.

Case $(G_1 = \forall X. G_1', G_2 = \forall X. G_2')$. Then we know that $\Sigma; \Delta \vdash \forall X. T_1 \equiv \forall X. T_2$ where $\forall X. T_1 \in C(\forall X. G_1'), \forall X. T_2 \in C(\forall X. G_2')$. Notice that $T_1 \in C(G_1'), T_2 \in C(G_2')$, and that $\Sigma; \Delta, X \vdash T_1 = T_2$. Then by induction hypotheses, $\Xi \vdash G_1' \sim G_2'[\Delta, X]$, and therefore $\Xi; \Delta \vdash \forall X. G_1' \sim \forall X. G_2'$ and the result holds.

Then we prove the other direction, i.e. $G_1 \subseteq G_2 \Rightarrow C(G_1) \sim C(G_2)$.

Case $(G_1 = B, G_2 = B)$. Then $B \subseteq B$, but we already know that $B \in C(B)$ and $\Sigma; \Delta \vdash B = B$, and the result holds immediately.

Case $(G_1 = G, G_2 = ?)$. Then $G \subseteq ?$. Let $T_1 \in C(G)$ and $\Sigma \in C(\Xi)$ such that $\Sigma; \Delta \vdash T_1$. As $?, \text{Type}$, we can choose $T_1 \in \text{Type}$, so $\Sigma; \Delta \vdash T_1$, and the result holds.

Case $(G_1 = \forall X. G_1', G_2 = \forall X. G_2')$. Then we know that $\Xi; \Delta \vdash \forall X. G_1' \sim \forall X. G_2'$, then by looking at the premise of the corresponding definition, $\Xi; \Delta, X \vdash G_1' \sim G_2'$. Then by induction hypotheses $\forall T_1 \in C(G_1'), T_2 \in C(G_2'), \Sigma \in C(\Xi)$, such that $\Sigma; \Delta, X \vdash T_1 = T_2$. By definition of consistency $\forall X. T_1 \in C(G_i)$. Then by definition of equality, $\Sigma; \Delta \vdash \forall X. T_1 = \forall X. T_2$ and the result holds.
Definition B.9 (Consistent lifting of functions). Let $F_n$ be a function of type $\text{Type}^n \to \text{Type}$. Its consistent lifting $\tilde{F}_n^\#$, of type $\text{GType}^n \to \text{GType}$, is defined as: $F_n^\#(\tilde{G}) = A(\{ F_n(\tilde{T}) \mid \tilde{T} \in C(\tilde{G}) \})$

Lemma B.10. $G = A(C(G))$

Proof. Then we have to prove that $G = A(C(G))$. By optimality (Prop 5.3.b), we know that $A(C(G)) \subseteq G$, and by soundness (Prop 5.3.a), $C(G) \subseteq C(A(C(G)))$, i.e. $G \subseteq A(C(G))$. Therefore $G \subseteq A(C(G))$ and $A(C(G)) \subseteq G$, thus $G = A(C(G))$ and the result holds. □

Lemma B.11. $G[G'/X] = A(\{ T'[T'/X] \mid T \in C(G), T' \in C(G') \})$.

Proof. We proceed by induction on $G$. We only present interesting cases.

Case ($G = X$). Then $G[G'/X] = G'$, and $C(G) = \{ X \}$. Then we have to prove that $G' = A(\{ T' \mid T' \in C(G') \})$. But notice that $A(\{ T' \mid T' \in C(G') \}) = A(C(G'))$ and by Lemma B.10 the result holds immediately.

Case ($G = ?$). Then $G[G'/X] = ?$, and $C(G) = \text{Type}$. Then we have to prove that $? = A(\{ T[T'/X] \mid T \in \text{Type}, T' \in C(G') \})$. But notice that $A(\{ T[T'/X] \mid T \in \text{Type}, T' \in C(G') \}) = A(C(\text{Type}))$ and by Lemma B.10 the result holds immediately.

Case ($G = \forall Y.G''$). Then $G[G'/X] = \forall Y.G'[G'/X]$, and $C(G) = \forall Y.C(G'')$. Then we have to prove that $\forall Y.G''[\forall Y.G'[G'/X]] = A(\{ \forall Y.T''[\forall Y.T'[T'/X] \mid T'' \in C(G''), T' \in C(G') \}) = \forall Y.A(\{ T''[\forall Y.T'[T'/X] \mid T'' \in C(G''), T' \in C(G') \})$. Then by induction hypothesis on $G''$, $G''[\forall Y.G'[G'/X]] = A(\{ T''[\forall Y.T'[T'/X] \mid T'' \in C(G''), T' \in C(G') \})$, therefore $\forall Y.G''[\forall Y.G'[G'/X]] = \forall Y.A(\{ T''[\forall Y.T'[T'/X] \mid T'' \in C(G''), T' \in C(G') \})$ and the result holds.

□

Proposition B.12 (Consistent type functions). The definitions of $\text{dom}^\#$, $\text{cod}^\#$, $\text{inst}^\#$, and $\text{proj}^\#$ given in Fig. 3 are consistent liftings, as per Def. 5.6, of the corresponding functions from Fig. 1.

Proof. We present the proof for $\text{inst}^\#$ and $\text{dom}^\#$ (the other proofs are analogous).

First we prove that $\text{inst}^\#(G, G') = A(\text{inst}(C^2(G, G')))$, where $\text{inst}^\#(G, G')$ correspond to the algorithmic definitions presented in Fig. 3. Notice that

\[
A(\text{inst}(C^2(G, G')))
= A(\text{inst}(\{ \langle T, T' \rangle \mid T \in C(G), T' \in C(G') \}))
= A(\{ T[T'/X] \mid \forall X. T \in C(G), T' \in C(G') \})
\]

But then the result follows immediately from Lemma B.11.

Then we prove that $\text{dom}^\#(G) = A(\text{dom}(C(G)))$, where $\text{dom}^\#(G)$ correspond to the algorithmic definitions presented in Fig. 3. We proceed by induction on $G$.

Case ($G = G_1 \to G_2$). Notice that

\[
A(\text{dom}(C(G)))
= A(\text{dom}(C(G_1 \to G_2)))
= A(\text{dom}(\{ T_1 \to T_2 \mid T_1 \in C(G_1), T_2 \in C(G_2) \}))
= A(\{ T_1 \mid T_1 \in C(G_1) \})
= A(C(G_1))
\]
But $dom^\#(G_1 \rightarrow G_2) = G_1$. Then we have to prove that $G_1 = A(C(G_1))$ which holds immediately by Lemma B.10.

Case $(G = ?)$. Notice that
\[
A(dom(C(G))) = A(dom(C(?))) = A(dom(TYPE)) = A(TYPE) = ?
\]
and the result holds immediately as $dom^\#(?) = ?$.

Case $(G \neq ? \neq G_1 \rightarrow G_2)$. If $G$ has not the form $G_1 \rightarrow G_2$, or is not $?$, then $dom^\#(G)$ is undefined. Then as $?, T \in C(G)$ such that $T = T_1 \rightarrow T_2$ the result holds immediately as $dom(T)$ is undefined $\forall T \in C(G)$.

\[\Box\]

### B.3 Well-formedness

In this section we present auxiliary definitions of the static semantics of GSF.

**Definition B.13** (Well-formedness of type name store).

\[
\frac{}{\vdash \cdot}
\quad
\frac{\alpha \notin \Xi}{\vdash \Sigma, \alpha : G}
\]

**Definition B.14** (Well-formedness of types).

\[
\frac{}{\vdash \Xi}
\quad
\frac{\Xi; \Delta \vdash B}{\Xi; \Delta \vdash G_1 \rightarrow G_2}
\quad
\frac{\Xi; \Delta \vdash G_1}{\Xi; \Delta, \forall X, G \vdash \Xi; \Delta \vdash G_1 \times G_2}
\quad
\frac{\Xi; \Delta \vdash G_1}{\Xi; \Delta \vdash G_2}
\quad
\frac{X \in \Delta}{\Xi; \Delta \vdash X}
\quad
\frac{\alpha : G \in \Xi}{\Xi; \Delta \vdash \alpha}
\quad
\frac{}{\Xi; \Delta \vdash ?}
\]

### B.4 Static Properties

In this section we present two static properties of GSF and the proof: the static equivalence for static terms and the static gradual guarantee.

**B.4.1 Static Equivalence for Static Terms.**

**Proposition B.15** (Static equivalence for static terms). Let $t$ be a static term and $G$ a static type ($G = T$). We have $\vdash_S t : T$ if and only if $\vdash t : T$.

**Proof.** We prove this proposition for open terms instead. The proof is direct thanks to the equivalence between the typing rules and the equivalence between type equality and type consistency rules for static types. We only present one case to illustrate the reasoning.

First we prove $\Sigma; \Delta \vdash_S t : T \Rightarrow \Sigma; \Delta \vdash t : T$ by induction on judgment $\Sigma; \Delta \vdash_S t : T$. 

...
Case \((\Sigma; \Delta \vdash_S t'[T'']) : \text{inst}(\forall X.T', T''))\). Then \(\Sigma; \Delta \vdash S t' : \forall X.T'\), and by induction hypothesis \(\Sigma; \Delta \vdash t' : \forall X.T'\). Then \(\text{inst}(\forall X.T, T'') = T[T''/X] = \text{inst}(\forall X.T', T'')\), and as \(\Sigma; \Delta \vdash T''\), therefore \(\Sigma; \Delta \vdash t'[T''] : T[T''/X]\) and the result holds.

Then we prove \(\Sigma; \Delta \vdash t : T \Rightarrow \Sigma; \Delta \vdash_S t : T\) by induction on judgment \(\Sigma; \Delta \vdash t : T\).

Case \((\Sigma; \Delta \vdash t'[T''] : \text{inst}(\forall X.T', T''))\). Then \(\Sigma; \Delta \vdash t' : \forall X.T'\), and by induction hypothesis \(\Sigma; \Delta \vdash_S t' : \forall X.T'\). Then \(\text{inst}(\forall X.T, T'') = T[T''/X] = \text{inst}(\forall X.T', T'')\), and as \(\Sigma; \Delta \vdash T''\), therefore \(\Sigma; \Delta \vdash t'[T''] : T[T''/X]\) and the result holds.

\[\Box\]

### B.4.2 Static Gradual Guarantee

In this section we present the proof of the static gradual guarantee property. In the Definition B.16 and Definition B.17 we present term precision and type environment precision.

**Definition B.16 (Term precision).**

\[
\begin{align*}
(Px) & \quad x \in x \\
(Pb) & \quad b \in b \\
(P\lambda) & \quad t \in t' \quad G \subseteq G' \\
& \quad (\lambda x : G.t) \subseteq (\lambda x : G'.t') \\
(P\triangledown) & \quad \exists t_1, t_2 \in \{ t'_1, t'_2 \} \\
& \quad \langle t_1, t_2 \rangle \subseteq \langle t'_1, t'_2 \rangle \\
(P\text{app}) & \quad t \in t' \quad t \in t' \quad G \subseteq G' \\
& \quad (t \cdot t) \subseteq (t' \cdot t') \\
(P\text{appG}) & \quad t \in t' \quad G \subseteq G' \\
& \quad t \cdot [t] \subseteq t' \cdot [G] \\
(P\text{pair}) & \quad \exists t \in t' \quad G \subseteq G' \\
& \quad \pi_1(t) \subseteq \pi_1(t') \\
\end{align*}
\]

**Definition B.17 (Type environment precision).**

\[
\Gamma \subseteq \Gamma' \quad G \subseteq G' \\
\Gamma, x : G \subseteq \Gamma', x : G' \\
\]

**Lemma B.18.** If \(\Xi; \Delta; \Gamma \vdash t : G \text{ and } \Gamma \subseteq \Gamma'\), then \(\Xi; \Delta; \Gamma' \vdash t : G'\) for some \(G \subseteq G'\).

**Proof.** Simple induction on type derivation \(\Xi; \Delta; \Gamma \vdash t : G\) (we only present interesting cases).

Case \((t = x)\). We know that \(\Sigma; \Delta; \Gamma \vdash x : G\) and \(\Gamma(x) = G\). By definition of \(\Gamma \subseteq \Gamma'\), \(\Gamma(x) \subseteq \Gamma'(x)\), therefore \(\Sigma; \Delta; \Gamma \vdash x : G'\), where \(G \subseteq G'\) and the result holds.

Case \((t = (\lambda x : G.t'))\). We know that \(\Sigma; \Delta; \Gamma \vdash (\lambda x : G.t') : G_1 \rightarrow G_2\), where \(\Sigma; \Delta; \Gamma, x : G_1 \vdash t' : G_2\). As \(\Gamma \subseteq \Gamma'\) and \(G_1 \subseteq G_1\), then by definition of precision for type environments, \(\Gamma, x : G_1 \subseteq \Gamma', x : G_1'\). Therefore by induction hypothesis on \(\Sigma; \Delta; \Gamma, x : G_1 \vdash t' : G_2\), \(\Sigma; \Delta; \Gamma', x : G_1 \vdash t' : G_2'\), where \(G_2 \subseteq G_2'\). Finally, by \((P\lambda)\), \(\Sigma; \Delta; \Gamma' \vdash (\lambda x : G.t') : G_1 \rightarrow G_2'\), and as \(G_1 \rightarrow G_2 \subseteq G_1 \rightarrow G_2'\), the result holds.

\[\Box\]

**Lemma B.19.** If \(\Xi; \Delta \vdash G_1 \sim G_2\) and \(G_1 \subseteq G_1'\) and \(G_2 \subseteq G_2'\) then \(\Xi; \Delta \vdash G_1' \sim G_2'\).

**Proof.** By definition of \(\Xi; \Delta \vdash \cdot \sim \cdot\), there exists \((T_1, T_2) \in C^2(G_1, G_2)\) such that \(T_1 = T_2\). \(G_1 \subseteq G_1'\) and \(G_2 \subseteq G_2'\), mean that \(C(G_1) \subseteq C(G_1')\) and \(C(G_2) \subseteq C(G_2')\), therefore \((T_1, T_2) \in C^2(G_1', G_2')\), and the result follows.

\[\Box\]

**Lemma B.20.** If \(G_1 \subseteq G_1'\) and \(G_2 \subseteq G_2'\) then \(G_1[G_2/X] \subseteq G_1'[G_2'/X]\).

**Proof.** By induction on the relation of \(G_1 \subseteq G_1'\). We only present interesting cases.
Case ($X \sqsubseteq X$). Then we have to prove that $X[G_2/X] \sqsubseteq X[G'_2/X]$, which is equivalent to $G_2 \sqsubseteq G'_2$, but that is part of the premise and the result holds immediately.

Case ($G_1 \sqsubseteq ?$). Then we have to prove that $G_1[G_2/X] \sqsubseteq ?$ which is always true.

Case ($\forall Y.G_3 \sqsubseteq \forall Y.G'_3$). Then we have to prove that $\forall Y.G_3[G_2/X] \sqsubseteq \forall Y.G'_3[G'_2/X]$, which is equivalent to prove that $G_3[G_2/X] \sqsubseteq G'_3[G'_2/X]$, which holds by induction hypothesis on $G_3 \sqsubseteq G'_3$.

\[\Box\]

**Lemma B.21.** If $G_1 \sqsubseteq G'_1$ and $G_2 \sqsubseteq G'_2$ then $\text{inst}^\sharp(G_1, G_2) \sqsubseteq \text{inst}^\sharp(G'_1, G'_2)$.

**Proof.** By induction on relation $G_1 \sqsubseteq G'_1$.

Case ($? \sqsubseteq ?$). The result is trivial as $\text{inst}^\sharp(?, G'_1) = ?$ and $? \sqsubseteq ?$.

Case ($\forall X.G_1 \sqsubseteq ?, \forall X.G_1 \sqsubseteq \forall X.G_2$). The result follows directly from Lemma B.20.

\[\Box\]

**Lemma B.22.** If $G_1 \sqsubseteq G_2$ then $\text{proj}^\sharp_i(G_1) \sqsubseteq \text{proj}^\sharp_i(G_2)$.

**Proof.** The proof is direct, analogous to Lemma B.21, by induction on relation $G_1 \sqsubseteq G_2$.

\[\Box\]

**Proposition B.23** (Static Gradual Guarantee for Open Terms). If $\Xi; \Delta; \Gamma \vdash t_1 : G_1$ and $t_1 \sqsubseteq t_2$, then $\Xi; \Delta; \Gamma \vdash t_2 : G_2$, for some $G_2$ such that $G_1 \sqsubseteq G_2$.

**Proof.** We prove the property on opens terms instead of closed terms: If $\Xi; \Delta; \Gamma \vdash t_1 : G_1$ and $t_1 \sqsubseteq t_2$ then $\Xi; \Delta; \Gamma \vdash t_2 : G_2$ and $G_1 \sqsubseteq G_2$.

The proof proceed by induction on the typing derivation.

**Case ($Gx, Gb$).** Trivial by definition of term precision ($\sqsubseteq$) using ($Px$), ($Pb$) respectively.

**Case ($G\lambda$).** Then $t_1 = (\lambda x : G'_1.t)$ and $G_1 = G'_1 \Rightarrow G'_2$. By ($G\lambda$) we know that:

\[
\begin{array}{c}
\Xi; \Delta; \Gamma \vdash \lambda x : G'_1.t : G'_2 \\
\text{(G\lambda)}
\end{array}
\]

(1)

Consider $t_2$ such that $t_1 \sqsubseteq t_2$. By definition of term precision $t_2$ must have the form $t_2 = (\lambda x : G''_1.t')$ and therefore

\[
\begin{array}{c}
t \sqsubseteq t' \quad G'_1 \sqsubseteq G''_1 \\
\text{(P\lambda)}
\end{array}
\]

(2)

Using induction hypotheses on the premises of (1) and (2), $\Xi; \Delta; \Gamma, x : G'_1 \vdash t' : G''_2$ with $G'_2 \sqsubseteq G''_2$. By Lemma B.18, $\Xi; \Delta; \Gamma, x : G''_2 \vdash t' : G''_2$ where $G''_2 \sqsubseteq G''_2$. Then we can use rule ($G\lambda$) to derive:

\[
\begin{array}{c}
\Xi; \Delta; \Gamma, x : G'_1 \vdash t' : G''_2 \\
\text{(G\lambda)}
\end{array}
\]

Where $G_2 \sqsubseteq G''_2$. Using the premise of (2) and the definition of type precision we can infer that

\[
G'_1 \Rightarrow G'_2 \sqsubseteq G''_1 \Rightarrow G''_2
\]

and the result holds.
Case (GA). Then $t_1 = (\lambda X.t)$ and $G_1 = \forall X.G'_1$. By (GA) we know that:

$$(\text{GA}) \quad \vdash \Xi; \Delta, X; \Gamma \vdash t : G'_1$$

Consider $t_2$ such that $t_1 \sqsubseteq t_2$. By definition of term precision $t_2$ must have the form $t_2 = (\lambda X.t')$ and therefore

$$(\text{P\Lambda}) \quad \vdash t \sqsubseteq t'$$

(4)

Using induction hypotheses on the premises of (3) and (4), $\Xi; \Delta, X; \Gamma \vdash t' : G''_1$ with $G'_1 \sqsubseteq G''_1$. Then we can use rule (GA) to derive:

$$(\text{GA}) \quad \vdash \Xi; \Delta, X; \Gamma \vdash t' : G''_1$$

Using the definition of type precision we can infer that

$$\forall X.G'_1 \sqsubseteq \forall X.G''_1$$

and the result holds.

Case (Gpair). Then $t_1 = \langle t'_1, t'_2 \rangle$ and $G_1 = G'_1 \times G'_2$. By (Gpair) we know that:

$$(\text{Gpair}) \quad \vdash \Xi; \Delta; \Gamma \vdash t'_1 : G'_1 \quad \Xi; \Delta; \Gamma \vdash t'_2 : G'_2$$

Consider $t_2$ such that $t_1 \sqsubseteq t_2$. By definition of term precision, $t_2$ must have the form $\langle t''_1, t''_2 \rangle$ and therefore

$$(\text{Ppair}) \quad \vdash t'_1 \sqsubseteq t''_1 \quad t'_2 \sqsubseteq t''_2$$

(6)

Using induction hypotheses on the premises of (5) and (6), $\Xi; \Delta; \Gamma \vdash t''_1 : G''_1$ and $\Xi; \Delta; \Gamma \vdash t''_2 : G''_2$, where $G'_1 \sqsubseteq G''_1$ and $G'_2 \sqsubseteq G''_2$. Then we can use rule (Gpair) to derive:

$$(\text{Gpair}) \quad \vdash \Xi; \Delta; \Gamma \vdash t''_1 : G''_1 \quad \Xi; \Delta; \Gamma \vdash t''_2 : G''_2$$

Finally, using the definition of type precision we can infer that

$$G'_1 \times G'_2 \sqsubseteq G''_1 \times G''_2$$

and the result holds.

Case (Gasc). Then $t_1 = t :: G_1$. By (Gasc) we know that:

$$(\text{Gasc}) \quad \vdash \Xi; \Delta; \Gamma \vdash t : G \quad \Xi; \Delta \vdash G \sim G_1$$

(7)

Consider $t_2$ such that $t_1 \sqsubseteq t_2$. By definition of term precision $t_2$ must have the form $t_2 = t' :: G_2$ and therefore

$$(\text{Pasc}) \quad \vdash t \sqsubseteq t' \quad G_1 \sqsubseteq G_2$$

(8)

Using induction hypotheses on the premises of (7) and (8), $\Xi; \Delta; \Gamma \vdash t' : G'$ where $G \sqsubseteq G'$. We can use rule (Gasc) and Lemma B.19 to derive:

$$(\text{Gasc}) \quad \vdash \Xi; \Delta; \Gamma \vdash t' : G' \quad \Xi; \Delta \vdash G' \sim G_2$$

$$(\text{Gasc}) \quad \vdash \Xi; \Delta; \Gamma \vdash t' :: G_2$$
Where $G_1 \subseteq G_2$ and the result holds.

Case (Cop). Then $t_1 = \text{op}(\overline{t})$ and $G_1 = G^*$. By (Gop) we know that:

$$
\Xi; \Delta; \Gamma \vdash \overline{t}: \overline{G} \quad \text{ty}(\text{op}) = \overline{G_2} \rightarrow G^*
$$

$$
\Xi; \Delta + \overline{G} \sim G_2
$$

(9)

Consider $t_2$ such that $t_1 \subseteq t_2$. By definition of term precision $t_2$ must have the form $t_2 = \text{op}(\overline{t})$ and therefore

$$
\Xi; \Delta; \Gamma \vdash t \subseteq \overline{t}
$$

(10)

Using induction hypotheses on the premises of (9) and (10), $\Xi; \Delta; \Gamma \vdash \overline{t}: \overline{G}^\cdot$, where $\overline{G} \subseteq \overline{G}^\cdot$. Using the Lemma B.19 we know that $\Xi; \Delta \vdash \overline{G}^\cdot \sim \overline{G_2}$. Therefore we can use rule (Gop) to derive:

$$
\Xi; \Delta; \Gamma \vdash \overline{t}: \overline{G}^\cdot \quad \text{ty}(\text{op}) = \overline{G_2} \rightarrow G^*
$$

$$
\Xi; \Delta + \overline{G}^\cdot \sim \overline{G_2}
$$

(11)

and the result holds.

Case (Gapp). Then $t_1 = t_1' \ t_2'$ and $G_1 = \text{cod}^\#(G_1^\cdot)$. By (Gapp) we know that:

$$
\Xi; \Delta; \Gamma \vdash t_1': G_1^\cdot \quad \Xi; \Delta; \Gamma \vdash t_2': G_2^\cdot
$$

$$
\Xi; \Delta + \text{dom}^\#(G_1^\cdot) \sim G_2^\cdot
$$

(11)

Consider $t_2$ such that $t_1 \subseteq t_2$. By definition of term precision $t_2$ must have the form $t_2 = t_1'' \ t_2''$ and therefore

$$
\Xi; \Delta; \Gamma \vdash t_1' \ t_2' \subseteq t_2''
$$

(12)

Using induction hypotheses on the premises of (11) and (12), $\Xi; \Delta; \Gamma \vdash t_1'': G_1''$ and $\Xi; \Delta; \Gamma \vdash t_2'': G_2''$, where $G_1' \subseteq G_1''$ and $G_2' \subseteq G_2''$. By definition type precision and the definition of $\text{dom}^\#$, $\text{dom}^\#(G_1^\cdot) \subseteq \text{dom}^\#(G_1'')$ and, therefore by Lemma B.19, $\Xi; \Delta \vdash \text{dom}^\#(G_1'') \sim G_2''$. Also, by the previous argument $\text{cod}^\#(G_1') \subseteq \text{cod}^\#(G_1'')$. Then we can use rule (Gapp) to derive:

$$
\Xi; \Delta; \Gamma \vdash t_1'' \ t_2'' \subseteq \text{cod}^\#(G_1'')
$$

(12)

and the result holds.

Case (GappG). Then $t_1 = t \ [G]$. By (GappG) we know that:

$$
\Xi; \Delta; \Gamma \vdash t: G_1^\cdot \quad \Xi; \Delta \vdash G
$$

$$
\Xi; \Delta; \Gamma \vdash t \ [G] : \text{inst}^\#(G_1^\cdot, G)
$$

(13)

where $G_1 = \text{inst}^\#(G_1^\cdot, G)$. Consider $t_2$ such that $t_1 \subseteq t_2$. By definition of term precision $t_2$ must have the form $t_2 = t' \ [G']$ and therefore

$$
\Xi; \Delta; \Gamma \vdash t' \ [G'] \subseteq t' \ [G']
$$

(14)
Using induction hypotheses on the premises of (13) and (14), $\Xi; \Delta; \Gamma \vdash t' : G_2'$ where $G_1' \sqsubseteq G_2'$. We can use rule (GapG) and Lemma B.4 to derive:

$$
(G_{asc}) \quad \Xi; \Delta; \Gamma \vdash t' : G_2' \quad \Xi; \Delta \vdash G' \quad \Xi; \Delta; \Gamma \vdash t' \ [G'] : \ inst^\#(G_2', G')
$$

Finally, by the Lemma B.21 we know that $inst^\#(G_1', G) \sqsubseteq inst^\#(G_2', G')$ and the result holds.

**Case (Cpairi).** Then $t_1 = \pi_i(t)$ and $G_1 = proj^\#_i(G)$. By (Gpair) we know that:

$$
(G_{pairi}) \quad \Xi; \Delta; \Gamma \vdash t : G \quad \Xi; \Delta; \Gamma \vdash \pi_i(t) : proj^\#_i(G)
$$

Consider $t_2$ such that $t_1 \sqsubseteq t_2$. By definition of term precision, $t_2$ must have the form $\pi_i(t')$ and therefore

$$
(P_{pairi}) \quad t \sqsubseteq t' \quad \pi_i(t) \sqsubseteq \pi_i(t')
$$

Using induction hypotheses on the premises of (15) and (16), $\Xi; \Delta; \Gamma \vdash t' : G'$ where $G \sqsubseteq G'$. Then we can use rule (Cpairi) to derive:

$$
(C_{pairi}) \quad \Xi; \Delta; \Gamma \vdash t' : G' \quad \Xi; \Delta; \Gamma \vdash \pi_i(t') : proj^\#_i(G')
$$

Finally, by the Lemma B.22 we can infer that $proj^\#_i(G) \sqsubseteq proj^\#_i(G')$ and the result holds.

□

**Proposition B.24 (Static Gradual Guarantee).** Let $t$ and $t'$ be closed GSF terms such that $t \sqsubseteq t'$ and $\vdash t : G$. Then $\vdash t' : G'$ and $G \sqsubseteq G'$.

**Proof.** Direct corollary of Prop. B.23. □
C  GSF: DYNAMICS

In this section, we expose auxiliary definitions of the dynamic semantics of GSF. First, we present type precision, interior and consistent transitivity definitions for evidence types. Then we show some important definitions, used in the dynamic semantics of GSF. Finally, we present the translation semantics from GSF to GSFr.

C.1 Evidence Type Precision

Figure 8 presents the definition of the evidence type precision.

\[
\begin{array}{c|c|c|c|c}
E \sqsubseteq E & E \sqsubseteq E & E \sqsubseteq E & E \sqsubseteq E & \forall X. E_1 \sqsubseteq \forall X. E_2 \\
B \sqsubseteq B & X \sqsubseteq X & E_1 \sqsubseteq E'_1 & E_2 \sqsubseteq E'_2 & E_1 \rightarrow E_2 \sqsubseteq E'_1 \rightarrow E'_2 \\
\end{array}
\]

Fig. 8. Evidence Type Precision

C.2 Initial Evidence

In Figure 9 we present the interior function, used to compute the initial evidence.

\[
I : \text{ETYPE} \times \text{ETYPE} \rightarrow \text{EVIDENCE}
\]

\[
\begin{align*}
I(E, E) &= I(?, E) = I(E, ?) = \langle E, E \rangle \\
I(E_1, E_2) &= \langle E'_1, E'_2 \rangle \\
I(aE_1, E_2) &= \langle aE'_1, E'_2 \rangle \\
I(E_1, aE_2) &= \langle E'_1, aE'_2 \rangle \\
I(\forall X. E, \forall X.? &= \langle E'_1, E'_2 \rangle \\
I(\forall X.?, \forall X. E) &= \langle E'_1, E'_2 \rangle \\
I(\forall X.?, E_1) &= \langle E'_1, E'_2 \rangle \\
I(\forall X.?, E_1) &= \langle E'_1, E'_2 \rangle \\
I(E_{11} \times E_{12}, ? \times ?) &= \langle E'_{11} \times E'_{12}, E'_{21} \times E'_{22} \rangle \\
I(E_{11} \times E_{12}, ?) &= \langle E'_{11} \times E'_{12}, E'_{21} \times E'_{22} \rangle \\
I(E_{11}, E_{12}) &= \langle E'_{11}, E'_{12} \rangle \\
I(E_{11}, E_{12}) &= \langle E'_{11}, E'_{12} \rangle \\
I(E_{21}, E_{11}) &= \langle E'_{21}, E'_{11} \rangle \\
I(E_{11}, E_{22}) &= \langle E'_{11}, E'_{22} \rangle \\
I(E_{11}, E_{22}) &= \langle E'_{11}, E'_{22} \rangle \\
I(E_{11}, E_{22}) &= \langle E'_{11}, E'_{22} \rangle \\
I(E_{11}, E_{22}) &= \langle E'_{11}, E'_{22} \rangle \\
I(E_{11}, E_{22}) &= \langle E'_{11}, E'_{22} \rangle \\
I(\forall X. E_1, \forall X. E_2) &= \langle \forall X. E'_1, \forall X. E'_2 \rangle \\
\end{align*}
\]

Fig. 9. GSF: Computing Initial Evidence
C.3 Consistent Transitivity

In Figure 10, we present the definition of consistent transitivity for evidence types.

\[
\begin{align*}
\text{(base)} & \quad (E, E) \circ (E, E) = (E, E) \\
\text{(idL)} & \quad (B, B) \circ (B, B) = (B, B) \\
\text{(idR)} & \quad (?, ?) \circ (E, E) = (E, E) \\
\text{(unl)} & \quad (E_1, E_2) \circ (E_3, E_4) = (E_1', E_2') \\
\text{(sealL)} & \quad (\alpha E_1, E_2) \circ (\alpha E_3, E_4) = (E_1', \alpha E_2') \\
\text{(sealR)} & \quad (E_1, E_2) \circ (\alpha E_3, E_4) = (\alpha E_1, E_2') \\
\text{(func)} & \quad \langle E_{41}, E_{31} \rangle \circ \langle E_{21}, E_{11} \rangle = \langle E_{3}, E_{1} \rangle \\
\text{(pair)} & \quad (\forall X. E_1, \forall X. E_2) \circ (\forall X. \exists, \forall X. \exists) = (\forall X. E_1', \forall X. E_2') \\
\text{Fig. 10. GSF: Consistent Transitivity}
\end{align*}
\]

C.4 GSF: Dynamic Semantics

In this section, we show the function definitions used in the dynamic semantics of GSFe, specifically in the type application rule (RappG).

**Definition C.1.**

\[\epsilon_{\text{out}} \triangleq \langle E, [\alpha E], E'[E'] \rangle \quad \text{where} \quad E_{\epsilon} = \text{lift}_{\varepsilon}(\text{unlift}(\pi_{\varepsilon}(\epsilon))), \alpha^E = \text{lift}_{\varepsilon}(\alpha), E' = \text{lift}_{\varepsilon}(G')\]
C.5 Translation from GSF to GSF

In this section we present the translation from GSF to GSF (Figure 11), which inserts ascriptions to ensure that top-level constructors match in every elimination form. We use the following normalization metafunction:

\[ \text{norm}(t, G_1, G_2, \xi) = \begin{cases} \epsilon t : G_2 & \text{if } G_1 \neq G_2 \land \epsilon = I_{\xi}(G_1, G_2) \\ t & \text{if } G_1 = G_2 \end{cases} \]

\[ I_{\xi}(G_1, G_2) = I(lift_{\xi}(G_1), lift_{\xi}(G_2)) \]

**Theorem C.6 (Translation Preserves Typing).** If \( \xi; \Delta; \Gamma \vdash t : G \), then \( \xi; \Delta; \Gamma \vdash t' : G \) and \( \xi; \Delta; \Gamma \vdash t' : G \).

**Proof.** The proof follows by induction on the typing derivation of \( \xi; \Delta; \Gamma \vdash t : G \), exploiting the fact that the term produced by \( \text{norm}(t, G_1, G_2, \xi) \) has type \( G_2 \). \( \square \)
\[ (Gb) \quad \text{ty}(b) = B \quad \Xi; \Delta \vdash \Gamma \] 
\[ \Xi; \Delta; \Gamma \vdash b \rightsquigarrow b : B \] 

\[ (G\lambda) \quad \Xi; \Delta; \Gamma, x : G \vdash t \rightsquigarrow t' : G' \] 
\[ \Xi; \Delta; \Gamma \vdash (\lambda x : G. t) \rightsquigarrow (\lambda x : G. t') : G \rightarrow G' \] 

\[ (\text{Gpair}) \quad \Xi; \Delta; \Gamma \vdash u_1 \rightsquigarrow u'_1 : G_1 \quad \Xi; \Delta; \Gamma \vdash u_2 \rightsquigarrow u'_2 : G_2 \] 
\[ \Xi; \Delta; \Gamma \vdash \langle u_1, u_2 \rangle \rightsquigarrow \langle u'_1, u'_2 \rangle : G_1 \times G_2 \] 

\[ (\text{Gapp}) \quad \Xi; \Delta; \Gamma \vdash t \rightsquigarrow t' : G \] 
\[ t'' = \text{norm}(t', G, \text{var}^\#(G), \text{schm}^\#(G), \Xi) \] 
\[ \Xi; \Delta; \Gamma \vdash \pi_t(t) \rightsquigarrow \pi_t(t') : \text{proj}^\#_1(G) \] 

\[ (\text{GappG}) \quad \Xi; \Delta; \Gamma \vdash t \rightsquigarrow t' : G \] 
\[ t'' = \text{norm}(t', G, \forall \text{var}^\#(G), \text{schm}^\#(G), \Xi) \] 
\[ \Xi; \Delta; \Gamma \vdash t [G'] \rightsquigarrow t'' [G'] : \text{inst}^\#(G, G'') \] 

\[ (\text{Gpair}) \quad \Xi; \Delta; \Gamma \vdash \pi_t(t) \rightsquigarrow \pi_t(t') : \text{proj}^\#_1(G) \] 

Fig. 11. GSF to GS\(\ell\) translation.
D GSF: PROPERTIES

In this section we present some properties of GSF. Section D.1, presents Type Safety and its proof. Section D.2, shows the property and proof about static terms do not fail.

D.1 Type Safety

In this section we present the proof of type safety for GSFs.

We define what it means for a store to be well typed with respect to a term. Informally, all free locations of a term and of the contents of the store must be defined in the domain of that store. Also, the store must preserve types between intrinsic locations and underlying values.

**Lemma D.1 (Canonical forms).** Consider a value $\Xi;\vdash u : G$. Then $v = \epsilon u :: G$, with $\Xi;\vdash u : G'$ and $\epsilon \vdash \Xi \vdash G' \sim G$. Furthermore:

1. If $G = B$, then $v = \epsilon b :: B$, with $\Xi;\vdash b : B$ and $\epsilon \vdash \Xi \vdash B \sim B$.
2. If $G = G_1 \rightarrow G_2$, then $v = \epsilon(\lambda x : G'_1.t) :: G_1 \rightarrow G_2$, with $\Xi;\vdash x : G'_1 \vdash t : G'_2$ and $\epsilon \vdash \Xi \vdash G'_1 \rightarrow G'_2 \sim G_1 \rightarrow G_2$.
3. If $G = \forall X.G_1$, then $v = \epsilon(\Lambda X.t) :: \forall X.G_1$, with $\Xi;\Delta;X;\vdash t : G'_1$ and $\epsilon \vdash \Xi \vdash \forall X.G'_1 \sim \forall X.G_1$.
4. If $G = G_1 \times G_2$, then $v = \epsilon(u_1,u_2) :: G_1 \times G_2$, with $\Xi;\vdash u_1 : G'_1$, $\Xi;\vdash u_2 : G'_2$ and $\epsilon \vdash \Xi \vdash G'_1 \times G'_2 \sim G_1 \times G_2$.

**Proof.** By direct inspection of the formation rules of evidence augmented terms.

D.2 (Substitution). If $\Xi;\Delta;\Gamma; x : G_1 \vdash t : G$, and $\Xi;\vdash v : G_1$, then $\Xi;\Delta;\Gamma \vdash t[v/x] : G$.

**Proof.** By induction on the derivation of $\Xi;\Delta;\Gamma; x : G_1 \vdash t : G$.

D.3 If $\epsilon \vdash \Xi;\Delta;X \vdash G_1 \sim G_2$, $\Xi;\vdash G', \alpha \notin \text{dom}(\Xi)$, and $E = \text{lift}_E(G')$, then $\epsilon[\alpha^E/X] \vdash \Xi;\Delta \vdash G_1[\alpha/X] \sim G_2[\alpha/X]$.

**Proof.** By induction on the judgment $\epsilon \vdash \Xi;\Delta;X \vdash G_1 \sim G_2$ and the definition of evidences.

D.4 (Type Substitution). If $\Xi;\Delta;X;\Gamma \vdash t : G$, $\Xi;\vdash G', \alpha \notin \text{dom}(\Xi)$, and $E = \text{lift}_E(G')$, then $\Xi;\Delta \vdash t[\alpha^E/X] : G[\alpha/X]$.

**Proof.** By induction on the derivation of $\Xi;\Delta;X;\Gamma \vdash t : G$ and Lemma D.3.

D.5 If $\epsilon_1 \vdash \Xi;\Delta \vdash G'_1 \sim G_1$, and $\epsilon_2 \vdash \Xi;\Delta \vdash G'_2 \sim G_2$, then $\epsilon_1 \times \epsilon_2 \vdash \Xi;\Delta \vdash G'_1 \times G'_2 \sim G_1 \times G_2$.

**Proof.** By definition of the judgment $\epsilon \vdash \Xi;\Delta;X \vdash G'_1 \times G'_2 \sim G_1 \times G_2$ and the definition of evidences.

D.6 If $\epsilon \vdash \Xi;\Delta \vdash G' \sim G$ then $p_1(\epsilon) \vdash \Xi;\Delta \vdash \text{proj}_1(G') \sim \text{proj}_1(G)$.

**Proof.** By definition of judgment $\epsilon \vdash \Xi;\Delta \vdash \text{proj}_1(G') \sim \text{proj}_1(G)$ and the definition of evidences.

**Proposition D.7 (\(\rightarrow\) is well defined).** If $\Xi;\vdash t : G$, then either

- $\Xi \triangleright t \rightarrow \Xi \triangleright t'$, $\exists \Xi' \text{ and } \Xi;\vdash t' : G$; or
- $\Xi \triangleright t \rightarrow \text{error}$
Proof. By induction on the structure of a derivation of $\Xi \vdash t \rightarrow r$, considering the last rule used in the derivation.

Case (Rapp). Then $t = (\varepsilon_1(\lambda x : G_{11}.t_1) :: G_1 \rightarrow G_2) (\varepsilon_2 u :: G_1)$. Then

\[
\frac{
\Xi; :: x : G_{11} \vdash t_1 : G_{12}}{
\Xi; :: \varepsilon_1 \vdash (\lambda x : G_{11}.t_1) :: G_{11} \rightarrow G_{12}}
\]

\[
\Xi; :: \varepsilon_1 \vdash G_{11} \rightarrow G_{12} \sim G_1 \rightarrow G_2
\]

\[
\Xi; :: (\varepsilon_1(\lambda x : G_{11}.t_1) :: G) (\varepsilon_2 u :: G_1) : G_2
\]

If $\varepsilon' = (\varepsilon_2 \circ \text{dom}(\varepsilon_1))$ is not defined, then $\Xi \vdash t \rightarrow \text{error}$, and then the result holds immediately. Suppose that consistent transitivity does hold, then

\[
\Xi \vdash (\varepsilon_1(\lambda x : G_{11}.t_1) :: G_1 \rightarrow G_2) (\varepsilon_2 u :: G_1) \rightarrow \Xi \vdash \text{cod}(\varepsilon_1)(t_1[\varepsilon' u :: G_{11}]/x) :: G_2
\]

As $\varepsilon_2 \vdash G'_2 \sim G_1$ and by inversion lemma $\text{dom}(\varepsilon_1) \vdash G_1 \sim G_{11}$, then $\varepsilon' \vdash G'_2 \sim G_{11}$. Therefore $\Xi; :: \varepsilon' u :: G_1 :: G_{11}$, and by Lemma D.2, $\Xi; :: t[(\varepsilon' u :: G_{11})/x] :: G_{12}$.

Let us call $t^{\varepsilon'} = t[(\varepsilon' u :: G_{11})/x]$. Then

\[
\Xi; :: \varepsilon_1 \vdash t_1[\varepsilon' u :: G_{11}]/x : G_{12} \quad \text{cod}(\varepsilon_1) \vdash \Xi; :: G_{12} \sim G_2
\]

and the result holds.

Case (RappG). Then $t = (\varepsilon \forall X.t_1 :: \forall X.G_X) [G']$. Consider $G_X = \text{schm}^\#(G)$, then

\[
\Xi \vdash (\varepsilon \forall X.t_1 :: G) [G'] \rightarrow \Xi' \vdash t^{E/F}[\varepsilon'[a/E]](\varepsilon[a/E] t_1[a/E/X] :: G_X[\alpha/X]) :: G_X[G'/X]
\]

where $\Xi' \triangleq \Xi$, $\alpha := G', \alpha \notin \text{dom}(\Xi)$, and $E' \triangleq \text{lift}_\Xi(G')$, and

$E'/\alpha^{\varepsilon,E} = \langle \varepsilon \forall X,G_X[\alpha/E/X], \text{lift}_\Xi(G_X[G'/X]) \rangle$. Notice that $\langle \varepsilon \forall X,G_X[\alpha/X], \text{lift}_\Xi(G_X[G'/X]) \rangle = I(G_X[\alpha/X], G_X[G'/X])$, and by definition of the special substitution, $\varepsilon \forall X,G_X[\alpha/E/X] \subseteq \text{lift}_\Xi(G_X[\alpha/X])$ (because $\text{lift}_\Xi(\alpha) = \alpha^{E'}$, and the substitution on evidences just extend unknowns with $\alpha$). Therefore $E'/\alpha^{\varepsilon,E} : I(G_X[\alpha/X], G_X[G'/X])$, and $E'/\alpha^{\varepsilon,E} \models \Xi; :: t_1[\alpha/E/X] :: G_1[\alpha/X]$. Also by Lemma D.3 $\varepsilon[a/E]] \models \Xi; :: t_1[\alpha/E/X] :: G_1[\alpha/X]$, and by Lemma D.4, $\Xi; :: t_1[\alpha/E'/X] : G_1[\alpha/X]$.

Then, as $\Xi \subseteq \Xi'$,

\[
\Xi; :: \varepsilon \forall X.t_1[\alpha/E'/X] : G_1[\alpha/X]
\]

\[
\Xi; :: \varepsilon \forall X.t_1[\alpha/E'/X] : G_1[\alpha/X]
\]

\[
\Xi; :: \varepsilon \forall X.t_1[\alpha/E'/X] : G_1[\alpha/X]
\]

and the result holds.
Case (Rasc). Then \( t = \varepsilon_1(\varepsilon_2u : G_2) :: G \). Then

\[
\begin{array}{c}
\begin{prooftree}
\Gamma \vdash \varepsilon_2 :: G_2
\end{prooftree}
\quad \begin{prooftree}
\Gamma \vdash \varepsilon_1 :: G_2 \sim G_1
\end{prooftree}
\end{array}
\quad \begin{prooftree}
\varepsilon_1 \vdash \varepsilon_2u :: G_2 : G_1
\end{prooftree}
\quad \begin{prooftree}
\varepsilon_1 \vdash \varepsilon_3 :: G_2 \sim 1
\end{prooftree}
\quad \begin{prooftree}
\varepsilon_1 \vdash \varepsilon_2u :: G_2 : G
\end{prooftree}
\end{array}
\]

If \((\varepsilon_2 \circ \varepsilon_1)\) is not defined, then \( \Xi \triangleright t \rightarrow \text{error} \), and then the result hold immediately. Suppose that consistent transitivity does hold, then

\[
\Xi \triangleright \varepsilon_1(\varepsilon_2u :: G_2) :: G \quad \rightarrow \quad \Xi \triangleright (\varepsilon_2 \circ \varepsilon_1)u :: G
\]

where \((\varepsilon_2 \circ \varepsilon_1) \vdash \Xi ; \vdash u :: G_u \sim G \). Then

\[
\begin{array}{c}
\Xi ; \vdash \varepsilon_2u :: G_u
\end{array}
\quad \begin{prooftree}
\Xi ; \vdash (\varepsilon_2 \circ \varepsilon_1)u :: G
\end{prooftree}
\]

and the result follows.

Case (Rop). Then \( t = \text{op}(\overline{\varepsilon u} :: B') \). Then

\[
\begin{array}{c}
\Xi \vdash \varepsilon u :: B \quad \rightarrow \quad \Xi \vdash \text{ty}(\text{op}) = B' \rightarrow B
\end{array}
\]

Let us assume that \( \text{ty}(\text{op}) : B' \rightarrow B \).

\[
\Xi \triangleright \text{op}(\overline{\varepsilon u} :: B') \quad \rightarrow \quad \Xi \triangleright \varepsilon_B \delta(\text{op}, \overline{u}) :: B
\]

But as \( \varepsilon_B \vdash \Xi ; \vdash B \sim B \), then

\[
\begin{array}{c}
\Xi ; \vdash \delta(\text{op}, \overline{u}) :: B
\end{array}
\quad \begin{prooftree}
\varepsilon_B \vdash \Xi ; \vdash B \sim B
\end{prooftree}
\]

and the result follows.

Case (Rpair). Then \( t = \langle \varepsilon_1u_1 :: G_1, \varepsilon_2u_2 :: G_2 \rangle \). Then

\[
\begin{array}{c}
\begin{prooftree}
\varepsilon_1 \vdash \Xi ; \vdash u_1 :: G_1'
\end{prooftree}
\quad \begin{prooftree}
\varepsilon_1 \vdash \Xi ; \vdash G_1' \sim G_1
\end{prooftree}
\end{array}
\quad \begin{prooftree}
\Xi \vdash \varepsilon_1u_1 :: G_1
\end{prooftree}
\quad \begin{prooftree}
\begin{prooftree}
\varepsilon_2 \vdash \Xi ; \vdash u_2 :: G_2'
\end{prooftree}
\quad \begin{prooftree}
\varepsilon_2 \vdash \Xi ; \vdash G_2' \sim G_2
\end{prooftree}
\end{prooftree}
\quad \begin{prooftree}
\Xi \vdash \varepsilon_2u_2 :: G_2
\end{prooftree}
\end{array}
\]

Then

\[
\Xi \triangleright \langle \varepsilon_1u_1 :: G_1, \varepsilon_2u_2 :: G_2 \rangle \quad \rightarrow \quad \Xi \triangleright \langle \varepsilon_1 \times \varepsilon_2 \rangle(u_1, u_2) :: G_1 \times G_2
\]

By Lemma D.5, \( \varepsilon_1 \times \varepsilon_2 \vdash \Xi ; \vdash G_1' \times G_2' \sim G_1 \times G_2 \). Then

\[
\begin{array}{c}
\begin{prooftree}
\varepsilon_1 \vdash \Xi ; \vdash u_1 :: G_1'
\end{prooftree}
\quad \begin{prooftree}
\varepsilon_2 \vdash \Xi ; \vdash u_2 :: G_2'
\end{prooftree}
\end{array}
\quad \begin{prooftree}
\langle u_1, u_2 \rangle :: G_1' \times G_2'
\end{prooftree}
\quad \begin{prooftree}
\varepsilon_1 \times \varepsilon_2 \vdash \Xi ; \vdash G_1' \times G_2' \sim G_1 \times G_2
\end{prooftree}
\]

and the result holds.

Case (Rproji). Then \( t = \pi_1(\varepsilon \langle u_1, u_2 \rangle :: G) \). Then
Then
\[
\Xi; \cdot \vdash p_1(\epsilon) u_i : \text{proj}_1^\#(G)
\]

By Lemma D.6, \( p_1(\epsilon) \models \Xi; \cdot \vdash \text{proj}_1^\#(G') \times G'_2 \sim \text{proj}_1^\#(G) \). Then
\[
\Xi; \cdot \vdash p_1(\epsilon) u_i : \text{proj}_1^\#(G) : \text{proj}_1^\#(G)
\]
and the result holds.

\[\square\]

**Proposition D.8** (\( \quad \rightarrow \quad \) is well defined). If \( \Xi; \cdot \vdash t : G \), then either

- \( \Xi \triangleright t \mapsto \Xi' \triangleright t' \), \( \Xi' \subseteq \Xi \) and \( \Xi'; \cdot \vdash t' : G \); or
- \( \Xi \triangleright t \mapsto \text{error} \)

**Proof.** By induction on the structure of \( t \).

- If \( t \) has some of this form: \( \epsilon_2(\epsilon_1 u : G_1) : G_2 \), \( \text{op}(\epsilon_2 u : G_1) \), \( (\lambda x : G_1, t) : G_2 \) \( \text{or} \) \( \epsilon u : G_1 \), \( \epsilon_1 u_1 : G_1 \), \( \epsilon_2 u_2 : G_2 \), \( \pi_i(\epsilon_1 u_1, u_2 : G_1 \times G_2) \) \( \text{or} \) \( \epsilon(\lambda x, t : \forall X.G) [G'] \), then by well-definedness of \( \quad \rightarrow \quad \) (Prop D.7), \( \Xi \triangleright t \mapsto \Xi' \triangleright t' \) and \( \Xi \subseteq \Xi' \) and \( \Xi'; \cdot \vdash t' : G \) or \( \Xi \triangleright t \mapsto \text{error} \).

- If \( \Xi \triangleright t \mapsto \text{error} \), then by the rule Rerr \( \Xi \triangleright t \mapsto \text{error} \) and the result holds immediately.

- If \( t = f[\ell] \), we know that \( \Xi; \cdot \vdash f[\ell] : G \) and \( \Xi; \cdot \vdash t_1 : G' \), where \( f : G' \rightarrow G \). Then, by the induction hypothesis \( \Xi \triangleright t_1 \mapsto \Xi' \triangleright t'_1 \), \( \Xi \subseteq \Xi' \) and \( \Xi' \; ; \; \cdot \vdash t'_1 : G \) or \( \Xi \triangleright t_1 \mapsto \Xi' \triangleright \text{error} \).

If \( \Xi \triangleright t_1 \mapsto \Xi' \triangleright \text{error} \), by the Rf rule the result holds.

\[\square\]

**Proposition D.9** (\( \quad \rightarrow \quad \) is well defined). If \( \Xi; \cdot \vdash t : G \), \( t \leadsto t_\epsilon \), then \( t_\epsilon \) is a value \( v \); or \( \Xi \triangleright t_\epsilon \mapsto \Xi' \triangleright t'_\epsilon \), \( \Xi' \subseteq \Xi' \) and \( \Xi'; \cdot \vdash t'_\epsilon : G \); or \( \Xi \triangleright t_\epsilon \mapsto \text{error} \).

**Proof.** By induction on the structure of \( t \), using Lemma D.8 and Canonical Forms (Lemma D.1).

\[\square\]

Now we can establish type safety of GSF: programs of GSF do not get stuck, though they may terminate with cast errors. Also the store of a program is well typed.

**Proposition D.10** (Type Safety). If \( \vdash t : G \) then either \( t \Downarrow \Xi \Downarrow v \) \text{ with } \( \Xi \Downarrow v : G \), \( t \Downarrow \text{error} \), or \( t \Downarrow \).

**Proof.** Direct by D.9.

\[\square\]
D.2 Static Terms Do Not Fail

**Lemma D.11.** If $\varepsilon_1$ and $\varepsilon_2$ two static evidences, such that $\varepsilon_1 \vdash \Sigma; \Delta \vdash T_1 \leadsto T_2$ and $\varepsilon_2 \vdash \Sigma; \Delta \vdash T_2 \leadsto T_3$, then $\varepsilon_1 \circ \varepsilon_2 = \langle p_1(\varepsilon_1), p_2(\varepsilon_2) \rangle$.

**Proof.** Straightforward induction on types $T_1, T_2, T_3$ ($\Sigma; \Delta \vdash T_2 \leadsto T_2$ coincides with $\Sigma; \Delta \vdash T_2 = T_3$), and optimality of evidences (Lemma 7.2), because the resulting evidence cannot gain precision as each component of the evidences are static (note that precision between static terms coincide with equality of static terms $\Sigma; \Delta \vdash \cdot = \cdot$).

**Lemma D.12.** Let $T_1$ and $T_2$ two static types, and $\Xi$ a static store, such that $\Xi; \Delta \vdash T_1 \leadsto T_2$. Then $\mathcal{I}(\mathcal{T}_1, \mathcal{T}_2) = \mathcal{I}(\mathcal{L}(\mathcal{T}_1), \mathcal{L}(\mathcal{T}_2)) = \langle \mathcal{L}(\mathcal{T}_1), \mathcal{L}(\mathcal{T}_2) \rangle$.

**Proof.** Straightforward induction on types $T_1, T_2$, and noticing that we cannot gain precision from the types.

**Proposition D.13 (Static terms progress and preservation).** Let $t$ be a static term, $\Xi$ a static store ($\Xi = \gamma$), and $G$ a static type ($G = \Gamma$). If $\Sigma; \cdot :: t : T$, then either $\Sigma \cdot t \leadsto \Sigma' \cdot t'$ and $\Sigma' ; \cdot :: t' : T$, for some $\Sigma'$ and $t'$ static; or $t$ is a value $v$.

**Proof.** By induction on the structure of a derivation of $\Sigma; \cdot :: t : T$.

Note that $\Xi; \Delta \vdash T_1 \leadsto T_2$ coincides with $\Xi; \Delta \vdash T_1 = T_2$, so we use the latter notation throughout the proof.

**Case (t = e u :: G).** The result is trivial as $t$ is a value.

**Case (t = (e_1(\lambda x : T_{11}, t_1) :: T_1 \rightarrow T_2) (e_2 u :: T_1)).** Then

\[
\begin{array}{c}
\Xi; \cdot :: x : T_{11} \vdash t_1 : T_{12} \\
\Xi; \cdot :: (\lambda x : T_{11}, t_1) : T_{11} \rightarrow T_{12} \\
\varepsilon_1 \vdash \Sigma; \Delta \vdash T_{11} \rightarrow T_{12} = T_1 \rightarrow T_2 \\
\Xi; \cdot :: (\varepsilon_1(\lambda x : T_{111}, t_1) :: T_1 \rightarrow T_2) : T_1 \rightarrow T_2 \\
\varepsilon_2 \vdash \Sigma; \Delta \vdash T_2' = T_1 \\
\Xi; \cdot :: (\varepsilon_2 u :: T_1) : T_2 \\
\Xi; \cdot :: (\varepsilon_1(\lambda x : T_{111}, t_1) :: T_1 \rightarrow T_2) (\varepsilon_2 u :: T_1) : T_2
\end{array}
\]

By Lemma D.11, $e' = (\varepsilon_2 \circ \text{dom}(\varepsilon_1))$ is defined and by Lemma D.12, the new evidence is also static. Then

\[
\Xi \cdot (e_1(\lambda x : T_{111}, t_1) :: T) (e_2 u :: T_1) \quad \rightarrow \quad \Xi \cdot \text{cod}(\varepsilon_1)(t_1[\varepsilon' u :: T_{111}]) \cdot x :: T_2
\]

And the result holds immediately by the Lemma D.2 and the typing rule (Easc).

**Case (t = (\varepsilon X . t_1 :: \forall X . T_x) [T']).** Then

\[
\begin{array}{c}
\Xi; \cdot :: \varepsilon : T_1 \\
\Xi; \cdot :: (\varepsilon X . t_1 :: \forall X . T_x) :: T \\
\varepsilon_1 \vdash \Sigma; \Delta \vdash [\varepsilon ; \cdot :: T_1] \forall X . T_x \\
\Xi; \cdot :: (\varepsilon X . t_1 :: \forall X . T_x) [T'] :: T_x[T'/X]
\end{array}
\]

Then

\[
(\varepsilon X . t_1 :: \forall X . T_x) [T'] \quad \rightarrow \quad \varepsilon' \cdot \varepsilon_X . T_x'[\varepsilon'[a^E'/X] :: T_x[a^E/X]) :: T_x[T'/X]
\]

where $\Xi' \equiv \Xi$, $\alpha = T'$, $\alpha \in \text{dom}(\Xi)$, and $E' \equiv \text{lift} \Xi(T')$, and

\[
(\varepsilon X . T_x) = \langle \text{lift} \Xi(T_x)[a^E/X], \text{lift} \Xi(T_x[T'/X]) \rangle.
\]

Then, $\Xi \subseteq \Xi'$, and $\Xi'$ is extended with a type name that maps to a static type. Finally, the result holds immediately by the Lemma D.4 and Lemma D.3, and the typing rule (Easc).
Case \( t = \Xi \triangleright (\varepsilon_1 u :: T_2) :: T \). Then

\[
\begin{align*}
(\text{Easc}) & \quad \Xi; \mathbf{\vdash} t : T_u \quad \varepsilon_2 \vdash \Sigma; \Delta \vdash T_u = T_2 \\
(\text{Easc}) & \quad \Xi; \mathbf{\vdash} \varepsilon_2 u :: T_2 : T_2 \\
(\text{Easc}) & \quad \Xi; \mathbf{\vdash} \varepsilon_1(\varepsilon_2 u :: T_2) :: T : T
\end{align*}
\]

By Lemma D.11, \( \varepsilon_2 \circ \varepsilon_1 \) is defined and by Lemma D.12, the new evidence is also static. Then

\[
\Xi \triangleright \varepsilon_1(\varepsilon_2 u :: T_2) :: T \quad \Rightarrow \quad \Xi \triangleright (\varepsilon_2 \circ \varepsilon_1) u :: T
\]

and the result holds by the typing rule (Easc).

Case \( t = \text{op}(\varepsilon u :: B') \). Then

\[
\begin{align*}
(\text{Easc}) & \quad \Xi; \mathbf{\vdash} t : T_u \\
(\text{Easc}) & \quad \Xi; \mathbf{\vdash} \varepsilon u :: B' :: B' \\
(\text{Easc}) & \quad \Xi; \mathbf{\triangleright} \text{ty}(\text{op}) = B' \rightarrow B \\
& \quad \Xi; \mathbf{\triangleright} \varepsilon u :: B' : B
\end{align*}
\]

Let us assume that \( \text{ty}(\text{op}) : B' \rightarrow B \). Then

\[
\Xi \triangleright \text{op}(\varepsilon u :: B') \quad \Rightarrow \quad \Xi \triangleright \varepsilon \delta(\text{op}, \text{u}) :: B
\]

And the result holds by the typing rule (Easc).

Case \( t = (\varepsilon_1 u_1 :: T_1, \varepsilon_2 u_2 :: T_2) \). Then

\[
\begin{align*}
(\text{Easc}) & \quad \Xi; \mathbf{\triangleright} \varepsilon_1 u_1 :: T_{1}' \\
(\text{Easc}) & \quad \Xi; \mathbf{\triangleright} \varepsilon_2 u_2 :: T_{2}' \\
(\text{Easc}) & \quad \Xi; \mathbf{\triangleright} \langle \varepsilon_1 u_1 :: T_{1}, \varepsilon_2 u_2 :: T_{2} \rangle :: T_1 \times T_2
\end{align*}
\]

and the result holds by the Lemma D.5.

Case \( t = \pi_i(\varepsilon (u_1, u_2) :: T) \). Then

\[
\begin{align*}
(\text{Epair}) & \quad \Xi; \mathbf{\triangleright} \langle u_1, u_2 \rangle :: T_{1}' \times T_{2}' \\
& \quad \Xi; \mathbf{\triangleright} \varepsilon (u_1, u_2) :: T \\
(\text{Epair}) & \quad \Xi; \mathbf{\triangleright} \pi_i(\varepsilon (u_1, u_2) :: T) :: \text{prof}_{i}^\#(T)
\end{align*}
\]

Then

\[
\Xi \triangleright \pi_i(\varepsilon (u_1, u_2) :: T) \quad \Rightarrow \quad \Xi \triangleright p_i(\varepsilon u_i :: \text{prof}_{i}^\#(T))
\]

And the result holds by Lemma D.6.

Case \( t = t_1 t_2 \). Then by induction hypothesis \( \Xi \triangleright t_1 \quad \Rightarrow \quad \Xi \triangleright t_{1}' \), and \( t_{1}' \) is static, and so \( t_{1}' t_2 \).

Case \( t = \nu t_2 \). Then by induction hypothesis \( \Xi \triangleright t_2 \quad \Rightarrow \quad \Xi \triangleright t_{2}' \), and \( t_{2}' \) is static, and so \( \nu t_{2}' \).

Case \( t = t_1 [T], t = \langle t_1, t_2 \rangle, t = \text{op}(\overline{t}), t = \pi_i(t_{1}) \). Similar inductive reasoning to application cases.

\[
\square
\]

**Proposition D.14 (Static terms do not fail).** Let \( t \) be a static term. If \( t : T \) then \( \neg(t \downarrow) \text{error} \).

**Proof.** Direct by Lemma D.13.
GSF: PARAMETRICITY

In this section we present the logical relation for parametricity of GSF, the proof of the fundamental property, and the soundness of the logical relation wrt contextual approximation.

E.1 Auxiliary Definitions

In this section we show function definitions used in the logical relation of GSF (Figure 6).

Definition E.1. \( \text{ev}(\varepsilon u :: G) = \varepsilon \)

Definition E.2.

\[
\begin{array}{ll}
\text{const}(E) = \\
B & E = B \\
? \rightarrow ? & E = E_1 \rightarrow E_2 \\
\forall X.? & E = \forall X. E_1 \\
? \times ? & E = E_1 \times E_2 \\
\alpha & E = \alpha E_1 \\
X & E = X \\
? & E = ?
\end{array}
\]

E.2 Fundamental Property

Theorem 8.1 (Fundamental Property). If \( \Xi; \Delta; \Gamma \vdash t : G \) then \( \Xi; \Delta; \Gamma \vdash t \leq t : G \).

Proof. By induction on the type derivation of \( t \).

Case (Easc). Then \( t = \varepsilon s :: G \), and therefore:

\[
\Xi; \Delta; \Gamma \vdash t : G' \quad \varepsilon \vdash \Xi; \Delta \vdash G' \sim G
\]

We follow by induction on the structure of \( s \).

- If \( s = b \) then:

\[
\Xi; \Delta; \Gamma \vdash b : B \quad \Xi; \Delta \vdash \varepsilon b :: G \leq \varepsilon b :: G : G
\]

Then we have to prove that \( \Xi; \Delta; \Gamma \vdash \varepsilon b :: G \), but the result follows directly by Prop E.3 (Compatibility of Constant).

- If \( s = \lambda x : G_1. t' \) then:

\[
\Xi; \Delta; \Gamma \vdash \lambda x : G_1. t' : G_2 \\
\Xi; \Delta \vdash \lambda x : G_1. t' : G_1 \rightarrow G_2
\]

Then we have to prove that:

\[\Xi; \Delta; \Gamma \vdash \varepsilon (\lambda x : G_1. t') :: G \leq \varepsilon (\lambda x : G_1. t') :: G : G\]

By induction hypotheses we already know that \( \Xi; \Delta; \Gamma', x : G_1 \vdash t' \leq t' : G_2 \). But the result follows directly by Prop E.4 (Compatibility of term abstraction).

- If \( s = \Lambda X. t' \) then:

\[
\Xi; \Delta; \Gamma \vdash \Lambda X. t' : \forall X. G^* \\
\Xi; \Delta \vdash \Lambda X. t' \quad \forall X. G^*
\]

Then we have to prove that:

\[\Xi; \Delta; \Gamma \vdash \varepsilon (\Lambda X. t') :: G \leq \varepsilon (\Lambda X. t') :: G : G\]

By induction hypotheses we already know that \( \Xi; \Delta; X; \Gamma \vdash t' \leq t' : G^* \). But the result follows directly by Prop E.5 (Compatibility of type abstraction).
• If \( s = \langle u_1, u_2 \rangle \) then:
\[
\frac{\Xi; \Delta; \Gamma \vdash u_1 : G_1 \quad \Xi; \Delta; \Gamma \vdash u_2 : G_2}{\Xi; \Delta; \Gamma \vdash \langle u_1, u_2 \rangle : G_1 \times G_2}
\]

Then we have to prove that:
\[
\Xi; \Delta; \Gamma \vdash \langle u_1, u_2 \rangle \vdash G \leq \varepsilon \langle u_1, u_2 \rangle \vdash G
\]

We know by premise that \( \Xi; \Delta; \Gamma \vdash \pi_1(u_1) : G_1 \vdash G_1 \) and \( \Xi; \Delta; \Gamma \vdash \pi_2(u_2) : G_2 \vdash G_2 \). Then by induction hypotheses we already know that: \( \Xi; \Delta; \Gamma \vdash \pi_1(u_1) \vdash G_1 \leq \pi_1(u_1) \vdash G_1 \) and \( \Xi; \Delta; \Gamma \vdash \pi_2(u_2) : G_2 \vdash G_2 \). But the result follows directly by Prop E.6 (Compatibility of pairs).

• If \( s = t' \), and therefore:
\[
\frac{\Xi; \Delta; \Gamma \vdash t' : G'}{\Xi; \Delta; \Gamma \vdash t' \vdash G'}
\]

By induction hypotheses we already know that \( \Xi; \Delta; \Gamma \vdash t' \leq t' : G' \), then the result follows directly by Prop E.9 (Compatibility of ascriptions).

Case (Epair). Then \( t = \langle t_1, t_2 \rangle \), and therefore:
\[
\frac{\Xi; \Delta; \Gamma \vdash t_1 : G_1 \quad \Xi; \Delta; \Gamma \vdash t_2 : G_2}{\Xi; \Delta; \Gamma \vdash \langle t_1, t_2 \rangle : G_1 \times G_2}
\]

where \( G = G_1 \times G_2 \). Then we have to prove that:
\[
\Xi; \Delta; \Gamma \vdash \langle t_1, t_2 \rangle \vdash G_1 \times G_2
\]

By induction hypotheses we already know that \( \Xi; \Delta; \Gamma \vdash t_1 \leq t_1 : G_1 \) and \( \Xi; \Delta; \Gamma \vdash t_2 \leq t_2 : G_2 \). But the result follows directly by Prop E.7 (Compatibility of pairs).

Case (Ex). Then \( t = x \), and therefore:
\[
\frac{x : G \in \Gamma \quad \Xi; \Delta; \Gamma \vdash x}{\Xi; \Delta; \Gamma \vdash x : G}
\]

Then we have to prove that \( \Xi; \Delta; \Gamma \vdash x \leq x : G \). But the result follows directly by Prop E.8 (Compatibility of variables).

Case (Eop). Then \( t = op(\Gamma') \), and therefore:
\[
\frac{\Xi; \Delta; \Gamma \vdash \Gamma' : \overline{G'} \quad ty(op) = \overline{G'} \rightarrow G}{\Xi; \Delta; \Gamma \vdash op(\Gamma') : G}
\]

Then we have to prove that: \( \Xi; \Delta; \Gamma \vdash op(\Gamma') \leq op(\Gamma') : G \). By the induction hypothesis we obtain that: \( \Xi; \Delta; \Gamma \vdash \overline{\Gamma'} \leq \overline{\Gamma'} : \overline{G} \). Then the result follows directly by Prop E.10 (Compatibility of app operator).

Case (Eapp). Then \( t = t_1 t_2 \), and therefore:
\[
\frac{\Xi; \Delta; \Gamma \vdash t_1 : G_{11} \rightarrow G_{12} \quad \Xi; \Delta; \Gamma \vdash t_2 : G_{11}}{\Xi; \Delta; \Gamma \vdash t_1 t_2 : G_{12}}
\]

where \( G = G_{12} \). Then we have to prove that:
\[
\Xi; \Delta; \Gamma \vdash t_1 t_2 \leq t_1 t_2 : G_{12}
\]

By the induction hypothesis we obtain that: \( \Xi; \Delta; \Gamma \vdash t_1 \leq t_1 : G_{11} \rightarrow G_{12} \) and \( \Xi; \Delta; \Gamma \vdash t_2 \leq t_2 : G_{11} \). Then the result follows directly by Prop E.11 (Compatibility of term application).
**Case (EappG).** Then \( t = t' \ [G_2] \), and therefore:

\[
\frac{\forall x. G_1}{\Xi; \Delta; \Gamma \vdash t': \forall x. G_1} \\
\frac{\Xi; \Delta; \Gamma \vdash t' [G_2]: G_1[G_2/X]}{\Xi; \Delta; \Gamma \vdash t': [G_2] : G_1[G_2/X]}
\]

where \( G = G_1[G_2/X] \). Then we have to prove that:

\[
\Xi; \Delta; \Gamma \vdash t' [G_2] \preceq t' [G_2] : G_1[G_2/X]
\]

By induction hypotheses we know that:

\[
\Xi; \Delta; \Gamma \vdash t' \preceq t' : \forall x. G_1
\]

Then the result follows directly by Prop E.12 (Compatibility of type application).

**Case (Epair1).** Then \( t = \pi_1(t') \), and therefore:

\[
\frac{\Xi; \Delta; \Gamma \vdash t': G_1 \times G_2}{\Xi; \Delta; \Gamma \vdash \pi_1(t') : G_1}
\]

where \( G = G_1 \). Then we have to prove that: \( \Xi; \Delta; \Gamma \vdash \pi_1(t') \preceq \pi_1(t') : G_1 \). By the induction hypothesis we obtain that: \( \Xi; \Delta; \Gamma \vdash t' \preceq t' : G_1 \times G_2 \). Then the result follows directly by Prop E.13 (Compatibility of access to the first component of the pair).

**Case (Epair2).** Then \( t = \pi_2(t') \), and therefore:

\[
\frac{\Xi; \Delta; \Gamma \vdash t': G_1 \times G_2}{\Xi; \Delta; \Gamma \vdash \pi_2(t') : G_2}
\]

where \( G = G_2 \). Then we have to prove that: \( \Xi; \Delta; \Gamma \vdash \pi_2(t') \preceq \pi_2(t') : G_2 \). By the induction hypothesis we obtain that: \( \Xi; \Delta; \Gamma \vdash t' \preceq t' : G_1 \times G_2 \). Then the result follows directly by Prop E.14 (Compatibility of access to the second component of the pair).

\[\square\]

**Proposition E.3 (Compatibility-\(\varepsilon b\)).** If \( b \in B, \varepsilon \vdash \Xi; \Delta \vdash B \sim G \) and \( \Xi; \Delta \vdash \Gamma \) then:

\[
\Xi; \Delta; \Gamma \vdash \varepsilon b :: G \preceq \varepsilon b :: G : G
\]

**Proof.** As \( b \) is constant then it does not have free variables or type variables, then \( b = \rho(\gamma_1(b)) \). Then we have to proof that for all \( W \in S[\Xi] \) it is true that:

\[
(W, \rho_1(\varepsilon) b :: \rho(G), \rho_2(\varepsilon) b :: \rho(G)) \in T_\rho[G]
\]

As \( \rho_1(\varepsilon) b :: G \) are values, then we have to proof that:

\[
(W, \rho_1(\varepsilon) b :: \rho(G), \rho_2(\varepsilon) b :: \rho(G)) \in \mathcal{V}_\rho[G]
\]

1. If \( G = B \), we know that \( \langle B, B \rangle = \varepsilon \vdash \Xi; \Delta \vdash B \sim B \), then \( \rho_1(\varepsilon) = \varepsilon \) and the result follows immediately by the definition of \( \mathcal{V}_\rho[B] \).

2. If \( G \in TypeName \) then \( \varepsilon = \langle H_3, a^{E_4} \rangle \). Notice that as \( a^{E_4} \) cannot have free type variables therefore \( H_3 \) neither. Then \( \varepsilon = \rho_1(\varepsilon) \). As \( a \) is sync, then let us call \( G'' = W.\Xi_1(a) \). We have to prove that:

\[
(W, \langle H_3, a^{E_4} \rangle b :: a, \langle H_3, a^{E_4} \rangle b :: a) \in \mathcal{V}_\rho[a]
\]

which, by definition of \( \mathcal{V}_\rho[a] \), is equivalent to prove that:

\[
(W, \langle H_3, E_4 \rangle b :: G'', \langle E_3, E_4 \rangle b :: G'') \in \mathcal{V}_\rho[G'']
\]

Then we proceed by case analysis on \( \varepsilon \):
• (Case $\epsilon = (H_3, \alpha^{E_3})$). We know that $\langle H_3, \alpha^{E_3} \rangle \vdash \Xi; \Delta \vdash B \sim \alpha$, then by Lemma E.30, $\langle H_3, \beta^{E_3} \rangle \vdash \Xi; \Delta \vdash B \sim \beta$. As $\beta^{E_3} \subseteq G''$, then $G''$ can either be $\alpha$ or $\beta$.

If $G'' = \alpha$, then by definition of $\mathcal{V}_\rho[\alpha]$, we have to prove that the resulting values belong to $\mathcal{V}_\rho[\beta]$. Also as $\langle H_3, \beta^{E_3} \rangle \vdash \Xi; \Delta \vdash \beta$, by Lemma E.28, $\langle H_3, \beta^{E_3} \rangle \vdash \Xi; \Delta \vdash B \sim \beta$, and then we proceed just like this case once again (this process is finite as there are no circular references by construction and it ends up in something different to a type name). If $G'' = \beta$ we use an analogous argument as for $G'' = \alpha$.

• (Case $\epsilon = (H_3, \alpha^{E_3})$). Then using similar arguments as before, we have to prove that

$$(W, (H_3, H_4) b :: G'', (H_3, H_4) b :: G'') \in \mathcal{V}_\rho[\mathcal{V}_\rho]'$$

By Lemma E.30, $\langle H_3, H_4 \rangle \vdash \Xi; \Delta \vdash B \sim G''$. Then if $G'' = \alpha$, we proceed as the case $G = \alpha$, with the evidence $\epsilon = (H_3, H_4)$. If $G''$ is HeadType, we proceed as the previous case where $G = B$, and the evidence $\epsilon = (H_3, H_4)$.

(3) If $G = \alpha$ we have the following cases:

• $(G = ?; \epsilon = (H_3, H_4))$. By the definition of $\mathcal{V}_\rho[\alpha]$ in this case we have to prove that:

$$(W, \rho_1(\epsilon) b :: const(H_4), \rho_2(\epsilon) b :: const(H_4)) \in \mathcal{V}_\rho[const(H_4)]$$

but as $\text{const}(H_4) = B$ (note that $H_3 = B$ then since $H_4$ is HeadType has to be $B$). The theorem follows immediately since is part of the premise.

• $(G = ?; \epsilon = (H_3, \alpha^{E_3}))$. Notice that as $\alpha^{E_3}$ cannot have free type variables therefore $E_3$ neither. Then $\epsilon = \rho_1(\epsilon)$. By the definition of $\mathcal{V}_\rho[\alpha]$ we have to prove that

$$(W, (H_3, \alpha^{E_3}) u_1 :: \alpha, (H_3, \alpha^{E_3}) u_2 :: \alpha) \in \mathcal{V}_\rho[\alpha]$$

Note that by Lemma E.28 we know that $\epsilon \vdash \Xi; \Delta \vdash B \sim \alpha$. Then we proceed just like the case $G \in \text{Type Name}$.

\[\Box\]

**Proposition E.4 (Compatibility-El)**. If $\Xi; \Delta; \Gamma \vdash t \leq t' : G_2$, $\epsilon \vdash \Xi; \Delta \vdash G_1 \rightarrow G_2 \sim G$ then:

$$\Xi; \Delta; \Gamma \vdash \epsilon(\lambda x : G_1, t) :: G \leq \epsilon(\lambda x : G_1, t') :: G : G$$

**Proof**. First, we are required to show that $\Xi; \Delta; \Gamma \vdash \epsilon(\lambda x : G_1, t) :: G : G$ and $\Xi; \Delta; \Gamma \vdash \epsilon(\lambda x : G_1, t') :: G : G$, which follow from $\epsilon \vdash \Xi; \Delta \vdash G_1 \rightarrow G_2 \sim G$ and $\Xi; \Delta; \Gamma \vdash \lambda x : G_1, t : G_1 \rightarrow G_2$ and $\Xi; \Delta; \Gamma \vdash \lambda x : G_1, t' : G_1 \rightarrow G_2$ respectively, which follow (respectively) from $\Xi; \Delta; \Gamma \vdash t : G_2$ and $\Xi; \Delta; \Gamma \vdash t' : G_2$, which follow from $\Xi; \Delta; \Gamma \vdash \epsilon(\lambda x : G_1, t) :: G$ and $\Xi; \Delta; \Gamma \vdash \epsilon(\lambda x : G_1, t') :: G$.

Consider arbitrary $W, \rho, \gamma$ such that $W \in S[\Xi], (W, \rho) \in D[\Delta]$ and $(W, \gamma) \in G_1, G_2 \vdash G$. We are required to show that:

$$(W, \rho(\gamma_1(\epsilon(\lambda x : G_1, t) :: G))), \rho(\gamma_2(\epsilon(\lambda x : G_1, t) :: G))) \in \mathcal{T}_\rho[G]$$

Consider arbitrary $i, u_1$ and $\Xi_1$ such that $i < W.j$ and:

$$(W, \Xi_1 \triangleright \rho(\gamma_1(\epsilon(\lambda x : G_1, t) :: G))) \rightarrow^i \Xi_1 \triangleright u_1$$

Since $\rho(\gamma_1(\epsilon(\lambda x : G_1, t) :: G)) = \rho(\gamma_1(\epsilon(\lambda x : G_1, t))), \rho(\gamma_1(\epsilon(\lambda x : G_1, t))) :: \rho(G)$ and $\rho(\gamma_1(\epsilon(\lambda x : G_1, t))) :: \rho(G)$ is already a value, where $\rho_1(\epsilon) = \rho_1(\epsilon), we have i = 0 and u_1 = \rho(\gamma_1(\epsilon(\lambda x : G_1, t))) :: \rho(G)$ and $\Xi_1 = W.\Xi_1$. Since $\rho_2(\epsilon(\lambda x : G_1, t)) :: \rho(G)$ is already a value, we are required to show that $\exists W', \exists j + i = W.j, W' \geq W, W_1.\Xi_1 = \Xi_1, W'.\Xi_2 = \Xi_2$ and:

$$(W', \rho(\epsilon(\lambda x : G_1, t))) :: \rho(G), \rho(\epsilon(\lambda x : G_1, t))) :: \rho(G) \in \mathcal{V}_\rho[G]$$
Let \( W' = W \), then we have to show that:
\[
(W, e'^1_1(\lambda x : \rho(G_1), \rho(y_1(t)))) :: \rho(G), e'^0_2(\lambda x : \rho(G_1), \rho(y_2(t')) :: \rho(G)) \in \mathcal{V}_\rho[G]
\]

First we have to prove that:
\[
W.\Xi_1; \Delta; \Gamma \vdash e'^0_2(\lambda x : \rho(G_1), \rho(y_1(t))) :: \rho(G) : \rho(G)
\]

As we know that \( \Xi_1; \Delta; \Gamma \vdash e(\lambda x : G_1.t) :: G \), by Lemma E.26 the result follows immediately. The case \( W.\Xi_2; \Delta; \Gamma \vdash e'^0_2(\lambda x : \rho(G_1), \rho(y_2(t'))) :: \rho(G) : \rho(G) \) is similar.

The type \( G \) can be \( G'_1 \rightarrow G'_2 \), for some \( G'_1 \) and \( G'_2 \), or \( ? \) or a \textsc{TypeName}.

1. \( G = G'_1 \rightarrow G'_2 \), we are required to show that \( \forall W'', v_1' = e'_1 u'_1 :: \rho(G'_1), v_2' = e'_2 u'_2 :: \rho(G'_1) \), such that \( W'' \geq W' \) and \( (W'', v_1', v_2') \in \mathcal{V}_\rho[G'_1] \), it is true that:
\[
(W'', e'^0_1(\lambda x : \rho(G_1), \rho(y_1(t))) :: \rho(G'_1 \rightarrow G'_2) v_1', e'^0_2(\lambda x : \rho(G_1), \rho(y_2(t'))) :: \rho(G'_1 \rightarrow G'_2) v_2') \in \mathcal{T}_\rho[G'_2]
\]

If \( (e'_1 \circ \text{dom}(e'^0_1)) \) fails, then by Lemma E.27 \( (e'_2 \circ \text{dom}(e'^0_2)) \) and the result follows immediately.

Else, if \( (e'_1 \circ \text{dom}(e'^0_1)) \) follow we have to prove that:

\[
( \downarrow W'', \text{cod}(e'^0_1)(\rho(y_1(t)))(e'_1 \circ \text{dom}(e'^0_1)) u'_1 :: \rho(G_1)/x : \rho(G_1))) :: \rho(G'_2), \\
\text{cod}(e'^0_2)(\rho(y_2(t')))(e'_2 \circ \text{dom}(e'^0_2)) u'_2 :: \rho(G_1)/x : \rho(G_1)) :: \rho(G'_2) \in \mathcal{T}_\rho[G'_2]
\]

Note that \( \text{dom}(e'^0_1) + W''.\Xi_1 \vdash (\rho(G'_1) \sim \rho(G'_1)) \). By the Lemma E.18 ( with the type \( G_1 \) and the evidences \( \text{dom}(e'^0_1) + W''.\Xi_1 + \rho(G'_1) \sim \rho(G'_1) \)) it is true that:
\[
(W'', \text{dom}(e'^0_1) v_1' :: G_1, \text{dom}(e'^0_2) v_2' :: G_1) \in \mathcal{T}_\rho[G_1]
\]

Since \( (e'_1 \circ \text{dom}(e'^0_1)) \) does not fail, it is true that:
\[
(W'', (e'_1 \circ \text{dom}(e'^0_1)) u'_1 :: G_1, (e'_2 \circ \text{dom}(e'^0_2)) u'_2 :: G_1) \in \mathcal{V}_\rho[G_1]
\]

We instantiate the hypothesis \( \Xi_1; \Delta; \Gamma \vdash t \leq t' : G_2 \), with \( W''', \rho \) and \( \gamma[x : \rho(G_1) \rightarrow (v'_1, v'_2)] \), where \( v'_1 = (e'_1 \circ \text{dom}(e'^0_1)) u'_1 :: \rho(G_1) \). Note that \( S[\Xi_1] \ni W''' \geq W \) by the definition of \( S[\Xi] \), \( (W''', \rho) \in \mathcal{D}[\Delta] \) by the definition of \( \mathcal{D}[\Delta] \) and \( (W'', \gamma[x \mapsto (v'_1, v'_2)]) \) in \( \mathcal{G}_\rho[\Gamma, x : \rho(G_1)] \), which follow from: \( (W'', \gamma) \in \mathcal{G}_\rho[\Gamma] \) and \( (W'', v_1', v_2') \in \mathcal{V}_\rho[G_1] \) which follows from above. Then, we have that:
\[
(W'', \rho(y_1(t)))[v'_1/x], \rho(y_2(t'))[v'_2/x]) \in \mathcal{T}_\rho[G_2]
\]

If the following term reduces to error, then the result follows immediately.
\[
W'''.\Xi_1 \vdash \rho(y_1(t))[v'_1/x]
\]

If the above is not true, then the following terms reduce to values \( (v_1f, v_2f) \) and \( \exists W'''' \geq W'' \) such that \( (W''', v_1f, v_2f) \in \mathcal{V}_\rho[G_2] \).

\[
W''.\Xi_1 \vdash \rho(y_1(t))[v'_1/x] \longrightarrow^* W'''.\Xi_1 \triangleright v_1f
\]

\[
W''.\Xi_2 \vdash \rho(y_2(t'))[v'_2/x] \longrightarrow^* W'''.\Xi_2 \triangleright v_2f
\]

We instantiate the induction hypothesis in the previous result with the type \( G_2 \) and the evidence \( \text{cod}(e'^0_1) + W''.\Xi_1 \vdash G'' \sim G'_2 \), then we obtain that:
\[
(\uparrow W'', \text{cod}(e'^0_1) v_1f :: \rho(G'_2), \text{cod}(e'^0_2) v_2f :: \rho(G'_2)) \in \mathcal{T}_\rho[G'_2]
\]

and the result follows immediately.
Let $u_1 = \lambda x : \rho(G_1).\rho(y_1(t)), u_2 = \lambda x : \rho(\rho(G_1).\rho(y_2(t'))) and G^* = G_1 \rightarrow G_2$, we have to prove that:

$$(W, \rho_1(\varepsilon)u_1 :: \rho(G), \rho_2(\varepsilon)u_2 :: \rho(G)) \in \mathcal{V}_\rho[G]$$

(2) If $G \in \text{TypeName}$ then $\varepsilon = (H_3, \alpha^{E_i})$. Notice that as $\alpha^{E_i}$ cannot have free type variables therefore $H_3$ neither. Then $\varepsilon = \rho_i(\varepsilon)$. As $\alpha$ is sync, then let us call $G'' = W.\exists_\varepsilon(\alpha)$. We have to prove that:

$$(W, (H_3, \alpha^{E_i}) u_1 :: \alpha, (H_3, \alpha^{E_i}) u_2 :: \alpha) \in \mathcal{V}_\rho[\alpha]$$

which, by definition of $\mathcal{V}_\rho[\alpha]$, is equivalent to prove that:

$$(W, (H_3, E_4) u_1 :: G'', (E_3, E_4) u_2 :: G'') \in \mathcal{V}_\rho[G'']$$

Then we proceed by case analysis on $\varepsilon$:

- (Case $\varepsilon = (H_3, \alpha^{B^{E_i}})$). We know that $(H_3, \alpha^{B^{E_i}}) + \Xi; \Delta + G^* \sim \alpha$, then by Lemma E.30, $(H_3, \beta^{E_i}) + \Xi; \Delta + G^* \sim \beta$. As $\beta^{E_i} \subseteq G''$, then $G''$ can either be $? \tau$ or $\beta$.

If $G'' = ? \tau$, then by definition of $\mathcal{V}_\rho[? \tau]$, we have to prove that the resulting values belong to $\mathcal{V}_\rho[?]$. Also as $(H_3, \beta^{E_i}) + \Xi; \Delta + G^* \sim \beta$, by Lemma E.28, $(H_3, \beta^{E_i}) + \Xi; \Delta + G^* \sim \beta$, and then we proceed just like this case once again (this is process is finite as there are no circular references by construction and it ends up in something different to a type name). If $G'' = \beta$ we use an analogous argument as for $G'' = ? \tau$.

- (Case $\varepsilon = (H_3, \alpha^{M^{E_i}})$). Then using similar arguments as before, we have to prove that:

$$(W, (H_3, H_4) u_1 :: G'', (H_3, H_4) u_2 :: G'') \in \mathcal{V}_\rho[G'']$$

By Lemma E.30, $(H_3, H_4) + \Xi; \Delta + G^* \sim G''$. Then if $G'' = ? \tau$, we proceed as the case $G = ? \tau$, with the evidence $\varepsilon = (H_3, H_4)$. If $G'' \in \text{HeadType}$, we proceed as the previous case where $G = G_1' \rightarrow G_2'$, and the evidence $\varepsilon = (H_3, H_4)$.

(3) If $G = ? \tau$ we have the following cases:

- $(G = ? \tau, \varepsilon = (H_3, H_4))$. By the definition of $\mathcal{V}_\rho[? \tau]$ in this case we have to prove that:

$$(W, \rho_1(\varepsilon)u_1 :: \rho(G), \rho_2(\varepsilon)u_2 :: \rho(G)) \in \mathcal{V}_\rho[\text{const}(H_4)]$$

but as $\text{const}(H_4) = ? \tau \rightarrow ? \tau$, we proceed just like this case where $G = G_1' \rightarrow G_2'$, where $G_1' = ? \tau$ and $G_2' = ? \tau$.

- $(G = ? \tau, \varepsilon = (H_3, \alpha^{E_i}))$. Notice that as $\alpha^{E_i}$ cannot have free type variables therefore $E_3$ neither. Then $\varepsilon = \rho_i(\varepsilon)$. By the definition of $\mathcal{V}_\rho[? \tau]$ we have to prove that:

$$(W, (H_3, \alpha^{E_i}) u_1 :: \alpha, (H_3, \alpha^{E_i}) u_2 :: \alpha) \in \mathcal{V}_\rho[\alpha]$$

Note that by Lemma E.28 we know that $\varepsilon + \Xi; \Delta + G^* \sim \alpha$. Then we proceed just like the case $G \in \text{TypeName}$.

\[ \square \]

**Proposition E.5 (Compatibility-\(\text{EA}\)).** If $\exists; \Delta; X + t_1 \leq t_2 : G, \varepsilon + \exists; \Delta + \forall X.G \sim G'$ and $\exists; \Delta + \Gamma$ then $\exists; \Delta; \Gamma + \varepsilon(\Lambda X.t_1) :: G' \leq \varepsilon(\Lambda X.t_2) :: G'$.\]

**Proof.** First, we are required to prove that $\exists; \Delta; \Gamma + \varepsilon(\Lambda X.t_1) :: G' : G'$, but by unfolding the premises we know that $\exists; \Delta; X + t_1 : G$, therefore:

$$\exists; \Delta; X; \Gamma + t_1 : G \quad \exists; \Delta + \Gamma \quad \exists; \Delta; \Gamma + \Lambda X.t_1 \in \forall X.G$$
Then we can conclude that:

\[ \Xi; \Delta; \Gamma \vdash \forall X. t_i \in \forall X.G \quad \varepsilon \vdash \Xi; \Delta \vdash \forall X.G \sim G' \]

\[ \Xi; \Delta; \Gamma \vdash \varepsilon(\forall X. t_i) : G' : G' \]

Consider arbitrary \( W, \rho, \gamma \) such that \( W \in S[\Xi], (W, \rho) \in D[\Delta] \) and \((W, \gamma) \in G_\rho[\Gamma]\). We are required to show that:

\( (W, \rho(\gamma_1(\varepsilon(\forall X. t_i) :: G'))), \rho(\gamma_2(\varepsilon(\forall X. t_i) :: G'))) \in T_\rho[G'] \)

First we have to prove that:

\( W.\Xi_i \vdash \rho(\gamma_1(\varepsilon(\forall X. t_i) :: G')) : \rho(G') \)

As we know that \( \Xi; \Delta; \Gamma \vdash \varepsilon(\forall X. t_i) :: G' : G' \), by Lemma E.26 the result follows immediately.

By definition of substitutions \( \rho(\gamma_1(\varepsilon(\forall X. t_i) :: G')) = \varepsilon_1(\forall X. \rho(\gamma_1(t_i))) :: \rho(G') \), where \( \varepsilon_1 = \rho_1(\varepsilon) \), therefore we have to prove that:

\( (W, \varepsilon_1^t(\forall X. \rho(\gamma_1(t_i)))) :: (G'), \varepsilon_2^t(\forall X. \rho(\gamma_2(t_i)))) :: \rho(G') \) \( \in T_\rho[G'] \)

We already know that both terms are values and therefore we only have to prove that:

\( (W, \varepsilon_1^t(\forall X. \rho(\gamma_1(t_i)))) :: (G'), \varepsilon_2^t(\forall X. \rho(\gamma_2(t_i)))) :: \rho(G') \) \( \in V_\rho[G'] \)

The type \( G' \) can be \( \forall X.G'_i \), for some \( G'_i \), or a \( \text{TYPENAME} \). Let \( u_1 = \forall X. \rho(\gamma_1(t_i)) \), \( u_2 = \forall X. \rho(\gamma_2(t_i)) \) and \( G^* = \forall X.G \), we have to prove that:

\( (W, \rho_1(\varepsilon)u_1 :: \rho(G), \rho_2(\varepsilon)u_2 :: \rho(G)) \in V_\rho[G'] \)

(1) If \( G' = \forall X.G'_i \), then consider \( W' \subseteq W \), and \( G_1, G_2, R \) and \( \alpha \), such that \( W'.\Xi_i \vdash G_i \), and \( R \in \text{REL}_{W'.\Xi_i}[G_1, G_2] \).

\( W'.\Xi_i, \alpha := G_i \vdash E_i^{E_i} [\alpha/E_i] :: \forall X. \rho(G'_i) \) \( [G_i] \rightarrow \)

\[ W'.\Xi_i, \alpha := G_i \vdash E_i^{E_i} [\alpha/E_i] :: \forall X. \rho(G'_i) \] \( [G_i] \rightarrow \)

\[ \text{where } E_i' = \text{lift}(W.\Xi_i)(G_i) \]

Note that \( \varepsilon \vdash \Xi; \Delta \vdash \forall X.G \sim \forall X.G'_i \), then \( \varepsilon = \langle \forall X.E_1, \forall X.E_2 \rangle \), for some \( E_1, E_2, K \) and \( L \). By the Lemma E.25 we know that \( \varepsilon_1^t = W.\Xi_i; \Delta \vdash \forall X. \rho(G) \sim \forall X. \rho(G'_i) \), then \( \varepsilon_1^t = \langle \forall X.E_1, \forall X.E_2 \rangle \), where \( \forall X. E_1 = \rho_1(\varepsilon_1) \) and \( E_2 = \rho_1(\varepsilon_2) \).

Then we have to prove that:

\( (W', (\rho_1^t(\alpha/E_i])\rho(\gamma_1(t_i))[\alpha/E_i/X] :: (G'_i)\alpha/X)](\alpha/E_i)](\alpha/E_i/X), \rho_1(\varepsilon)u_1 :: \rho(G'_i)[\alpha/X]) \in T_{\rho[\alpha/X]}[G'_i] \)

where \( W' = \downarrow (W' \Box (\alpha, G_1, G_2, R)) \).

Let \( \rho' = \rho[\alpha/X] \). We instantiate the premise \( \Xi; \Delta; \Gamma \vdash t_i \leq t_2 : G \) with \( W'', \rho' \) and \( \gamma \), such that \( W'' \subseteq S[\Xi], \alpha \in \text{dom}(W''[\alpha/X \mapsto R]) \) then \( (W'', \rho') \in D[\Delta, X] \). Also note that as \( X \) is fresh, then \( (v_1', v_2') \in \text{cod}(\gamma) \), such that \( \Xi; \Delta; \Gamma \vdash v_i' : G^*, X \notin \text{FV}(G^*) \), then it is easy to see that \( (W'', \gamma) \in G_\rho[\alpha/X][\Gamma] \). Then we know that:

\( (W'', \rho' \gamma_1(t_i)), \rho' \gamma_2(t_i)) \in T_\rho[G] \)

But note that:

\( \rho'(\gamma_1(t_i)) = \rho[\alpha/X](\gamma_1(t_i)) = \rho(\gamma_1(t_i))[\alpha/E_i/X] \)

Then we have that:

\( (W'', \rho(\gamma_1(t_i))[\alpha/E_i/X], \rho(\gamma_2(t_i))[\alpha/E_i/X]) \in T_{\rho[\alpha/X]}[G] \)
If the following term reduces to error, then the result follows immediately.

$$W'' \Xi_1 \triangleright \rho(y_1(t_1))[\alpha^{E_1}/X]$$

If the above is not true, then the following terms reduce to values \((v_{1f}, v_{2f})\) and \(\exists W''' \ni W''\) such that \((W'''', v_{1f}, v_{2f}) \in V_\rho[\alpha \rightarrow X][\bar{G}]\).

\[
W'''\Xi_1 \triangleright \rho(y_1(t_1))[\alpha^{E_1}/X] \longrightarrow^* W''''\Xi_1 \triangleright v_{1f}
\]

We instantiate the Lemma \(E.18\) with the type \(G'_1\) and the evidence \(\langle E_1, E_2 \rangle \triangleright \Xi; \Delta, X \mapsto G \sim G'_1\) (remember that \(\varepsilon = \langle \forall X.E_1, \forall X.E_2 \rangle\)). Note that \(\varepsilon_1^\rho \left[\alpha^{E_1}\right] = \rho[X \mapsto \alpha]_{W'' \Xi_1}(\langle E_1, E_2 \rangle), \rho[X \mapsto \alpha](G'_1)\) = \(\rho(G'_1)[\alpha/X]\), \(W''' \in S[\Xi]\) and \((W''', \rho[X \mapsto \alpha]) \in D[\Delta, X]\). Then we obtain that:

\[
\langle W'''', \varepsilon_1^\rho \left[\alpha^{E_1}\right] v_{1f} : \rho(G'_1)[\alpha/X], (v_{2f})^\rho \left[\alpha^{E_1}\right] v_{2f} : \rho(G'_1)[\alpha/X] \rangle \in T_\rho[G'_1]
\]

and the result follows immediately.

(2) If \(G' \in \text{TypeName}\) then \(\varepsilon = \langle H_\varepsilon, \alpha^{E_\varepsilon}\rangle\). Notice that as \(\alpha^{E_\varepsilon}\) cannot have free type variables therefore \(H_\varepsilon\) neither. Then \(\varepsilon = \rho_1(\varepsilon)\). As \(\varepsilon\) is sync, then let us call \(G'' = W.\Xi_1(\alpha)\). We have to prove that:

\[
(W, \langle H_\varepsilon, \alpha^{E_\varepsilon} \rangle u_1 : \alpha, \langle H_\varepsilon, \alpha^{E_\varepsilon} \rangle u_2 : \alpha) \in V_\rho[\alpha]
\]

which, by definition of \(V_\rho[\alpha]\), is equivalent to prove that:

\[
(W, \langle H_\varepsilon, \alpha^{E_\varepsilon} \rangle u_1 : G'', \langle E_3, E_4 \rangle u_2 : G'') \in V_\rho[G'']
\]

Then we proceed by case analysis on \(\varepsilon\):

- (Case \(\varepsilon = \langle H_\varepsilon, \alpha^{E_\varepsilon} \rangle\)). We know that \(\langle H_\varepsilon, \alpha^{E_\varepsilon} \rangle \triangleright \Xi; \Delta \mapsto \alpha\), then by Lemma \(E.30\), \(\langle H_\varepsilon, \beta^{E_\varepsilon} \rangle \triangleright \Xi; \Delta \mapsto \beta\). As \(\beta^{E_\varepsilon} \sqsubseteq G''\), then \(G''\) can either be \(?\) or \(\beta\).

If \(G'' = \?\), then by definition of \(V_\rho[?]\), we have to prove that the resulting values belong to \(V_\rho[\beta]\). Also as \(\langle H_\varepsilon, \beta^{E_\varepsilon} \rangle \triangleright \Xi; \Delta \mapsto \beta\), by Lemma \(E.28\), \(\langle H_\varepsilon, \beta^{E_\varepsilon} \rangle \triangleright \Xi; \Delta \mapsto \beta\), and then we proceed just like this case once again (this is process is finite as there are no circular references by construction and it ends up in something different to a type name). If \(G'' = \beta\) we use an analogous argument as for \(G'' = \?\).

- (Case \(\varepsilon = \langle H_\varepsilon, \alpha^{E_\varepsilon} \rangle\)). Then using similar arguments as before, we have to prove that:

\[
(W, \langle H_3, H_4 \rangle u_1 : G'', \langle H_3, H_4 \rangle u_2 : G'') \in V_\rho[G'']
\]

By Lemma \(E.30\), \(\langle H_3, H_4 \rangle \triangleright \Xi; \Delta \mapsto \alpha\). Then if \(G'' = \?\), we proceed as the case \(G'' = \?\), with the evidence \(\varepsilon = \langle H_3, H_4 \rangle\). If \(G'' \in \text{HeadType}\), we proceed as the previous case where \(G'' = \forall X.G\), and the evidence \(\varepsilon = \langle H_3, H_4 \rangle\).

(3) If \(G' = \?\) we have the following cases:

- (\(G' = \?, \varepsilon = \langle H_3, H_4 \rangle\)). By the definition of \(V_\rho[?]\) in this case we have to prove that:

\[
(W, \rho_1(\varepsilon) u_1 : \rho(G), \rho_2(\varepsilon) u_2 : \rho(G)) \in V_\rho[\text{const}(H_3)]
\]

but as \(\text{const}(H_3) = \forall X.\?\), we proceed just like the case where \(G' = \forall X.G'_1\), where \(G'_1 = \?\).

- (\(G' = \?, \varepsilon = \langle H_3, \alpha^{E_\varepsilon} \rangle\)). Notice that as \(\alpha^{E_\varepsilon}\) cannot have free type variables therefore \(E_3\) neither. Then \(\varepsilon = \rho_1(\varepsilon)\). By the definition of \(V_\rho[?]\) we have to prove that:

\[
(W, \langle H_3, \alpha^{E_\varepsilon} \rangle u_1 : \alpha, \langle H_3, \alpha^{E_\varepsilon} \rangle u_2 : \alpha) \in V_\rho[\alpha]
\]
Note that by Lemma E.28 we know that \( \varepsilon \vdash \Xi; \Delta \vdash G' \sim \alpha \). Then we proceed just like the case \( G' \in \text{TypeName} \).

\[ \square \]

**Proposition E.6 (Compatibility-EPAIRU).** If \( \Xi; \Delta; \Gamma \vdash \pi_1(\varepsilon)u_1 :: G_1 \leq \pi_1(\varepsilon)u'_1 :: G_1 : G_1, \Xi; \Delta; \Gamma \vdash \pi_2(\varepsilon)u'_2 :: G_2 \leq \pi_2(\varepsilon)u''_2 :: G_2 : G_2, \) and \( \varepsilon \vdash \Xi; \Delta \vdash G_1 \times G_2 \sim G \) then:

\[ \Xi; \Delta; \Gamma \vdash \varepsilon(u_1, u_2) :: G \leq \varepsilon(u'_1, u''_2) :: G : G \]

**Proof.** Straightforward as the definition of related pairs depends on a weaker property of the premise: \( \Xi; \Delta; \Gamma \vdash t_1 \leq t'_1 :: G_1 \) and \( \Xi; \Delta; \Gamma \vdash t_2 \leq t'_2 :: G_2 \), then \( \Xi; \Delta; \Gamma \vdash \langle t_1, t_2 \rangle \leq \langle t'_1, t'_2 \rangle : G_1 \times G_2 \).

\[ \square \]

**Proposition E.7 (Compatibility-EPAIR).** If \( \Xi; \Delta; \Gamma \vdash t_1 \leq t'_1 :: G_1 \) and \( \Xi; \Delta; \Gamma \vdash t_2 \leq t'_2 :: G_2 \), then \( \Xi; \Delta; \Gamma \vdash \langle t_1, t_2 \rangle \leq \langle t'_1, t'_2 \rangle : G_1 \times G_2 \).

**Proof.** We proceed by induction on subterms \( t_i \), analogous to the function application case, but using Prop E.6 instead.

\[ \square \]

**Proposition E.8 (Compatibility-EX).** If \( x : G \in \Gamma \) and \( \Xi; \Delta \vdash \Gamma \) then \( \Xi; \Delta ; \Gamma \vdash x \leq x : G \).

**Proof.** First, we are required to show \( \Xi; \Delta; \Gamma \vdash x : G \), which is immediate. Consider arbitrary \( W, \rho, \gamma \) such that \( W \in S[\Xi], (W, \rho) \in D[\Delta] \) and \( (W, \gamma) \in G_\rho[\Gamma] \). We are required to show that:

\[ (W, \rho(\gamma_1(x)), \rho(\gamma_2(x))) \in T_\rho[G] \]

Consider arbitrary \( i, v_1 \) and \( \Xi_1 \) such that \( i < W.j \) and \( W.\Xi_1 \not\succ \rho(\gamma_1(x)) \rightarrow i \Xi_1 \not\succ v_1 \). Since \( \rho(\gamma_1(x)) = \gamma_1(x) \) and \( \gamma_1(x) \) is already a value, we have \( i = 0 \) and \( \gamma_1(x) = v_1 \). We are required to show that exists \( \Xi_2, v_2 \) such that \( W.\Xi_2 \not\succ \gamma_2(x) \rightarrow v \Xi_2 \not\succ v_2 \) which is immediate (since \( \rho(\gamma_2(x)) = \gamma_2(x) \) is a value and \( \Xi_2 = W.\Xi_2 \)). Also, we are required to show that \( \exists W', \) such that \( W'.j + i = W.j \wedge W' \geq W \wedge W'.\Xi_1 = \Xi_1 \wedge W'.\Xi_2 = \Xi_2 \wedge (W', \gamma_1(x), \gamma_2(x)) \in \mathcal{V}_\rho[G] \). Let \( W' = W \), then \( (W, \gamma_1(x), \gamma_2(x)) \in \mathcal{V}_\rho[G] \) because of the definition of \( (W, \gamma) \in G_\rho[\Gamma] \).

\[ \square \]

**Proposition E.9 (Compatibility-ESC).** If \( \Xi; \Delta; \Gamma \vdash t_1 \leq t_2 : G \) and \( \varepsilon \vdash \Xi; \Delta \vdash G' \sim G' \) then \( \Xi; \Delta; \Gamma \vdash \epsilon t_1 :: G' \leq \epsilon t_2 :: G' : G' \).

**Proof.** First we are required to prove that \( \Xi; \Delta; \Gamma \vdash \epsilon t_1 :: G' : G' \), but by \( \Xi; \Delta; \Gamma \vdash t_1 \leq t_2 : G \) we already know that \( \Xi; \Delta; \Gamma \vdash t_1 : G \), therefore:

\[ (\text{Esc}) \Xi; \Delta; \Gamma \vdash t_1 : G \quad \varepsilon \vdash \Xi; \Delta \vdash G' \sim G' \]

\[ \Xi; \Delta; \Gamma \vdash \epsilon t_1 :: G' : G' \]

Consider arbitrary \( W, \rho, \gamma \) such that \( W \in S[\Xi], (W, \rho) \in D[\Delta] \) and \( (W, \gamma) \in G_\rho[\Gamma] \). We are required to show that:

\[ (W, \rho(\gamma_1(\varepsilon t_1 :: G'))), \rho(\gamma_2(\varepsilon t_2 :: G')) \in T_\rho[G'] \]

But by definition of substitutions \( \rho(\gamma_1(\varepsilon t_1 :: G')) = \rho(\varepsilon)\rho(\gamma_1(t_1)) : \rho(G') \), therefore we have to prove that:

\[ (W, \rho(\varepsilon)\rho(\gamma_1(t_1)) : \rho(G'), \rho(\varepsilon)\rho(\gamma_2(t_2)) : \rho(G')) \in T_\rho[G'] \]

First we have to prove that:

\[ W.\Xi_1 \vdash \rho(\varepsilon)\rho(\gamma_1(t_1)) : \rho(G') : G' \]

As we know that \( \Xi; \Delta; \Gamma \vdash \epsilon t_1 :: G' : G' \), by Lemma E.26 the result follows immediately.
Second, consider arbitrary \( i < W.j, \Xi_1 \). Either there exist \( v_1 \) such that:

\[
W.\Xi_1 \triangleright \rho(\varepsilon)\rho(y_1(t_1^G)) :: \rho(G') \mapsto \Xi_1 \triangleright v_1
\]
or

\[
W.\Xi_1 \triangleright \rho(\varepsilon)\rho(y_1(t_1^G)) :: \rho(G') \mapsto \Xi_1 \triangleright \text{error}
\]

Let us suppose that \( W.\Xi_1 \triangleright \rho(\varepsilon)\rho(y_1(t_1^G)) :: \rho(G') \mapsto \Xi_1 \triangleright v_1 \). Hence, by inspection of the operational semantics, it follows that there exist \( i_1 + 1 < i, \Xi_1 \) and \( v_1 \) such that:

\[
W.\Xi_1 \triangleright \rho(\varepsilon)\rho(y_1(t_1^G)) :: \rho(G') \mapsto i_1 \Xi_1 \triangleright \rho(\varepsilon)v_1 :: \rho(G')
\]

We instantiate the hypothesis \( \Xi ; \Delta ; \Gamma \vdash t_1^G \leq t_2^G : G \) with \( W, \rho \) and \( \varepsilon \) to obtain that:

\[
(W, \rho(\varepsilon)\rho(y_1(t_1^G)), \rho(y_2(t_2^G))) \in \mathcal{T}_\rho[G]
\]

We instantiate \( \mathcal{T}_\rho[G] \) with \( i_1, \Xi_1 \) and \( v_1 \) (note that \( i_1 < i < W.j \)), hence there exists \( v_1 \) and \( W_1 \), such that \( W_1 \geq W, W_1 . j = W . j - i_1 \), \( W . \Xi_2 \triangleright \rho(y_2(t_2^G)) \mapsto W_1 . \Xi_2 \triangleright v_1 ; W_1 . \Xi_1 = \Xi_1, v_1 \) and \( (W_1, v_1, v_1) \) \( \in \mathcal{V}_\rho[G] \).

Since we have that \( (W_1, v_1, v_1) \in \mathcal{V}_\rho[G] \), then it is true that \( (W_1, \rho(\varepsilon)v_1 :: G', \rho(\varepsilon)v_2 :: G') \in \mathcal{T}_\rho[G'] \) by the Lemma E.18.

By the inspection of the operational semantics:

\[
W.\Xi_1 \triangleright \rho(\varepsilon)\rho(y_1(t_1^G)) :: \rho(G') \mapsto i_1 \Xi_1 \triangleright \rho(\varepsilon)v_1 :: \rho(G') \mapsto \Xi_1 \triangleright v_1
\]

We instantiate \( (W_1, \rho(\varepsilon)v_1 :: G', \rho(\varepsilon)v_2 :: G') \in \mathcal{T}_\rho[G'] \) with \( 1, v_1 \) and \( \Xi_1 \). Therefore there must exist \( v_2 \) and \( W' \) such that \( W' \geq W_1 \) (note that \( W' \geq W \)), \( W' . j = W . j - (i_1 - 2) = W . j - i_1 - 1 = W . j - i \),

\[
W_1 . \Xi_2 \triangleright \rho(\varepsilon)v_2 :: \rho(G') \mapsto \Xi_2 \triangleright v_2
\]

and \( (W', v_1, v_2) \in \mathcal{V}_\rho[G'] \) then the result follows. \( \square \)

**Proposition E.10 (Compatibility-Eop).** If \( \Xi ; \Delta ; \Gamma \vdash \bar{t} \leq \bar{t}' : \overline{G} \) and \( ty(op) = \overline{G} \rightarrow G \) then \( \Xi ; \Delta ; \Gamma \vdash op(\bar{t}) \leq op(\bar{t}') : G \).

**Proof.** Similar to the term application. \( \square \)

**Proposition E.11 (Compatibility-Eapp).** If \( \Xi ; \Delta ; \Gamma \vdash t_1 \leq t_1' : G_1 \rightarrow G_2 \) and \( \Xi ; \Delta ; \Gamma \vdash t_2 \leq t_2' : G_1 \) then \( \Xi ; \Delta ; \Gamma \vdash t_1 \ t_2 \leq t_1' \ t_2' : G_2 \).

**Proof.** First, we are required to show that:

\[
\Xi ; \Delta ; \Gamma \vdash t_1 \ t_2 : G_2
\]

which follows directly from (Eapp) as \( \Xi ; \Delta ; \Gamma \vdash t_1 : G_1 \), and \( \Xi ; \Delta ; \Gamma \vdash t_2 : G_2 \). Also, we are required to prove that:

\[
\Xi ; \Delta ; \Gamma \vdash t_1' \ t_2' : G_2
\]

which follows analogously.

Second, consider \( \Delta \) and \( \Gamma \) such that \( \Gamma \supseteq \text{FV}(t_1 \ t_2) \), and \( \Gamma \supseteq \text{FV}(t_1' \ t_2') \), and consider arbitrary \( W, \rho, \gamma \) such that \( W \in \text{S}[\Xi], (W, \rho) \in \text{D}[\Delta] \) and \( (W, \gamma) \in \text{G}_\rho[\Gamma] \). We are required to show that:

\[
(W, \rho(y_1(t_1 \ t_2)), \rho(y_2(t_1' \ t_2'))) \in \mathcal{T}_\rho[G_2]
\]

Consider arbitrary \( i, v_1 \) and \( \Xi_1 \) such that \( i < W.j \) and:

\[
W.\Xi_1 \triangleright \rho(y_1(t_1 \ t_2)) \mapsto \Xi_1 \triangleright v_1 \lor W.\Xi_1 \triangleright \rho(y_1(t_1 \ t_2)) \mapsto \text{error}
\]
Hence, by inspection of the operational semantics, it follows that there exist $i_1 < i$, $\Xi_{11}$ and $v_{11}$ such that:

$$W.\Xi_1 \triangleright \rho(y_1(t_1)) \rightarrow^{i_1} \Xi_{11} \triangleright v_{11} \lor W.\Xi_1 \triangleright \rho(y_1(t_1)) \rightarrow^{i_1} \text{error}$$

If $W.\Xi_1 \triangleright \rho(y_1(t_1)) \rightarrow^{i_1} \text{error}$ then $W.\Xi_1 \triangleright \rho(y_1(t_1')) \rightarrow^{i_1} \text{error}$ and the result holds immediately. Let us assume that the reduction do not fail. We instantiate the hypothesis $\Xi; \Delta; \Gamma \vdash t_1 \leq t_1': G_{11} \rightarrow G_{12}$ with $W$, $\rho$ and $\gamma$ we obtain that:

$$(W, \rho(y_1(t_1))), \rho(y_2(t_1')) \in T_\rho[G_{11} \rightarrow G_{12}]$$

We instantiate this with $i_1$, $\Xi_{11}$ and $v_{11}$ (note that $i_1 < i < W.j$), hence there exists $v_{11}'$ and $W_1$, such that $W_1 \geq W$, $W_1.j = W.j - i_1$, $W.\Xi_2 \triangleright \rho(y_2(t_1')) \rightarrow^* W_1.\Xi_2 \triangleright v_{11}'$, $W_1.\Xi_1 = \Xi_{11}$ and $(W_1, v_{11}, v_{11}') \in \mathcal{V}_\rho[G_{11} \rightarrow G_{12}]$.

Note that:

$$W.\Xi_1 \triangleright \rho(y_1(t_1, t_2)) \rightarrow^{i_1} \Xi_{11} \triangleright v_{11}(\rho(y_1(t_2))) \rightarrow^{i-\xi_1} \Xi_1 \triangleright v_1$$

or

$$W.\Xi_1 \triangleright \rho(y_1(t_1, t_2)) \rightarrow^{i_1} \Xi_{11} \triangleright v_{11}(\rho(y_1(t_2))) \rightarrow^{i-\xi_1} \text{error}$$

Hence, by inspection of the operational semantics, it follows that there exist $i_2 < i - i_1$, $\Xi_{22}$ and $v_{22}$ such that:

$$\Xi_{11} \triangleright \rho(y_1(t_2)) \rightarrow^{i_2} \Xi_{22} \triangleright v_{22} \lor \Xi_{11} \triangleright \rho(y_1(t_2)) \rightarrow^{i_2} \text{error}$$

We instantiate the hypothesis $\Xi; \Delta; \Gamma \vdash t_2 \leq t_2': G_{11}$ with $W_1$, $\rho$ and $\gamma$, then we obtain that:

$$(W_1, \rho(y_1(t_2)), \rho(y_2(t_2'))) \in T_\rho[G_{12}]$$

If $\Xi_{11} \triangleright \rho(y_1(t_2)) \rightarrow^{i_2} \text{error}$ then we instantiate with $\Xi_{22}$ and $\Xi_{22} \triangleright \rho(y_1(t_2')) \rightarrow^{i_1} \text{error}$ and the result holds immediately. Let us assume that the reduction do not fail. We instantiate this with $i_2$ (note that $i_2 < i - i_1 < W'.j = W.j - i_1'$), $\Xi_{22}$ and $v_{22}$, hence there exists $v_{22}'$ and $W_2$, such that $W_2.\Xi_1 = \Xi_{22}$, $W_2 \geq W_1$, $W_2.j = W_1.j - i_2$ and

$$W_1.\Xi_2 \triangleright \rho(y_2(t_2')) \rightarrow^* W_2.\Xi_2 \triangleright v_{22}'$$

and $(W_2, v_{22}, v_{22}') \in \mathcal{V}_\rho[G_{11}]$.

Note that:

$$W.\Xi_1 \triangleright \rho(y_1(t_1, t_2)) \rightarrow^{i_1} \Xi_{11} \triangleright v_{11}(\rho(y_1(t_2))) \rightarrow^{i_2} \Xi_{22} \triangleright v_{11} \lor v_{22} \rightarrow^{i-\xi_1} \Xi_1 \triangleright v_1$$

Since $(W_1, v_{11}, v_{11}') \in \mathcal{V}_\rho[G_{11} \rightarrow G_{12}]$, we instantiate this with $W_2$, $\rho(G_{11} \rightarrow G_{12})$, $v_{22}$ and $v_{22}'$ (note that $(W_2, v_{22}, v_{22}') \in \mathcal{V}_\rho[G_1]$ and $W_2 \geq W_1$). Then $(W_2, v_{11} v_{22}, v_{11}', v_{22}') \in T_\rho[G_2]$.

Since $(W_2, v_{11} v_{22}, v_{11}', v_{22}') \in T_\rho[G_2]$, we instantiate this with $i - i_1 - i_2$ (note that $i - i_1 - i_2 < W_2.j = W.j - i_1 - i_2$ since $i < W.j$), $v_1$ and $\Xi_1$.

If $W_2.\Xi_1 \triangleright v_{11} v_{22} \rightarrow^{i-\xi_1} \text{error}$ then $W_2.\Xi_2 \triangleright v_{11}' v_{22}' \rightarrow^* \text{error}$ and the result holds. Let us assume that the reduction does not fail. Hence there exists $v_2$ and $W'$, such that $W' \geq W$ (note that $W' \geq W$), $W'.j = W_2.j - (i - i_1 - i_2) = W.j - i$, $W_2.\Xi_2 \triangleright v_{11}' v_{22}' \rightarrow^* W'.\Xi_2 \triangleright v_2$, $W'.\Xi_1 = \Xi_1$ and $(W_2, v_1, v_2) \in \mathcal{V}_\rho[T_2]$, then the proof is complete. 

**Proposition E.12 (Compatibility-EAPP).** If $\Xi; \Delta; \Gamma \vdash t_1 \leq t_2 : \forall X . G$ and $\Xi; \Delta \vdash G'$, then $\Xi; \Delta; \Gamma \vdash t_1[G'] \leq t_2[G'] : G[G'/X]$. 


(EappG) \[ \Xi; \Delta; \Gamma \vdash t_1 [G'] : G[G'/X] \]

Consider arbitrary \( W, \rho, \gamma \) such that \( W \in \mathcal{S}[\Xi], (W, \rho) \in \mathcal{D}[\Delta] \) and \((W, \gamma) \in \mathcal{G}_\rho[\Gamma]\). We are required to show that:

\[ (W, \rho(y_1(t_1[G'])), \rho(y_2(t_2[G']))) \in \mathcal{T}_\rho[\mathcal{G}[G'/X]] \]

But by definition of substitutions \( \rho(y_1(t_1[G'])) = \rho(y_1(t_1)[\rho(G')]) \), therefore we have to prove that:

\[ (W, \rho(y_1(t_1))[\rho(G')], \rho(y_2(t_2))[\rho(G')]) \in \mathcal{T}_\rho[\mathcal{G}[G'/X]] \]

First we have to prove that:

\[ W.\Xi_1 \triangleright \rho(y_1(t_1))[\rho(G')] : \rho(G)[\rho(G')/X] \]

As we know that \( \Xi; \Delta; \Gamma \vdash t_1 \leq t_2 : \forall X. G \) we already know that \( \Xi; \Delta; \Gamma \vdash t_1 : \forall X. G \), therefore:

\[ (W.\Xi_1 \triangleright \rho(y_1(t_1))[\rho(G')] \triangleright^i \Xi_1 \triangleright v_1 \]

Hence, by inspection of the operational semantics, it follows that there exist \( i + 1 < i, \epsilon_1 \) and \( v_{11} \) such that:

\[ W.\Xi_1 \triangleright \rho(y_1(t_1))[\rho(G')] \triangleright^i \Xi_1 \triangleright v_{11}[\rho(G')] \]

We instantiate the premise \( \Xi; \Delta; \Gamma \vdash t_1 \leq t_2 : \forall X. G \) with \( W, \rho \) and \( \gamma \) to obtain that:

\[ (W, \rho(y_1(t_1)), \rho(y_2(t_2))) \in \mathcal{T}_\rho[\forall X. G] \]

We instantiate \( \mathcal{T}_\rho[\forall X. G] \) with \( i_1, \Xi_{11} \) and \( v_{11} \) (note that \( i_1 < i < W.j \)), hence there exists \( v_{12} \) and \( W_1 \), such that \( W_1 \geq W, W_1.j = W.j - i_1, W.\Xi_2 \triangleright \rho(y_2(t_2)) \triangleright^* W_1.\Xi_2 \triangleright v_{12}, W_1.\Xi_1 = \Xi_{11}, v_{12} \) and:

\[ (W_1, v_{11}, v_{12}) \in \mathcal{T}_\rho[\forall X. G] \]

Then by inspection of the operational semantics:

\[ W.\Xi_1 \triangleright \rho(y_1(t_1))[\rho(G')] \triangleright^* W_1.\Xi_1 \triangleright v_{11}[\rho(G')] \]

\[ \triangleright^i W_1.\Xi_1, \alpha : \rho(G') \triangleright \epsilon_1(\epsilon'_1 t'_1 : \rho(G)[\alpha/X]) : \rho(G)[\rho(G')/X] \]

for some \( \epsilon_1, \epsilon_2, \epsilon'_1, t'_1 \) and \( \alpha \notin \text{dom}(W_1.\Xi_i) \). Let us call \( t''_1 = (\epsilon'_1 t'_1 : \rho(G)[\alpha/X]) \). We instantiate \( \mathcal{T}_\rho[\forall X. G] \) with \( \alpha, t''_1, \rho(G'), R = \mathcal{V}_\rho[G'], \epsilon_1, \epsilon_2 \) and \( W_1 \). Then \((W_1, t''_1, t''_1) \in \mathcal{T}_{\rho[\forall X \rightarrow \alpha]}[G] \), where \( W_1 = (\downarrow W_1) \boxtimes (\alpha, \rho(G'), \rho(G'), \mathcal{V}_\rho[G']) \).

We instantiate \( \mathcal{T}_{\rho[\forall X \rightarrow \alpha]}[G] \) with \( i_2 = i - i_1 - 2 \) (note that \( i - i_1 - 2 < W'.j = W.j - i_1 - 2 < W.j \)), \( \Xi_1, v'_1 \) such that:

\[ W_1.\Xi_1 \triangleright (\epsilon'_1 t'_1 : \rho(G)[\alpha/X]) \triangleright^i \Xi_1 \triangleright (\epsilon'_1 v''_1 : \rho(G)[\alpha/X]) \triangleright^i \Xi_1 \triangleright v'_1 \]

for some \( v''_1 \). Therefore there must exist \( v''_2 \), and \( W' \) such that \( W' \geq W_i \) (note that \( W' \geq W_i \)), \( W', j = W'.j - (i - i_1 - 2) = W.j - i \),

\[ W_1.\Xi_2 \triangleright (\epsilon'_2 t'_2 : \rho(G)[\alpha/X]) \triangleright^* W_1.\Xi_2 \triangleright (\epsilon'_2 v''_2 : \rho(G)[\alpha/X]) \triangleright^i W'.\Xi_2 \triangleright v'_2 \]

for some \( v''_2 \), \( W'.\Xi_1 = \Xi_1 \) and \((W', v'_1, v''_2) \in \mathcal{V}_\rho[\forall X \rightarrow \alpha][G] \).

Notice that \( t_i \) reduce to a type abstraction of the form \( v_{11} = (\forall X. E_{i1}, \forall X. E_{i2}) \forall X. t''_i : \forall X. \rho(G) \). Let us call \( v'_1 = \epsilon''_{i1} u''_{i2} : \rho(G)[\alpha/X] \), as \( \pi_2(\epsilon''_{i1}) \equiv \pi_2(\epsilon''_{i2}) \), then \( G_p = \text{unlift}(\pi_2(\epsilon''_{i1})) \), then
Let us call \( v_1 = (\epsilon''_1 \circ \epsilon_1)u''_1 \models \rho(G)[\rho(G')/X] \). Where the theorem holds by instantiating \( T_\rho[G[G'/X]] \) with \( \Xi_1, v_1 \), \( i = i_1 + i_2 + 2 \) and therefore \( W'.\Xi_1 \triangleright v_1' \models \rho(G)[\rho(G')/X] \). Then there must exists some \( v_2 \) such that \( W'.\Xi_2 \triangleright v_2' \models \rho(G)[\rho(G')/X] \). and the result follows.

Now let us suppose that \( W.\Xi_1 \triangleright v(\gamma_1(t_1))[\rho(G')] \ models \( \Xi_1 \triangleright \text{error} \). We instantiate the hypothesis \( \Xi ; \Delta; \Gamma \vdash t_1 \leq t_2 : \forall x. G \) with \( W, \rho \) and \( \gamma \) to obtain that \( (W, \rho(\gamma_1(t_1)), \rho(\gamma_2(t_2))) \in T_\rho[\forall x. G]. \) If \( W.\Xi_1 \triangleright v(\gamma_1(t_1))[\rho(G')] \ models \( \Xi_1 \triangleright \text{error} \), for some \( \Xi_1 \) and \( i_1 < W.j \) then \( W.\Xi_2 \triangleright v(\gamma_2(t_2))[\rho(G')] \ models \( \text{error} \), for some \( \Xi_2 \) , and the result follows immediately.

If not, then there exists some \( i_1, \Xi_{11} \) and \( v_{11} \), and therefore there exists \( v_{12} \) and \( W_1 \), such that \( W.\Xi_2 \triangleright v(\gamma_2(t_2)) \ models \( \Xi_2 \triangleright \text{error} \). Then by inspection of the operational semantics:

\[
W.\Xi_i \triangleright v(\gamma_i(t_i))[\rho(G')] \models \Xi_i \triangleright v_i[\rho(G')]
\]

\[
\models W_i.\Xi_i, \alpha := \rho(G) \models \epsilon_i t_i \models \rho(G')[\rho(G')/X]
\]

for some \( \epsilon_i, t_i, \alpha \notin \text{dom}(W_i, \Xi_i) \).

We instantiate \( T_\rho[\forall x. G] \) with \( v(\gamma_1) \), \( t', \rho(G'), V_\rho[G'], \epsilon_1 \) and \( \downarrow W_1 \). Then \( (W_1', t_1', t_2') \in T_\rho[\forall x. G] \), where \( W_1' = (\downarrow W_1) \models (\alpha, \rho(G'), \rho(G'), V_\rho[G']). \)

Then if \( W_i.\Xi_1 \models t_i' \models t_2 \) \text{error} for some \( t_2 < W_i'.j \), then \( W_i.\Xi_2 \models t_2' \models \text{error} \) and the result follows immediately. \( \Box \)

**Proposition E.13 (Compatibility-Epair1).** If \( \Xi ; \Delta; \Gamma \vdash t_1 \leq t_2 : G_1 \times G_2 \) then \( \Xi; \Delta; \Gamma \vdash \pi_1(t_1) \leq \pi_2(t_2) : G_1. \)

**Proof.** Similar to the function application case, using the definition of related pairs instead. \( \Box \)

**Proposition E.14 (Compatibility-Epair2).** If \( \Xi ; \Delta; \Gamma \vdash t_1 \leq t_2 : G_1 \times G_2 \) then \( \Xi; \Delta; \Gamma \vdash \pi_2(t_1) \leq \pi_2(t_2) : G_2. \)

**Proof.** Similar to the function application case, using the definition of related pairs instead. \( \Box \)

**Lemma E.15.** Let \( E_i = \text{lift}_{E_i}(G_p) \) for some \( G_p \subseteq G, \langle E_{11}, E_{12} \rangle \vdash \Xi_i \vdash G \sim G, \text{ and } E_{12} \equiv E_{22}, \text{then } \langle E_{11}, E_{12} \rangle \circ \langle E_{11}, E_i \rangle \iff \langle E_{21}, E_{22} \rangle \circ \langle E_{21}, E_i \rangle. \)

**Proof.** Note that by definition \( E_1 \equiv E_2 \). Also, \( \forall \alpha^E \in \text{FTN}(E_i), E = \text{lift}_{E_i}(\Xi_i(\alpha)). \) Then we prove the \( \Rightarrow \) direction (the other is analogous), by induction on the structure of the evidences \( \langle E_{11}, E_{12} \rangle. \) We skip cases where \( E_i = ? \) or \( E_{11} = ?, \) as the result is trivial (combination never fails).

**Case** \( \langle E_{11}, E_{12} \rangle = \langle E_{11}, \alpha^{E_{11}} \rangle. \) Then \( E_{21}, E_{22} \rightarrow \langle E_{21}, \alpha^{E_{21}} \rangle, \) and \( E_i \rightarrow \langle \alpha^{E_i}, \alpha^{E_i} \rangle, \) where \( E_i^1 = \text{lift}_{E_i}(\Xi_i(\alpha)), \) and therefore \( E_{12}^1 \subseteq E_i^1. \) And then by Lemma E.31, the result holds immediately as both combinations are defined.

**Case** \( \langle E_{11}, E_{12} \rangle = \langle E_{11}, B \rangle. \) Then \( E_{21}, E_{22} \rightarrow \langle E_{12}, B \rangle, \) and \( E_i \rightarrow \langle B, B \rangle, \) and the result trivially holds.

**Case** \( \langle E_{11}, E_{12} \rangle = \langle \alpha^{E_i}, E_{12} \rangle. \) The result holds by de inspection of consistent transitivity rule (sealR) and induction on evidence \( \langle E_{i1}, E_{12} \rangle. \)
Case \( \langle E_{11}, E_{12} \rangle = \langle E_{111} \rightarrow E_{112}, E_{121} \rightarrow E_{122} \rangle \). Then \( \langle E_{11}, E_{12} \rangle = \langle E_{111} \rightarrow E_{112}, E_{121} \rightarrow E_{122} \rangle \), and \( \langle E_{11}, E_{1} \rangle = \langle E'_{11} \rightarrow E'_{12}, E'_{1} \rightarrow E'_{12} \rangle \). As consistent transitivity is a symmetric relation, then the result holds by induction hypothesis on combinations of evidence \( \langle E_{111} \rightarrow E_{112} \rangle \circ \langle E'_1, E'_1 \rangle \) and \( \langle E_{121} \rightarrow E_{122} \rangle \circ \langle E'_{12}, E'_{12} \rangle \).

For the other cases we proceed analogous to the function case. \( \square \)

**Proposition E.16 (Compositionality).** If

- \( W.\Xi_i(\alpha) = \rho(G') \) and \( W.\kappa(\alpha) = \mathcal{V}_p[G'] \),
- \( E_i' = \text{lift}_{W.\Xi_i}(\rho(G')) \),
- \( E_i = \text{lift}_{W.\Xi_i}(G_p) \) for some \( G_p \subseteq \rho(G) \),
- \( \epsilon_i = \langle E_i[\alpha^E_i/X], E_i[E'_i/X] \rangle \), such that \( \epsilon_i \vdash W.\Xi_i \vdash \rho(G[\alpha/X]) \sim \rho(G[G'/X]) \), and
- \( \epsilon_i^{-1} = \langle E_i[E'_i/X], E_i[\alpha^E_i/X] \rangle \), such that \( \epsilon_i^{-1} \vdash W.\Xi_i \vdash \rho(G[G'/X]) \sim \rho(G[\alpha/X]) \), then

\[
\begin{align*}
(1) & \quad (W, \epsilon'_1 u_1 : \rho'(G), \epsilon'_2 u_2 : \rho'(G)) \in \mathcal{V}_p'[G] \Rightarrow \\
(2) & \quad (W, \epsilon'_1 u_1 : \rho(G[X/X]), \epsilon'_2 u_2 : \rho(G[G'/X])) \in \mathcal{V}_p'[G[G'/X]] \Rightarrow \\
& \quad (W, \epsilon'_1 -1 \epsilon'_1 u_1 : \rho(G[G'/X]), \epsilon'_2 -1 \epsilon'_2 u_2 : \rho(G[G'/X])) : \rho'(G) \in \mathcal{T}_p[G]
\end{align*}
\]

**Proof.** We proceed by induction on \( G \). Let \( v_i = \epsilon'_i u_i : \rho'(G) \). We prove (1) first.

**Case (G = X).** Let \( v_i = \langle H_{11}, \alpha^E_{11} \rangle u_i : \alpha \). Then we know that

\[
(W, \langle H_{11}, \alpha^E_{11} \rangle u_1 : \alpha, \langle H_{21}, \alpha^E_{21} \rangle u_2 : \alpha) \in \mathcal{V}_p[X \rightarrow \alpha][X]
\]

which is equivalent to

\[
(W, \langle H_{11}, \alpha^E_{11} \rangle u_1 : \alpha, \langle H_{21}, \alpha^E_{21} \rangle u_2 : \alpha) \in \mathcal{V}_p[X \rightarrow \alpha][\alpha]
\]

As \( W.\Xi_i(\alpha) = \rho(G') \) and \( W.\kappa(\alpha) = \mathcal{V}_p[G'] \), we know that:

\[
(W, \langle H_{11}, E_{12} \rangle u_1 : \rho(G'), \langle H_{21}, E_{22} \rangle u_2 : \rho(G')) \in \mathcal{V}_p[G']
\]

Then \( \epsilon_i \vdash W.\Xi_i \vdash \alpha \sim \rho(G') \), and \( \epsilon_i \) has to have the form \( \epsilon_i = \langle \alpha^E_i, E'_i \rangle \). As \( E'_i = \text{lift}_{W.\Xi_i}(\rho(G')) \) (initial evidence for \( \alpha \)), then \( E_{12} \subseteq E'_i \), and therefore by Lemma E.31: \( \langle H_{11}, \alpha^E_{11} \rangle \circ \langle \alpha^E_i, E'_i \rangle = \langle H_{11}, E_{12} \rangle \), and then we have to prove that

\[
(\downarrow W, \langle H_{11}, E_{12} \rangle u_1 : \rho(G'), \langle H_{21}, E_{22} \rangle u_2 : \rho(G')) \in \mathcal{V}_p[G']
\]

which we already know, and the result holds.

**Case (G = Y).** Let \( v_i = \langle H_{11}, \beta^E_{11} \rangle u_i : \beta \), where \( \rho'(Y) = \beta \). Then we know that

\[
(W, \langle H_{11}, \beta^E_{11} \rangle u_1 : \beta, \langle H_{21}, \beta^E_{21} \rangle u_2 : \beta) \in \mathcal{V}_p[X \rightarrow \alpha][Y]
\]

which is equivalent to

\[
(W, \langle H_{11}, \beta^E_{11} \rangle u_1 : \beta, \langle H_{21}, \beta^E_{21} \rangle u_2 : \beta) \in \mathcal{V}_p[X \rightarrow \alpha][\beta]
\]
Then $\varepsilon_1 \vdash W.\Xi_1 \vdash \beta \sim \beta$, and $\varepsilon_1$ has to have the form $\varepsilon_1 = \langle \beta^F_i , \beta^E_i \rangle$, and $\beta^F_i = \text{lift}_{W,\Xi_1}(\beta)$. By Lemma E.15, we assume that both combinations of evidence are defined (otherwise the result holds immediately):

$$\langle H_{i1}, \beta^{E_{1z}} \rangle \circ \langle \beta^{E_i}, \beta^{E_i} \rangle = \langle H_{i1}, \beta^{E_{1z}} \rangle$$

Then we have to prove that

$$\langle \downarrow W, \langle H_{i1}, \beta^{E_{1z}} \rangle u_1 : \beta, \langle H_{i1}, \beta^{E_{1z}} \rangle u_2 : \beta \rangle \in \mathcal{V}_{\rho}[\beta]$$

which we already know by Lemma E.20, and the result holds.

Case $G = \emptyset$. Let $v_1 = \langle H_{i1}, E_{i2} \rangle u_1 : \emptyset$. Then by definition of $\mathcal{V}_{\rho}[\emptyset]$, let $G'' = \text{const}(E_{i2})$ (where $G'' \neq \emptyset$). Then we know

$$(W, \langle H_{i1}, E_{i2} \rangle u_1 : G'', \langle H_{i1}, E_{i2} \rangle u_2 : G'' \rangle) \in \mathcal{V}_{\rho}[G'']$$

If $\varepsilon_1 = \langle ?, ? \rangle$, then, by Lemma E.18, the result holds immediately. If $\varepsilon_1 = \langle E_i, E_i \rangle$, where $E_i \neq \emptyset$, then we proceed similar to the other cases where $G \neq \emptyset$.

Case $(G = G_1 \to G_2)$. We know that

$$(W, v_1, v_2) \in \mathcal{V}_{\rho}[G_1 \to G_2]$$

Then we have to prove that

$$\langle \downarrow W, (\varepsilon'_1 \circ \varepsilon_1)(\lambda x : G'_1.t_1) : \rho(G_1[G'/X]) \to \rho(G_2[G'/X]) \rangle$$

$$\langle \varepsilon'_2 \circ \varepsilon_2)(\lambda x : G'_2.t_2) : \rho(G_1[G'/X]) \to \rho(G_2[G'/X]) \rangle \in \mathcal{V}_{\rho}[G_1[G'/X] \to G_2[G'/X]]$$

Let us call $\nu'' = (\varepsilon'_1 \circ \varepsilon_1)(\lambda x : G'_1.t_1) : \rho'(G_1) \to \rho'(G_2)$. By unfolding, we have to prove that

$$\forall W'. W. \forall v'_1, v'_2, (W', v'_1, v'_2) \in \mathcal{V}_{\rho}[G_1[G'/X]] \Rightarrow (W', v'_1, v''_1, v''_2) \in \mathcal{T}_{\rho}[G_2[G'/X]]$$

Suppose that $\nu'_j = \varepsilon''_j u'_j : \rho(G_1[G'/X])$, by inspection of the reduction rules, we know that

$$(W'. \Xi_1 \vdash v''_j v'_j \leftarrow W'. \Xi_1 \vdash (\text{cod}(\varepsilon'_1) \circ \text{cod}(\varepsilon_1)) t_1 \{(\varepsilon'' \circ (\text{dom}(\varepsilon_1) \circ \text{dom}(\varepsilon'_1)) u'_j : G'_1) / x \} : \rho(G_2[G'/X])$$

This is equivalent by Lemma E.19.

$$(W'. \Xi_1 \vdash v''_j v'_j \leftarrow W'. \Xi_1 \vdash (\text{cod}(\varepsilon'_1) \circ \text{cod}(\varepsilon_1)) t_1 \{(\varepsilon'' \circ (\text{dom}(\varepsilon_1) \circ \text{dom}(\varepsilon'_1)) u'_j : G'_1) / x \} : \rho(G_2[G'/X])$$

Notice that $\text{dom}(\varepsilon_1) \vdash W. \Xi_1 \vdash \rho(G_1[G'/X]) \sim \rho(G_1[\alpha/X])$, by Lemma E.15, we assume that both combinations of evidence are defined (otherwise the result holds immediately), then let us assume that $(\varepsilon'_1 \circ \text{dom}(\varepsilon_1))$ is defined. We can use induction hypothesis on $\nu'_j$, with evidences $\text{dom}(\varepsilon_1)$. Then we know that $(\downarrow W', (\text{cod}(\varepsilon'_1) \circ \text{dom}(\varepsilon_1)) u'_j : \rho'(G_1), (\varepsilon''_j \circ \text{dom}(\varepsilon)_2) u'_j : \rho'(G_1)) \in \mathcal{V}_{\rho}[G_1]$. Let us call $\nu''_j = (\varepsilon''_j \circ \text{dom}(\varepsilon_1)) u'_j : \rho'(G_1)$.

Now we instantiate

$$(W, v_1, v_2) \in \mathcal{V}_{\rho}[G_1 \to G_2]$$

with $W'$ and $v''_j$, to obtain that either both executions reduce to an error (then the result holds immediately), or $\exists W''$ such that $(W'', v_{f_1}, v_{f_2}) \in \mathcal{V}_{\rho}[G_2]$

$$W'. \Xi_1 \vdash v_1 v''_j \leftarrow W'. \Xi_1 \vdash \text{cod}(\varepsilon_1) t_1 \{(\varepsilon'' \circ (\text{dom}(\varepsilon_1)) \circ \text{dom}(\varepsilon'_1)) u'_j : G'_1) / x \} : \rho'(G_2)$$

$$\leftarrow \ast W''. \Xi_1 \vdash v_{f_1}$$

Suppose that $v_{f_1} = \varepsilon_{f_1} u_{f_1} : \rho'(G_2)$. Then we use induction hypothesis once again using evidences $\text{cod}(\varepsilon_1)$ over $v_{f_1}$ (noticing that by Lemma E.15, the combination of evidence either both fail or both are defined), to obtain that,

$$\langle \downarrow W'', (\varepsilon_{f_1} \circ \text{cod}(\varepsilon_1)) u_{f_1} : \rho(G_2[G'/X]), (\varepsilon_{f_2} \circ \text{cod}(\varepsilon_2)) u_{f_2} : \rho(G_2[G'/X]) \rangle \in \mathcal{V}_{\rho}[G_2[G'/X]]$$

and the result holds.
Case $\forall Y.G_1$. We know that

$$ (W, v_1, v_2) \in V_\rho[\forall Y.G_1] $$

Then we have to prove that

$$ \langle \downarrow W, (\epsilon'_1 \circ \epsilon_1)(\lambda Y.t_1) :: \forall Y.\rho(G_1[G'/X]), (\epsilon'_2 \circ \epsilon_2)(\lambda Y.t_2) :: \forall Y.\rho(G_1[G'/X]) \rangle \in V_\rho[\forall Y.G_1[G'/X]] $$

Let $\epsilon'_1 = \langle \forall Y.E_{i1}, \forall Y.E_{i2} \rangle$ and $\epsilon_1 = \langle \forall Y.E'_{i1}, \forall Y.E'_{i2} \rangle = \langle \forall Y.E'_i[\alpha E_i/X], \forall Y.E''_i[E'_i/X] \rangle$, where $E_i = \forall Y.E''_i$. Let us call $v''_i = (\epsilon'_1 \circ \epsilon_1)(\lambda Y.t_1) :: \forall Y.\rho(G_1[G'/X])$. By unfolding, we have to prove that

$$ \forall W' \geq \downarrow W.\forall t''_1, t''_2, G'_1, G'_2, \beta, \epsilon'_1, \epsilon'_2, \forall \epsilon i \in \text{REL}_{W', \beta}[G'_1, G'_2]. $$

$$ (W'.\exists \Xi_1 \vdash G'_1) \wedge W'.\exists \Xi_2 \vdash G'_2 \wedge $$

$$ W'.\exists \Xi_1 \triangleright v''_1[G'_1] \rightarrow W'.\exists \Xi_1, \beta := G'_1 \triangleright \epsilon'_1 \triangleright t''_1 :: \rho(G_1[G'/X][G'_1/Y]) \wedge $$

$$ W'.\exists \Xi_2 \triangleright v''_2[G'_2] \rightarrow W'.\exists \Xi_2, \beta := G'_2 \triangleright \epsilon'_2 \triangleright t''_2 :: \rho(G_1[G'/X][G'_2/Y]) \Rightarrow $$

$$ (\downarrow (W' \boxtimes (\beta, G'_1, G'_2, R), t''_1, t''_2)) \in T_\rho[\beta \Rightarrow \beta][G_1[G'/X]] $$

By inspection of the reduction rules we know that

$$ t''_i = ((E_i[\beta E_i/Y], E_{i2}[\beta E_i/Y]) \circ (E'_i[\alpha E_i/X][\beta E_i/Y], E''_i[E'_i/X][\beta E_i/Y]))t_i[\beta E_i/Y] :: \rho(G_1[G'/X][\beta/Y]) $$

By the reduction rule of the type application we know that:

$$ (W', \exists \Xi_1, \beta := G'_1 \triangleright \epsilon'_1 \triangleright t''_1 :: \rho(G_1[G'/X][G'_1/Y]) ) $$

where $E'_i = \text{lift}_{W'.\exists \Xi_1}(G'_i)$, and $t'_i = ((E_i[\beta E_i/Y], E_{i2}[\beta E_i/Y])t_i[\beta E_i/Y] :: \rho(G_1[G'/X][\beta/Y]))$. Now we instantiate

$$ (W, v_1, v_2) \in V_\rho[\forall Y.G_1] $$

with $W', G'_1, G'_2, R, v'_i, t'_i, \beta$, and evidences $\langle E_i[\beta E_i/Y], E_{i2}[E'_i/Y] \rangle$, to obtain that

$$ (W', t'_i, t'_2) \in T_\rho[\beta \Rightarrow \beta][G_1] $$

where $W'' = \downarrow (W' \boxtimes (\beta, G'_1, G'_2, R)$ then either both executions reduce to an error (then the result holds immediately), or $\exists W'' \geq W''$, such that $(W'', v_{f1}, v_{f2}) \in V_\rho[\beta \Rightarrow \beta][G_1]$ and

$$ W''.\exists \Xi \triangleright ((E_i[\beta E_i/Y], E_{i2}[\beta E_i/Y])t_i[\beta E_i/Y] :: \rho'(G_1[\beta/Y])) $$

$$ \Rightarrow v''_{f1} \triangleright ((E_i[\beta E_i/Y], E_{i2}[\beta E_i/Y])v_{mi} :: \rho'(G_1[\beta/Y])) $$

$$ \Rightarrow W''.\exists \Xi \triangleright v_{f1} $$

Suppose that $v_{f1} = (\epsilon_{f1} \circ (\langle E_{i1}[\beta E_i/Y], E_{i2}[\beta E_i/Y] \rangle)u_{f1} :: \rho'(G_1[\beta/Y])$. As $E_{i2}[\beta E_i/Y] \equiv E_{22}[\beta E_i/Y]$, then $\text{unlift}(E_{i2}[\beta E_i/Y]) = \text{unlift}(E_{22}[\beta E_i/Y])$. Then we use induction hypothesis using $\rho'(Y \Rightarrow \beta)$, evidences $\langle E''_i[E'_i/Y], E''_i[E'_i/Y] \rangle$, where $E''_i[E'_i/Y] = \text{lift}_{W''.\exists \Xi_1}(\text{unlift}(E_{i2}[\beta E_i/Y]))$ as $E_i = \forall Y.E'_i$,

$$ I(\text{lift}_{W''.\exists \Xi_1}(G_1[\beta/Y]), \text{lift}_{W''.\exists \Xi_1}(G_1[\beta/Y])) = \langle E''_i[E'_i/Y], E''_i[E'_i/Y] \rangle $$

also we know that:

$$ \langle E''_i[E'_i/Y][\alpha E_i/X], E''_i[E'_i/Y][E'_i/X] \rangle = \langle E''_i[\alpha E_i/X][E'_i/Y], E''_i[E'_i/X][E'_i/Y] \rangle $$

Matías Toro, Elizabeth Labrada, and Éric Tanter
Note that $\rho(G_1[\beta/Y]) = \rho(Y \mapsto \beta)(G_1)$. Then we know that

$$\langle W', (\varepsilon_{t_1} \circ (E_{t_1}[\beta_1/E_1], E_{t_2}[\beta_2/E_2])) \circ (E'_{t_1}[\alpha E_1/X] [E_1'/Y], E'_{t_2}[\alpha E_2/X] [E_2'/Y]) \rangle \mapsto \rho(Y \mapsto \beta)(G_1[G'/X]),$$

and $\langle W', (\varepsilon_{t_2} \circ (E_{t_1}[\beta_1/E_1], E_{t_2}[\beta_2/E_2])) \circ (E'_{t_1}[\alpha E_1/X] [E_1'/Y], E'_{t_2}[\alpha E_2/X] [E_2'/Y]) \rangle \mapsto \rho(Y \mapsto \beta)(G_1[G'/X])$

and by inspection of the reduction rules:

$$\langle W', \Xi \mapsto t' \rangle$$

then we prove as (2):

Case $(G_1 \times G_2)$. Analogous to the function case.

Case $(B)$. Trivial.

Then we prove as (2):

Case $(G = X)$. Let $v_1 = \langle H_{t_1}, E_{t_1} \rangle u_1 :: X[\beta_1/G'/X] = (H_{t_1}, E_{t_1}) u_1 :: G'$. Then we know that

$$(\downarrow W, (H_{t_1}, E_{t_1}) u_1 :: G', (H_{t_2}, E_{t_2}) u_2 :: G') \in \mathcal{V}_\rho[G']$$

and $\epsilon_{t_2}^{-1} = (E'_{t_1}, \alpha E_1)$. Then we have to prove that

$$(W, (H_{t_1}, E_{t_1}) \circ (E'_{t_1}, \alpha E_1)) u_1 :: \alpha, (H_{t_2}, E_{t_2}) \circ (E'_{t_2}, \alpha E_2)) u_2 :: \alpha) \in \mathcal{V}_\rho[\alpha \mapsto \alpha][\alpha]$$

By Lemma E.15, we assume that both combinations of evidence are defined (otherwise the result holds immediately) Then by definition of transitivity $(H_{t_1}, E_{t_1}) \circ (E'_{t_1}, \alpha E_1) = (H_{t_1}, \alpha E_1)$. Then we have to prove that

$$(\downarrow W, (H_{t_1}, \alpha E_{t_2}) u_1 :: \alpha, (H_{t_2}, \alpha E_{t_2}) u_2 :: \alpha) \in \mathcal{V}_\rho[\alpha \mapsto \alpha][\alpha]$$

but as $\alpha$ is sync, then that is equivalent to

$$(\downarrow W, (H_{t_1}, E_{t_2}) u_1 :: G', (H_{t_2}, E_{t_2}) u_2 :: G') \in \mathcal{V}_\rho[G']$$

which is part of the premise by Lemma E.20, and the result holds.

Case $(G = Y)$. Let $v_1 = \langle H_{t_1}, \beta E_{t_2} \rangle u_1 :: \rho(Y[\beta_1/G'/X]) = (H_{t_1}, \beta E_{t_2}) u_1 :: \beta$ (where $\rho(Y) = \beta$). Then we know that

$$(W, (H_{t_1}, \beta E_{t_2}) u_1 :: \beta, (H_{t_2}, \beta E_{t_2}) u_2 :: \beta) \in \mathcal{V}_\rho[\beta]$$

Note that $\epsilon_{t_2}^{-1} \vdash W, \Xi \mapsto \beta \sim \beta$, and $\epsilon_1$ has to have the form $\epsilon_1 = \langle \beta E_1, \beta E'_1 \rangle = (\text{lift}_{W, \Xi}(\beta), \text{lift}_{W, \Xi}(\beta))$. As $\epsilon_1$ is the initial evidence for $\beta$, then $E_{t_2} \subseteq E'_{t_1}$, and therefore by definition of the transtivity:

$$(H_{t_1}, \beta E_{t_2}) \circ (\beta E'_1, \beta E'_1) = (H_{t_1}, \beta E_{t_2})$$

Then we have to prove that:

$$(\downarrow W, (H_{t_1}, \beta E_{t_2}) \circ (\beta E'_1, \beta E'_1)) u_1 :: \beta, ((H_{t_2}, \beta E_{t_2}) \circ (E'_{t_2}, \beta E'_{t_2})) u_2 :: \beta) \in \mathcal{V}_\rho[\beta]$$

or what is the same

$$(\downarrow W, (H_{t_1}, \beta E_{t_2}) u_1 :: \beta, (H_{t_2}, \beta E_{t_2}) u_2 :: \beta) \in \mathcal{V}_\rho[\beta]$$

which is part of the premise and the result holds.
Case ($G = ?$). Let $\nu_i = \langle H_{i1}, E_{i2} \rangle u_i :: ?$. Then by definition of $V'_\rho[?]$, let $G'' = \text{const}(E_{i2})$ (where $G'' \neq ?$). Then we know

\[(W, \langle H_{i1}, E_{i2} \rangle u_1 :: G'', \langle H_{i2}, E_{i2} \rangle u_2 :: G''') \in V'_\rho[\Gamma']\]

If $\epsilon^{-1} = (\gamma, ?)$. Then by Lemma E.18, the result holds immediately. The other cases are analogous to other cases.

Case ($G = G_1 \rightarrow G_2$). Let $\nu_i = \epsilon'_i(\lambda x G^i.t_i) :: \rho(G'[X])$ We know that

\[(W, \nu_1, \nu_2) \in V'_\rho[G'[G'/X] \rightarrow G_2[G'/X]]\]

Then we have to prove that

\[\downarrow W, (\epsilon'_i \circ \epsilon^{-1})(\lambda x G^i.t_i) :: \rho'(G_1) \rightarrow \rho'(G_2),\]

\[(\epsilon'_i \circ \epsilon^{-1})(\lambda x G^i.t_i) :: \rho'(G_1) \rightarrow \rho'(G_2)) \in V'_\rho[G_1 \rightarrow G_2]\]

Let us call $\nu''_i = (\epsilon'_i \circ \epsilon^{-1})(\lambda x G^i.t_i) :: \rho'(G_1) \rightarrow \rho'(G_2)$. By unfolding, we have to prove that

\[\forall W'' \downarrow W. \forall \nu'_i, \nu'_j (W', \nu'_i, \nu'_j, \nu''_i, \nu''_j \in V'_\rho[G_1] \Rightarrow (W', \nu'_i, \nu'_j) \in T_{\rho'}[G_2]\]

Suppose that $\nu'_i = \epsilon''u'_i :: \rho'(G_1)$, by inspection of the reduction rules, we know that

\[W'. \Xi \nu'_i \nu'_j \rightarrow W'. \Xi \nu'_i (\text{cod}(\epsilon'_i) \circ \text{cod}(\epsilon^{-1})) t_i ([\epsilon''u'_i :: G'_i]/X) :: \rho'(G_2)\]

This is equivalent by Lemma E.19,

\[W'. \Xi \nu'_i \nu'_j \rightarrow W'. \Xi \nu'_i (\text{cod}(\epsilon'_i) \circ \text{cod}(\epsilon^{-1})) t_i ([\epsilon''u'_i :: G'_i]/X) :: \rho'(G_2)\]

Notice that $\text{dom}(\epsilon^{-1}) \vdash W. \Xi \rho(G_1[\alpha'/X]) \sim \rho(G_1[G'/X])$, and as $\text{dom}(\epsilon^{-1})$ is constructed using the interior (and thus $\pi_2(\epsilon''u'_i) \subseteq \pi_1(\text{dom}(\epsilon^{-1}))$), then by definition of evidence $(\epsilon''u'_i :: \text{dom}(\epsilon^{-1}))$ is always defined. We can use induction hypothesis on $\nu'_i$, with evidences $\text{dom}(\epsilon^{-1})$. Then we know that

\[(W', (\epsilon''u'_i :: \text{dom}(\epsilon^{-1}))u'_i :: \rho(G_1[G'/X])/\epsilon'_i :: \rho(G_1[G'/X])) \in V'_\rho[G_1[G'/X]]\]

Let us call $\nu'''_i = (\epsilon''u'_i :: \text{dom}(\epsilon^{-1}))u'_i :: \rho(G_1[G'/X])$.

Now we instantiate

\[(W, \nu_1, \nu_2) \in V'_\rho[G_1[G'/X] \rightarrow G_2[G'/X]]\]

with $W$ and $\nu'''_i$, to obtain that either both executions reduce to an error (then the result holds immediately), or $\exists W'' \downarrow W''$ such that $W''', \nu_{f1}, \nu_{f2} \in V'_\rho[G_2[G'/X]]$

\[W'. \Xi \nu _i \nu''_i \rightarrow W'. \Xi \nu''_i \rightarrow W'. \Xi \nu''_i \text{cod}([\epsilon''u'_i :: \text{dom}(\epsilon^{-1})]) \text{dom}(\epsilon''u'_i :: G'_i/X) :: \rho(G_2[G'/X])\]

\[\rightarrow W'. \Xi \nu''_i :: \rho(G_2[G'/X])\]

Suppose that $\nu_{f1} = \epsilon_{f1}u_{f1} :: \rho(G_2[G'/X])$. Then we use induction hypothesis once again using evidences $\text{cod}(\epsilon^{-1})$ over $\nu_{f1}$ (noticing that the combination of evidence does not fail as the evidence is obtained via the interior function i.e. the less precise evidence possible), to obtain that,

\[\downarrow W'', (\epsilon_{f1} :: \text{cod}(\epsilon^{-1}))u_{f1} :: \rho'(G_2), (\epsilon_{f2} :: \text{cod}(\epsilon^{-1}))u_{f2} :: \rho'(G_2)) \in V'_\rho[G_2]\]

and the result holds.

The remaining cases are similar.

\[\square\]

**Definition E.17.** $\rho \vdash \epsilon_1 \equiv \epsilon_2$ if $\text{unlift}((\pi_2(\epsilon_1))) = \text{unlift}((\pi_2(\epsilon_2)))$

**Proposition E.18.** If
then:

\[(W, \rho_1(v_1) :: \rho(G'), \rho_2(v_2) :: \rho(G')) \in \mathcal{T}_\rho[G']\]

where \(\text{sync}(\alpha, \rho) \iff W, \Xi_1(\alpha) = W, \Xi_2(\alpha) \wedge W, \kappa(\alpha) = [\mathcal{V}_\theta W, \Xi_1(\alpha)]\).)

**Proof.** We proceed by induction on \(G\). We know that \(u_i \in G_i\) for some \(G_i\), notice that \(G_i \in \text{HEADTYPE} \cup \text{TYPEVAR}\). In every case we apply Lemma E.27 to show that \((\varepsilon_1 \circ \epsilon_1^p) \iff (\varepsilon_2 \circ \epsilon_2^p)\), so in all cases we assume that the transitivity does not fail (otherwise the proof holds immediately).

Let us call \(\epsilon_1^p = \rho_1(\epsilon)\) and \(\epsilon_2^p = \rho_2(\epsilon)\).

**Case** \((G = B\) and \(G' = B\)). We know that \(v_1\) has the form \(\langle B, B \rangle u :: B\), and we know that \((W, \langle B, B \rangle u :: B) \in \mathcal{V}_\rho[B]\). Also as \(\varepsilon + \Xi; \Delta + B \sim B\), then \(\varepsilon = \langle B, B \rangle\), then as \(\rho_1(B) = B, \varepsilon_1 \circ \rho_1(\epsilon) = \varepsilon_1\), and we have to prove that \((W, \langle B, B \rangle u :: B, \langle B, B \rangle u :: B) \in \mathcal{V}_\rho[B]\), which is part of the premise and the result holds.

**Case** \((G = G_i' \rightarrow G_i''\) and \(G' = G_i' \rightarrow G_i''\)). We know that:

\[(W, v_1, v_2) \in \mathcal{V}_\rho[G_i' \rightarrow G_i'']\]

Where \(v_1 = \epsilon_i(\lambda x^{G_i'.t_i}.t_i) :: G_i' \rightarrow G_i''\) and \(\epsilon_i + W, \Xi_i + G_i \sim G_i' \rightarrow G_i''\).

We have to prove that:

\[(\downarrow W, \epsilon_1^p v_1 :: G_i' \rightarrow G_i'', \epsilon_2^p v_2 :: G_i' \rightarrow G_i'') \in \mathcal{T}_\rho[G_i' \rightarrow G_i'']\]

First we suppose that \((\varepsilon_1 \circ \epsilon_1^p)\) does not fail, then we have to prove that:

\[\forall W' \exists W.\forall u_1', u_2'.(W', v_1', v_2') \in \mathcal{V}_\rho[G_i'] \Rightarrow (W', ((\varepsilon_1 \circ \epsilon_1^p)(\lambda x^{G_i'.t_i}.t_i) :: G_i' \rightarrow G_i'')) \wedge (\varepsilon_2 \circ \epsilon_2^p)(\lambda x^{G_i'.t_i}.t_i) :: G_i' \rightarrow G_i'') \exists W' \Xi_i \Rightarrow cod(\varepsilon_1 \circ \epsilon_1^p)(\exists G_i' \rightarrow G_i'')\]

We know by the Proposition E.21 that \(\text{dom}(\varepsilon_1 \circ \epsilon_1^p) = \text{dom}(\epsilon_1^p) \circ \text{dom}(\varepsilon_1)\). Then by the Proposition E.19 we know that:

\[\varepsilon_1^p \circ (\text{dom}(\varepsilon_1 \circ \epsilon_1^p)) = \varepsilon_1^p \circ (\text{dom}(\epsilon_1^p) \circ \text{dom}(\varepsilon_1)) = (\varepsilon_1^p \circ \text{dom}(\epsilon_1^p)) \circ \text{dom}(\varepsilon_1)\]

Also, by the Proposition E.22 it is follows that: \(\text{cod}(\varepsilon_1 \circ \epsilon_1^p) = cod(\varepsilon_1) \circ cod(\epsilon_1^p)\).

Then the following result is true:

\[W' \Xi_i \Rightarrow cod(\varepsilon_1 \circ \epsilon_1^p)(\exists G_i' \rightarrow G_i'') \wedge W' \Xi_i \Rightarrow cod(\varepsilon_1 \circ \epsilon_1^p)(\exists G_i' \rightarrow G_i'')\]

We instantiate the induction hypothesis in \((W', v_1', v_2') \in \mathcal{V}_\rho[G_i']\) with the type \(G_i''\) and the evidences \(\text{dom}(\varepsilon) \vdash \Xi; \bar{\Delta} \vdash G_i' \sim G_i''\). We obtain that:

\[(W', \text{dom}(\epsilon_1^p) v_1', \text{dom}(\epsilon_2^p) v_2') \in \mathcal{T}_\rho[G_i'']\]

\[\text{dom}(\varepsilon) \vdash \Xi; \bar{\Delta} \vdash G_i' \sim G_i''\]
In particular we focus on a pair of values such that \( (\epsilon'_1 \circ \text{dom}(\rho'_0)) \) does not fail (otherwise the result follows immediately). Then it is true that:

\[
(W', (\epsilon'_1 \circ \text{dom}(\rho'_0))u'_1 :: G'_1, (\epsilon'_2 \circ \text{dom}(\rho'_0))u'_2 :: \nabla_u G'_1) \in \mathcal{V}_p[G'_1']
\]

By the definition of \( \mathcal{V}_p[G'_1' \rightarrow G'_2'] \) we know that:

\[
\forall W'' \geq W. \forall v'_1, v'_2. \forall \lambda' v''_1, v''_2. (W'', v'_1, v''_1) \in \mathcal{V}_p[G'_2'] \Rightarrow (W'', v'_1, v''_1, v'_2, v''_2) \in \mathcal{T}_p[G'_2']
\]

We instantiate \( v''_1 = (\epsilon'_1 \circ \text{dom}(\rho'_0))u'_1 :: G'_1' \) and \( W'' = W' \), then we obtain that:

\[
(W', ((\lambda x. G'_1).t_1) :: G'_1' \rightarrow G'_2') ((\epsilon'_1 \circ \text{dom}(\rho'_0))u'_1 :: G'_1'),
\]

\[
(\epsilon'_2(\lambda x. G'_2).t_2) :: G'_2' ((\epsilon'_1 \circ \text{dom}(\rho'_0))u'_1 :: G'_1') \in \mathcal{T}_p[G'_2']
\]

Then by Lemma E.19, as \( (\epsilon'_1 \circ \text{dom}(\rho'_0)) \circ \text{dom}(\epsilon_i) = \epsilon'_1 \circ (\text{dom}(\rho'_0) \circ \text{dom}(\epsilon_i)) \) if \( \text{dom}(\epsilon_i) \circ \text{dom}(\epsilon_i) \) is not defined and \( (\text{dom}(\epsilon_i)) \circ \text{dom}(\epsilon_i)) \) is defined, we get a contradiction as both must behave uniformly as the terms belong to \( \mathcal{T}_p[G'_2'] \). Then if both combination of evidence fail, then the result follows immediately. Let us suppose that the combination does not fail, then

\[
W'. \Xi_1 \triangleright (\lambda x. G'_1).t_1 :: G'_1' \rightarrow G'_2' ((\epsilon'_1 \circ \text{dom}(\rho'_0))u'_1 :: G'_1') \rightarrow \nabla u W'' . \Xi_1 \triangleright v''_1
\]

The resulting terms reduce to values \( (v''_1) \) and \( \exists W''' \geq W'' \) such that \( (W'''', v'''_1, v'''_2) \in \mathcal{V}_p[G'_2'] \).

\[
W'. \Xi_1 \triangleright \text{dom}(\epsilon_i)((((\epsilon'_1 \circ \text{dom}(\rho'_0)) \circ \text{dom}(\epsilon_i))u'_1 :: G'_1)\cdot x. G'_1).t_1 :: G'_2' \rightarrow W'''. \Xi_1 \triangleright v'''_1
\]

We instantiate the induction hypothesis in the previous result with the type \( G'_2 \) and the evidence \( \text{cod}(\epsilon) \vdash \Xi; \Delta \vdash G'' \rightarrow G' \), then we obtain that:

\[
(W''', \text{cod}(\epsilon'_i)v'''_1 :: G'_2, \text{cod}(\epsilon'_2)v'''_2 :: G'_2) \in \mathcal{T}_p[G'_2']
\]

Then \( v'''_1 \) has to have the form: \( v'''_1 = (\epsilon''_1 \circ \text{cod}(\epsilon_i))u''_1 :: G''_1 \) form some \( \epsilon''_1, \epsilon''_2 \). Then as \( \epsilon''_2 = (\epsilon''_1 \circ \text{cod}(\epsilon_i)) = (\epsilon''_2 \circ \text{cod}(\epsilon_i)) \) then \( \text{cod}(\epsilon_i) \circ \text{cod}(\rho'_0) \) must behave uniformly (either the two of them fail, or the two of them does not fail), and the result immediately.

Case \( G = \forall X. G'_1' \) and \( G' = \forall X. G'_1' \). We know that:

\[
(W, v_1, v_2) \in \mathcal{V}_p[\forall X. G'_1']
\]

Where \( v_1 = \epsilon'_1(\lambda x. t_1) :: \forall X. \rho(G'_1') \) and \( \epsilon_1 + W. \Xi_1 \vdash G_i \sim \forall X. \rho(G'_1') \).

We have to prove that:

\[
(\downarrow W, \epsilon'_1 v_1 :: \forall X. \rho(G'_1'), \epsilon'_2 v_2 :: \forall X. \rho(G'_1')) \in \mathcal{T}_p[\forall X. G'_1']
\]

As \( (\epsilon_1 \circ \epsilon'_i) \) does not fail, then by the definition of \( \mathcal{T}_p[\forall X. G'_1'] \) we have to prove that:

\[
(W, (\epsilon_1 \circ \epsilon'_i)(\lambda x. t_1) :: \forall X. \rho(G'_1'), (\epsilon_2 \circ \epsilon'_i)(\lambda x. t_2) :: \forall X. \rho(G'_1')) \in \mathcal{V}_p[\forall X. G'_1']
\]
or what is the same:
\[
\forall W'': \forall \forall \forall t', t'_1, G^*_1, G^*_2, \alpha, \epsilon_{11}, \epsilon_{21}, \forall R \in \text{REL}_{W''}[G^*_1, G^*_2].
\]
\[
(W'').\Xi_1 \triangleright G^*_1 \land W'''.\Xi_2 \triangleright G^*_2 \land
\]
\[
W''.\Xi_1 \triangleright ((\epsilon_{11} \circ \epsilon_{1}^P) u_1 : \forall X. G^*_1)[G^*_1] \rightarrow \forall W''.\Xi_1, \alpha := G^*_1 \triangleright \epsilon_{11} t'_1 : G^*_1[X/X] \land
\]
\[
W'''.\Xi_2 \triangleright ((\epsilon_{21} \circ \epsilon_{2}^P) u_2 : \forall X. G^*_2)[G^*_2] \rightarrow \forall W'''.\Xi_2, \alpha := G^*_2 \triangleright \epsilon_{21} t'_2 : G^*_2[X/X] \Rightarrow
\]
\[
(\downarrow (W'''.\Xi_1 \triangleright (\alpha, G^*_1, G^*_2, R), t'_1, t'_2) \in T_{\rho[X \rightarrow a]}[G^*_1])
\]
For simplicity, let us call \( W''' := \downarrow (W'''.\Xi_1 \triangleright (\alpha, G^*_1, G^*_2, R)) \). Note that by the reduction rule of type application, we obtain that:
\[
(W'').\Xi_1 \triangleright ((\epsilon_{11} \circ \epsilon_{1}^P) \lambda X. t_1 : \forall X. \rho(G^*_1)) [G^*_1] \rightarrow
\]
\[
(W'''.\Xi_1, \alpha := G^*_1 \triangleright \epsilon_{11}^P\rho(G^*_1) ((\epsilon_{11} \circ \epsilon_{1}^P)[\alpha^{E_i}] t_1[\alpha^{E_i}/X] \vdash \rho(G^*_1)[\alpha/X] : \rho(G^*_1)[G^*_1/X])
\]
where \( E_i = \text{lift}(W'''.\Xi_1)(G^*_1) \). The resulting evidences \( \epsilon_{11} \circ \epsilon_{1}^P \) have the form:
\[
(\epsilon_{11}[\alpha^{E_i}/X], \epsilon_{12}[\alpha^{E_i}/X])t_1[\alpha^{E_i}/X] \vdash \rho(G^*_1)[\alpha/X] : \rho(G^*_1)[G^*_1/X]
\]
Then we have to prove that:
\[
(W''', (\epsilon_{11}[\alpha^{E_i}/X], \epsilon_{12}[\alpha^{E_i}/X])t_1[\alpha^{E_i}/X] \vdash \rho(G^*_1)[\alpha/X], (\epsilon_{21}[\alpha^{E_i}/X], \epsilon_{22}[\alpha^{E_i}/X])t_2[\alpha^{E_i}/X] \vdash \rho(G^*_1)[\alpha/X])
\]
\[
\in T_{\rho[X \rightarrow a]}[G^*_1]
\]
Also by the Proposition E.23 we know that:
\[
(\epsilon_{11}[\alpha^{E_i}]) \circ (\epsilon_{1}^P[\alpha^{E_i}]) = (\epsilon_{11}[\alpha^{E_i}]) \circ (\epsilon_{1}^P[\alpha^{E_i}])
\]
Note that:
\[
(\epsilon_{11}[\alpha^{E_i}]) \circ (\epsilon_{1}^P[\alpha^{E_i}]) = (\epsilon_{11}[\alpha^{E_i}]) \circ (\epsilon_{1}^P[\alpha^{E_i}])
\]
Then we have to prove that:
\[
(W'''', (\epsilon_{11}[\alpha^{E_i}]) \circ (\epsilon_{1}^P[\alpha^{E_i}])t_1[\alpha^{E_i}/X] \vdash \rho(G^*_1)[\alpha/X], (\epsilon_{21}[\alpha^{E_i}]) \circ (\epsilon_{2}^P[\alpha^{E_i}])t_2[\alpha^{E_i}/X] \vdash \rho(G^*_1)[\alpha/X])
\]
\[
\in T_{\rho[X \rightarrow a]}[G^*_1]
\]
Note by the reduction rule of type application, we obtain that:
\[
(W'').\Xi_1 \triangleright (\epsilon_{1} \lambda X. t_1 : \forall X. \rho(G''')) [G^*_1] \rightarrow
\]
\[
(W'''.\Xi_1, \alpha := G^*_1 \triangleright \epsilon_{1}^{P\rho(G''')} ((\epsilon_{1} \circ \epsilon_{1}^P)[\alpha^{E_i}] t_1[\alpha^{E_i}/X] \vdash \rho(G''')[\alpha/X] : \rho(G''')[G^*_1/X])
\]
Note that the evidence \( \epsilon_{1} \) has the form:
\[
(\forall X. E''', \forall X. E'''),
\]
then:
\[
(\epsilon_{1}^{P\rho(G''')} ((\epsilon_{1} \circ \epsilon_{1}^P)[\alpha^{E_i}] t_1[\alpha^{E_i}/X] \vdash \rho(G''')[\alpha/X] : \rho(G''')[G^*_1/X])
\]
As we know that \( (W, v_1, v_2) \in \forall \rho_{\forall X. G^*_1} \), then we can instantiate with \( \forall W'' \geq W, G^*_1, G^*_2, R, \alpha, \epsilon_{11}[\alpha^{E_i}][\alpha^{E_i}/X] : \rho(G''')[\alpha/X], \epsilon_{21}[\alpha^{E_i}][\alpha^{E_i}/X] : \rho(G''')[\alpha/X], \epsilon_{E_i}^{P\rho(G''')} \) and \( \epsilon_{E_i}^{P\rho(G''')} \).
Then we know that:
\[(W'', e_1[α^E]t_1[α^E_1/X] ∶ ρ(G''_1)[α/X]), e_2[α^E]t_2[α^E_2/X] ∶ ρ(G''_2)[α/X]) ∈ T_ρ[X ↦ α][G'']\]

If the following term reduces to error, then the result follows immediately.
\[W'''' ∪_i e_1[α^E]t_1[α^E_1/X] ∶ ρ(G''_1)[α/X])\]

If the above is not true, then the following terms reduce to values (ν_1f) and ∃W'''' ≥ W'''' such that (W'''', ν_1f, ν_2f) ∈ V_ρ[X ↦ α][G'']
\[W'''' ∪_i e_1[α^E]t_1[α^E_1/X] ∶ ρ(G''_1)[α/X]) → W'''''' ∪_i ν_1f\]

By definition of consistency and the evidence we know that ε[X] ⊢ W'''''' ∪_i ν_1f, G'''' ∼ G'. Then we instantiate the induction hypothesis in the previous result with G = G' and ε = ε[X]. Calling ρ' = ρ[X ↦ α], then we obtain that:
\[(W''''', ρ'_1(ε[X])ν_1f ∶ ρ'(G'_1), ρ'_2(ε[X])ν_2f ∶ ρ'(G'_2)) ∈ T_ρ'[G'_1]\]

but as ρ'_1(ε[X]) = ε'_1[α^E] which is equivalent to
\[(W''''', ε'_1[α^E])ν_1f ∶ ρ'(G'_1)[α/X], (ε'_2[α^E])ν_2f ∶ ρ'(G'_2)[α/X]) ∈ T_ρ'[G'_1]\]

and the result follows immediately.

Case (G = G_1 × G_2). Similar to function case.

Case (A)(G = α). This means that α ∈ dom(Ξ). We know that (W, ε_1u_1 ∶ α, ε_2u_2 ∶ α) ∈ V_ρ[α] and ε_1 ⊢ W, ε_1 ⊢ G_1 ∼ α, then ε_1 = ⟨E_1, α^E_1⟩.

We proceed by doing case analyze on ε.

(A.i) (ε = ⟨α^3, α^4⟩) Then by definition of the transitivity operator, ε_1 ∘ ε = ⟨E''''', α^E'''⟩ (where ⟨E_1, E'_1⟩ ∘ ⟨?, ?⟩ = ⟨E_1, E'_1⟩). Then we have to prove that
\[(↓ W, ⟨E_1, α^E_1⟩u_1 ∶ G', ⟨E_2, α^E_2⟩u_2 ∶ G') ∈ V_ρ[G']\]

where G' is either ? or α. In any case this is equivalent to prove that
\[(↓ W, ⟨E_1, α^E_1⟩u_1 ∶ α, ⟨E_2, α^E_2⟩u_2 ∶ α) ∈ V_ρ[α]\]

which is part of the premise and the result holds.

(A.ii) (ε = ⟨α^3, ?⟩) then G' = ?, and W_ε, ε_1(α) = ?.

Then by definition of the transitivity operator, ε_1 ∘ ε = ⟨E_1, ε_1⟩ (where ⟨E_1, ε_1⟩ ∘ ⟨?, ?⟩ = ⟨E_1, ε_1⟩). Then we have to prove that
\[(↓ W, ⟨E_1, ε_1⟩u_1 ∶ ?, ⟨E_2, ε_1⟩u_2 ∶ ?) ∈ V_ρ[?]\]

But by definition of V_ρ[α], the result holds immediately.

(A.iii) (ε = ⟨α^βE_3, E_4⟩). Then β ∈ dom(Ξ), and for transitivity to be defined, ε_1 = ⟨E_1, α^βE_4⟩. Then suppose that ε'_1 = ⟨α^βE_3, E_4⟩, then by definition of transitivity
\[⟨E_1, α^βE_4⟩ ∘ ⟨α^βE_3, E_4⟩ = ⟨E_1, β^E_4⟩ ∘ ⟨βE_3, E_4⟩\]

Also notice that by Lemmas E.29 and E.28, ⟨βE_3, E_4⟩ ⊢ Ξ; Δ ⊢ β ∼ G' where β is sync, and by definition of the logical relation
\[(W, ⟨E_1, β^E_4⟩u_1 ∶ β, ⟨E_2, β^E_4⟩u_2 ∶ β) ∈ V_ρ[β]\]
so we proceed just like case \((G = \alpha)\) one more time but with \(G = \beta\) and \(\varepsilon = \langle \beta \xi, E_4 \rangle\).

(A.iv) \((\varepsilon = \langle \alpha \xi, E_4 \rangle)\). So for transitivity to be defined, \(\varepsilon_1 = \langle H_i, \alpha \xi \rangle\). Then suppose that 
\[
\varepsilon_1^\rho = \langle \alpha \xi, E_4 \rangle,
\]
then by definition of transitivity
\[
\langle H_i, \alpha \xi \rangle \circ \langle \alpha \xi, E_4 \rangle = \langle H_i, H_i' \rangle \circ \langle H_3, E_4 \rangle.
\]

Also, as \(\alpha\) is sync then \(W.\Xi_1(\alpha) = W.\Xi_2(\alpha)\). Let us call \(G_\alpha = W.\Xi_1(\alpha)\). Then by definition of the interpretation for type names
\[
(W, \langle H_1, H_1' \rangle) u_1 :: G_\alpha, \langle H_2, H_2' \rangle u_2 :: G_\alpha) \in \mathcal{V}_\rho[G_\alpha]
\]
where \(G_\alpha \notin \text{Type}\).

Also notice that as \(\langle \alpha \xi, E_4 \rangle \vdash \Xi; \Delta \vdash G \sim G'\), where \(\alpha \subseteq G\), then by Lemma 2.28, \(\langle \alpha \xi, E_4 \rangle \vdash \Xi; \Delta \vdash \alpha \sim G'\). Also by Lemma 2.29 \(\langle H_3, E_4 \rangle \vdash \Xi; \Delta \vdash G_\alpha \sim G'\). Then we proceed just like case \((G \neq \alpha)\) where \(G = G_\alpha\) and \(\varepsilon = \langle H_3, E_4 \rangle\).

Case (B)\((G = X)\). Suppose that \(\rho(X) = \alpha\). We know that \(\alpha \notin \Xi\), i.e. \(\alpha\) may not be in sync, that \((W, \varepsilon_1 u_1 :: \alpha, \varepsilon_2 u_2 :: \alpha) \in \mathcal{V}_\rho[X]\) and that \(\varepsilon_1 + W.\Xi_1 + G_1 \sim \alpha\), then \(\varepsilon_1 = \langle E_1, \alpha \xi \rangle\).

Then by construction of evidences, \(\varepsilon\) must be either \(\langle X, X \rangle\) or \(\langle ?, ? \rangle\) (any other case will fail when the met is computed).

- \((\varepsilon = \langle X, X \rangle)\). Then \(\varepsilon_1^\rho = \langle \rho_i(X), \rho_i(X) \rangle\). But \(\rho_i(X)\) is the type that contains the initial precision for \(\alpha\). Therefore \(\alpha \xi \subseteq \rho_i(X)\), and by Lemma 3.31, \(\varepsilon_1 \circ [i] = \varepsilon_1\) and the result holds immediately (notice that if \(G' = ?\) then we have to show that they are related to \(\alpha\) which is part of the premise).

Case (C)\((G = ?)\). We know that \((W, \varepsilon_1 u_1 :: ?, \varepsilon_2 u_2 :: ?) \in \mathcal{V}_\rho[?]\) and \(\varepsilon_1 + W.\Xi_1 + G_1 \sim \beta\). We are going to proceed by case analysis on \(\pi_2(\varepsilon_1)\) and \(\rho + \varepsilon_1 \equiv \varepsilon_2\):

(C.i) \((\varepsilon_i = \langle E_i, \alpha \xi \rangle)\). Then this means we know that 
\[
(W, \varepsilon_1 u_1 :: \alpha, \varepsilon_2 u_2 :: \alpha) \in \mathcal{V}_\rho[\alpha]
\]
and \(\varepsilon_1 + W.\Xi_1 + G_1 \sim \alpha\), then \(\varepsilon_1 = \langle E_1, \alpha \xi \rangle\).

(a) \((\varepsilon = \langle \alpha \xi, E_4 \rangle)\). Then as \(\langle E_i, \alpha \xi \rangle \vdash \Xi; \Delta \vdash G_i \sim ?\), then by Lemma 2.28 \(\langle E_i, \alpha \xi \rangle \vdash \Xi; \Delta \vdash \alpha \sim G_i\). Also we know that \(? \subseteq G\), then \(G = ?\), and \(\alpha \subseteq G\). Finally, we reduce this case to the Case A if \(\alpha \in \Xi\) or Case B if \(\alpha \notin \Xi\).

(b) \((\varepsilon = \langle ?, ?, \rangle)\). Then \(G' = ?\), and does \(\varepsilon_1 \circ \varepsilon = \varepsilon_1\). Then we have to prove that \((\downarrow W, \varepsilon_1 u_1 :: ?, \varepsilon_2 u_2 :: ?) \in \mathcal{V}_\rho[?]\), and as \(\text{const}(\alpha \xi) = \alpha\) that is equivalent to prove that \((\downarrow W, \varepsilon_1 u_1 :: \alpha, \varepsilon_2 u_2 :: \alpha) \in \mathcal{V}_\rho[\alpha]\) which is part of the premise by Lemma 2.20 and the result holds immediately.

(c) \((\varepsilon = \langle ?, \beta \xi \dot{\cdots} \rangle)\). Where \(\beta\) cannot transitively point to some unsync variable. Then by definition of the transitivity operator, \(\varepsilon_1 \circ \varepsilon = \langle E_i', \beta \xi \dot{\cdots} \rangle\) (where \(\langle E_i, \alpha \xi \rangle \circ \langle ?, \beta \xi \dot{\cdots} \rangle = \langle E_i', \alpha \xi \rangle\)). Then we have to prove that 
\[
(\downarrow W, \langle E_i', \beta \xi \dot{\cdots} \rangle u_1 :: G', \langle E_2', \beta \xi \dot{\cdots} \rangle u_2 :: G') \in \mathcal{V}_\rho[G']
\]
where \(G'\) is either ? or \(\beta\). In any case this is equivalent to prove that 
\[
(\downarrow W, \langle E_1', \beta \xi \dot{\cdots} \rangle u_1 :: \beta, \langle E_2', \beta \xi \dot{\cdots} \rangle u_2 :: \beta) \in \mathcal{V}_\rho[\beta] \iff (\downarrow W, \langle E_1', \beta \xi \dot{\cdots} \rangle u_1 :: G'', \langle E_2', \beta \xi \dot{\cdots} \rangle u_2 :: G'') \in \mathcal{V}_\rho[G'']
\]
where \(G'' = W.\Xi_1(\beta) = W.\Xi_2(\beta)\) (note that \(\beta\) is sync). As \(\langle E_i, \alpha \xi \rangle \circ \langle ?, \beta \xi \dot{\cdots} \rangle = \langle E_i', \alpha \xi \rangle\), then we can reduce the demonstration to proof that:
\[
(\downarrow W, \langle E_1, \alpha \xi \rangle \circ \langle ?, \beta \xi \dot{\cdots} \rangle u_1 :: G'', \langle E_2, \alpha \xi \rangle \circ \langle ?, \beta \xi \dot{\cdots} \rangle u_2 :: G'') \in \mathcal{V}_\rho[G'']
\]
Finally, we reduce this case to this same case (note that we have base case because the sequence ends in $\epsilon$).

(d) $\epsilon = (\epsilon, \beta')$. Then by definition of the transitivity operator, $\epsilon_1 \circ \epsilon = (E''', \beta''')$ (where $\langle E_1, \alpha^{E_1} \rangle \circ (?, ?) = (E''', \epsilon''')$). Then we have to prove that

$$\langle \downarrow W, (E''', \beta''') \rangle u_1 :: G', (E'_2, \beta'''') u_2 :: G' \rangle \in V_\rho[G']$$

where $G'$ is either $\alpha$ or $\beta$. In any case this is equivalent to prove that

$$\langle \downarrow W, (E''', \beta''') \rangle u_1 :: \beta, (E'_2, \beta'''') u_2 :: \beta \rangle \in V_\rho[\beta]$$

$\iff$

$$\langle \downarrow W, (E''', E''''') u_1 :: G'', (E'_2, E'''') u_2 :: \beta \rangle \in V_\rho[G'']$$

where $G'' = W.\Xi_i(\beta) = W.\Xi_2(\beta) = \epsilon$ (note that $\beta$ is sync). As $\langle E_i, \alpha^{E_i} \rangle \circ (?, ?) = (E'_i, \epsilon_i)$, then we can reduce the demonstration to prove that:

$$\langle \downarrow W, (E_i, \alpha^{E_i}) u_1 :: \alpha, (E_2, \alpha^{E_i}) u_2 :: \alpha \rangle \in V_\rho[\alpha]$$

which is part of the premise and the result holds.

(C.i) $(\epsilon_1 = (H_{i1}, H_{i2}))$. Then as $G = ?$ and $G \sqsubseteq G$, then $G = \epsilon$. Let $G'' = \text{const}(H_{i2})$, and we know that $G'' \in \text{HEADType}$. By unfolding of the logical relation for $\epsilon$, we also know that

$$\langle W, (H_{i1}, H_{i2}) u_1 :: \epsilon, (H_{i2}, H_{i2}) u_2 :: \epsilon \rangle \in V_\rho[G'']$$

and we have to prove that

$$\langle \downarrow W, (H_{i1}, H_{i2}) \circ \epsilon_i^0 u_1 :: ? \rangle, \langle H_{i2}, H_{i2} \rangle \circ \epsilon_i^0 u_2 :: \epsilon \rangle \in V_\rho[G'']$$

Note that for consistent transitivity to hold, then $\epsilon$ has to take the following forms:

(a) $\epsilon = (H_3, E_i)$. Then as $\epsilon \models \Xi; \Delta \Rightarrow \epsilon' \Rightarrow G'$, by Lemma E.28, we proceed just like Case D, where $G \in \text{HEADType} (G = G''$).

(b) $\epsilon = (?, ?)$. Then $G' = ?$ (let us assume without losing generality that $H_{ij} = E_i \rightarrow E_i$, and thus $G'' = ? \Rightarrow ?$) $(H_{i1}, H_{i2}) \circ (?, ?) = (H_{i1}, H_{i2})$. Then we have to prove that the resulting values are in the interpretation of $G'' = ? \Rightarrow ?$, which we already know as premise and the result holds immediately.

(c) $\epsilon = (?, \alpha')$. Then (let us assume without losing generality that $H_{ij} = E_i \rightarrow E_i$, and thus $G'' = ? \Rightarrow ?$) $W_i.\Xi(\alpha) = ?$, and by inspection of the consistent transitivity rules, $(H_{i1}, H_{i2}) \circ (?, \alpha') = (H_{i1}, \alpha^{H_{i2}})$. Then by definition of the interpretation of $G'$ (which may be $\epsilon$ or $\alpha$), we have to prove that

$$\langle \downarrow W, (H_{i1}, H_{i2}) u_1 :: ?, (H_{i2}, H_{i2}) u_2 :: ? \rangle \in V_\rho[\epsilon]$$

which is part of the premise, and the result holds.

(d) $\epsilon = (?, \alpha^{E_i})$. Then (let us assume without losing generality that $H_{ij} = E_i \rightarrow E_i$, and thus $G'' = ? \Rightarrow ?$) $W_i.\Xi(\alpha) \in \{\beta, ?\}$, and by inspection of the consistent transitivity rules, $(H_{i1}, H_{i2}) \circ (?, \alpha^{E_i}) = (H_{i1}, \alpha^{E_{i2}})$, where $(H_{i1}, H_{i2}) \circ (?, E_{i4}) = (H_{i1}, E_{i4})$. Then by definition of the interpretation of $\alpha$ (after one or two unfolding of $G' = ?$), we have to prove that

$$\langle \downarrow W, (H_{i1}, H_{i2}) \circ (?, \beta^{E_{i4}}) u_1 :: \beta, (H_{i2}, H_{i2}) \circ (?, \beta^{E_{i4}}) u_2 :: \beta \rangle \in V_\rho[\beta]$$

and then we proceed to the same case one more time (notice that the recursion is finite, until we get to the previous sub case).

Case (D) $(G \in \text{HEADType})$. We know that $(W, \epsilon_1 u_1 :: \rho(G), \epsilon_2 u_2 :: \rho(G)) \in V_\rho[G]$ and $\epsilon_1 \vdash W.\Xi \vdash G_i \sim G$. Also $\epsilon_1 = (H_{i1}, H_{i2})$, for some $H_{i1}, H_{i2}$. We proceed by case analysis on $G'$ and $\epsilon$. 

Matías Toro, Elizabeth Labrada, and Éric Tanter
(D.i) \( \epsilon = \langle E_3, \alpha^{E_4} \rangle \). Then \( G' = \alpha \), or \( G' = ? \). Notice that as \( \alpha^{E_4} \) cannot have free type variables therefore \( E_3 \) neither. Then \( \epsilon = \rho_1(\epsilon) \). As \( \alpha \) is sync, then let us call \( G'' = W.\Xi(\alpha) \). In either case \( G' = \alpha \), or \( G' = ? \), what we have to prove boils down to

\[
\downarrow W, (\epsilon_1 \circ \langle E_3, \alpha^{E_4} \rangle)u_1 : \alpha, (\epsilon_2 \circ \langle E_3, \alpha^{E_4} \rangle)u_2 : \alpha) \in \mathcal{V}_\rho[\alpha]
\]

which, by definition of consistent transitivity, is equivalent to prove that

\[
\downarrow W, (\epsilon_1 \circ \langle E_3, E_4 \rangle)u_1 : G'', (\epsilon_2 \circ \langle E_3, E_4 \rangle)u_2 : G'') \in \mathcal{V}_\rho[G'']
\]

Then we proceed by case analysis on \( \epsilon \):

- (Case \( \epsilon = \langle E_3, \alpha^{E_4} \rangle \)). We know that \( \alpha \subseteq G' \) and that \( \langle E_3, \alpha^{\beta^{E_4}} \rangle \models \Xi; \Delta \vdash G \sim G' \), then by Lemma E.28, we know that \( \langle E_3, \alpha^{\beta^{E_4}} \rangle \models \Xi; \Delta \vdash G \sim \alpha \). Also by Lemma E.30, \( \langle E_3, \alpha^{\beta^{E_4}} \rangle \vdash \Xi; \Delta \vdash G \sim G'' \). As \( \beta^{E_4} \subseteq G'' \), then \( G'' \) can either be \( ? \) or \( \beta \).

If \( G'' = ? \), then by definition of \( \mathcal{V}_\rho[?] \), we have to prove that the resulting values belong to \( \mathcal{V}_\rho[\beta] \). Also as \( \langle E_3, \beta^{E_4} \rangle \vdash \Xi; \Delta \vdash G \sim \beta \), by Lemma E.28, \( \langle E_3, \beta^{E_4} \rangle \vdash \Xi; \Delta \vdash G \sim ? \), and then we proceed just like this case once again (this is process is finite as there are no circular references by construction and it ends up in something different to a type name). If \( G'' = \beta \) we use an analogous argument as for \( G'' = ? \).

- (Case \( \epsilon = \langle E_3, \alpha^{E_4}, E_4 \notin \text{SITYPENAME} \rangle \)). Then we have to prove that

\[
\downarrow W, (\epsilon_1 \circ \langle E_3, E_4 \rangle)u_1 : G'', (\epsilon_2 \circ \langle E_3, E_4 \rangle)u_2 : G'') \in \mathcal{V}_\rho[G'']
\]

By Lemma E.30, \( \langle E_3, E_4 \rangle \vdash \Xi; \Delta \vdash G \sim G'' \). Then if \( G'' = ? \), we proceed as the case \( G \in \text{HEADTYPE}, G' = ? \) with \( \epsilon = \langle E_3, E_4 \rangle \), where \( E_3, E_4 \notin \text{SITYPENAME U \{?} \) (Case (D.ii)).

If \( G'' \in \text{HEADTYPE} \), we proceed as the case \( G \in \text{HEADTYPE}, G' \in \text{HEADTYPE} \) with \( \epsilon = \langle E_3, E_4 \rangle \), where \( E_3, E_4 \in \text{HEADTYPE} \) (Case (D.iii)).

(D.ii) \( (G' =?, \epsilon = \langle H_3, H_4 \rangle \)). We have to prove that

\[
\downarrow W, (\epsilon_1 \circ \rho_1(\epsilon))u_1 : ?, (\epsilon_2 \circ \rho_2(\epsilon))u_2 : ?) \in \mathcal{V}_\rho[?]
\]

which is equivalent to prove that

\[
\downarrow W, (\epsilon_1 \circ \rho_1(\epsilon))u_1 : H, (\epsilon_2 \circ \rho_2(\epsilon))u_2 : H) \in \mathcal{V}_\rho[H]
\]

for \( H = \text{const}(H_{i2}) \) (and \( H \in \text{HEADTYPE} \)). But notice that as \( \epsilon \vdash \Xi; \Delta \vdash G \sim ? \), then as \( H_4 \subseteq H \subseteq ? \), then by Lemma E.28, \( \epsilon \vdash \Xi; \Delta \vdash G \sim H \), then we proceed just like the case \( G \in \text{HEADTYPE} \) and \( G' \in \text{HEADTYPE} \) (Case (D.iii)).

(D.iii) \( (G' \in \text{HEADTYPE}) \). This cases are already analyzed, by structural analysis of types, e.g.

(Case \( G = G_1 \rightarrow G_2 \) and \( G' = G_1' \rightarrow G_2' \), (Case \( G = \forall X.G_1 \) and \( G' = \forall X.G_1' \), etc.

\[\text{Case (G = B and G' = B). We know that v_1 has the form \langle B, B \rangle u : B, and we know that (W, \langle B, B \rangle u : B) \in \mathcal{V}_\rho[\beta]. \]Also as \( \epsilon \vdash \Xi; \Delta \vdash B \sim B \), then \( \epsilon = \langle B, B \rangle \), then as \( \rho_1(B) = B, \epsilon_1 \circ \rho_1(\epsilon) = \epsilon_1 \), and we have to prove that \( (W, \langle B, B \rangle u : B, \langle B, B \rangle u : B) \in \mathcal{V}_\rho[\beta] \), which is part of the premise and the result holds.}

\[\square\]

Lemma E.19 (Associativity of the Evidence).

\[ (\epsilon_1 \circ \epsilon_2) \circ \epsilon_3 = \epsilon_1 \circ (\epsilon_2 \circ \epsilon_3) \]

Proof. By induction on the structure of evidences.

\[\text{Case (\epsilon_1 = \langle E_{11}, \alpha^{E_{i2}} \rangle, \epsilon_2 = \langle \alpha^{E_{i1}}, E_{22} \rangle, \epsilon_3 = \langle E_{31}, E_{32} \rangle). By definition of consistent transitivity, we know that} \]

\[\]
• \((\epsilon_1 \circ \epsilon_2) \circ \epsilon_3 = (\langle E_{11}, E_{12} \rangle \circ \langle E_{21}, E_{22} \rangle) \circ \langle E_{31}, E_{32} \rangle\)
• \(\epsilon_1 \circ (\epsilon_2 \circ \epsilon_3) = \langle E_{11}, E_{12} \rangle \circ (\langle E_{21}, E_{22} \rangle \circ \langle E_{31}, E_{32} \rangle)\)

Then by the induction hypothesis \(\langle E_{11}, E_{12} \rangle \circ (\langle E_{21}, E_{22} \rangle \circ \langle E_{31}, E_{32} \rangle) = \langle E_{11}, E_{12} \rangle \circ \langle E_{21}, E_{22} \rangle \circ \langle E_{31}, E_{32} \rangle\), and the result follows immediately.

**Case** \(\epsilon_1 = \langle E_{11}, E_{12} \rangle, \epsilon_2 = \langle E_{21}, \alpha^E_{\epsilon_2} \rangle, \epsilon_3 = \langle \alpha^E_{\epsilon_1}, E_{32} \rangle\). Similar to the previous.

**Case** \(\epsilon_1 = \langle \alpha^E_{\epsilon_1}, E_{12} \rangle, \epsilon_2 = \langle E_{21}, E_{22} \rangle, \epsilon_3 = \langle E_{31}, E_{32} \rangle\). By definition of consistent transitivity, we know that

• \((\epsilon_1 \circ \epsilon_2) \circ \epsilon_3 = \langle \alpha^E_{\epsilon_1}, E_{12} \rangle \circ \langle E_{31}, E_{32} \rangle = \langle \alpha^E_{\epsilon_1}, E'_{12} \rangle\), where \(\langle E_{1}, E_{2} \rangle = (\langle E_{11}, E_{12} \rangle \circ \langle E_{21}, E_{22} \rangle)\), \(\langle E'_{1}, E'_{2} \rangle = (\langle E_{11}, E_{12} \rangle \circ \langle E_{21}, E_{22} \rangle) \circ \langle E_{31}, E_{32} \rangle\).

• \(\epsilon_1 \circ (\epsilon_2 \circ \epsilon_3) = \langle \alpha^E_{\epsilon_1}, E_{12} \rangle \circ (\langle E_{21}, E_{22} \rangle \circ \langle E_{31}, E_{32} \rangle)\)

• Note that by the induction hypothesis \(\langle E'_{1}, E'_{2} \rangle = (\langle E_{11}, E_{12} \rangle \circ \langle E_{21}, E_{22} \rangle) \circ \langle E_{31}, E_{32} \rangle = \langle E_{11}, E_{12} \rangle \circ (\langle E_{21}, E_{22} \rangle \circ \langle E_{31}, E_{32} \rangle)\)

Then, the result follows immediately because \(\langle \alpha^E_{\epsilon_1}, E_{12} \rangle \circ (\langle E_{21}, E_{22} \rangle \circ \langle E_{31}, E_{32} \rangle) = \langle \alpha^E_{\epsilon_1}, E'_{12} \rangle\).

**Case** \(\epsilon_1 = \langle E_{11}, E_{12} \rangle, \epsilon_2 = \langle E_{21}, E_{22} \rangle, \epsilon_3 = \langle E_{31}, \alpha^E_{\epsilon_3} \rangle\). Similar to the previous.

**Case** \(\epsilon_1 = (\epsilon_1, ?), \epsilon_2 = (E_{21}, E_{22}), \epsilon_3 = (E_{31}, E_{32})\). Trivially, by definition of consistent transitivity.

**Case** \(\epsilon_1 = (E_{11}, E_{12}), \epsilon_2 = (\epsilon_1, ?), \epsilon_3 = (E_{31}, E_{32})\). Trivially, by definition of consistent transitivity.

**Case** \(\epsilon_1 = (E_{11}, E_{12}), \epsilon_2 = (E_{21}, E_{22}), \epsilon_3 = (\epsilon_1, ?)\). Trivially, by definition of consistent transitivity.

**Case** \(\epsilon_1 = (E_{11}, E_{12}), \epsilon_2 = (E_{21}, E_{22}), \epsilon_3 = (? , ?)\). Trivially, by definition of consistent transitivity.

The other cases are pretty similar.

**Lemma E.20.** If \((W, t_1, t_2) \in \ell \ell_p \ll G \gg\), then \((\downarrow W, t_1, t_2) \in \ell \ell_p \ll G \gg\)

**Proof.** By definition of \(\ell \ell_p \ll G \gg\).

**Proposition E.21.** \(dom(\epsilon_1 \circ \epsilon_2) = dom(\epsilon_2) \circ dom(\epsilon_1)\)

**Proof.** Direct by inspection on the inductive definition of consistent transitivity.

**Proposition E.22.** \(cod(\epsilon_1 \circ \epsilon_2) = cod(\epsilon_1) \circ cod(\epsilon_2)\)

**Proof.** Direct by inspection on the inductive definition of consistent transitivity.

**Proposition E.23.** \((\epsilon_1 \circ \epsilon_2)[E] = \epsilon_1[E] \circ \epsilon_2[E]\)

**Proof.** Direct by inspection on the inductive definition of consistent transitivity.

**Lemma E.24.** (Optimality of consistent transitivity). If \(\epsilon_1 \circ \epsilon_2\) is defined, then \(\pi_1(\epsilon_3) \subseteq \pi_1(\epsilon_1)\) and \(\pi_2(\epsilon_3) \subseteq \pi_2(\epsilon_2)\).

**Proof.** Direct by inspection on the inductive definition of consistent transitivity.

**Lemma E.25.** If \(\epsilon \vdash \Xi; \Delta \vdash G_1 \sim G_2, W \in \mathcal{S}[\Xi]\) and \((W, \rho) \in \mathcal{D}[\Delta]\) then \(\epsilon^\rho \vdash W, \Xi; \Delta \vdash \rho(G_1) \sim \rho(G_2)\), where \(\epsilon^\rho = \rho_\epsilon(\epsilon)\).

**Proof.** Direct by induction on the structure of the types \(G_1\) and \(G_2\).
**Lemma E.26.** If $\Xi; \Delta; \Gamma \vdash t : G$, $W \in S[\Xi]$, $(W, \rho) \in D[\Delta]$ and $(W, \gamma) \in G[\Gamma]$ then $W, \Xi_1 \vdash \rho(y_1((t)) : \rho(G)$.

**Proof.** Direct by induction on the structure of the term.

**Lemma E.27.** If

- $\epsilon_1 \vdash W, \Xi_1 \vdash G_1 \sim \rho(G), \epsilon_1 \equiv \epsilon_2$
- $\epsilon \vdash \Xi; \Delta \vdash G \sim G'$
- $W \in S[\Xi], (W, \rho) \in D[\Delta]$
- $\forall \alpha \in \Xi. \alpha^{E\times} \in p_2(\epsilon_1) \Rightarrow E_1^\alpha \equiv E_2^\alpha$

then $\epsilon_1 \circ \rho_1(\epsilon) \iff \epsilon_2 \circ \rho_2(\epsilon)$.

**Proof.** We proceed by induction on the judgment $\epsilon_1 \vdash W, \Xi_1 \vdash G_1 \sim G$.

*Case $(\epsilon_1 = \langle B_1, B_1 \rangle)$. Then the result is trivial as by definition of $\epsilon_1 \equiv \epsilon_2$, $B_1 = B_2$, therefore $\epsilon_1 = \epsilon_2$. As $\epsilon$ cannot have free type variables (otherwise the result holds immediately), proving that $\epsilon_1 \circ \epsilon \iff \epsilon_1 \circ \epsilon$ is trivial.

*Case $(\epsilon_1 = \langle ?, ? \rangle)$. As the combination with $\langle ?, ? \rangle$ never produce runtime errors, the result follows immediately as both operation never fail.

*Case $(\epsilon_1 = \langle E_{11}, \alpha^{E_{21}} \rangle)$. We branch on two sub-cases:

- **Case $\alpha \in \Xi$.** Then $\epsilon$ has to have the form $\langle a^{E_3}, E_4 \rangle$, $\langle ?, ? \rangle$ or $\langle ?, \beta^{\cdots} \rangle$ (otherwise the transitivity operator will always fails in both branches). Also $E_4$ cannot be a type variable $X$ for instance, because $X$ is consistent with only $X$ or $?$, and in either case the evidence gives you $X$ on both sides of the evidence. And $\alpha$ cannot point to a type variable by construction (e.g., type $\alpha^X$ does not exists). Then $\epsilon$ cannot have free type variables, therefore $\rho_1(\epsilon) = \epsilon$, and therefore we have to prove: $\epsilon_1 \circ \epsilon \iff \epsilon_2 \circ \epsilon$. For cases where $\epsilon = \langle ?, ? \rangle$ or $\epsilon = \langle ?, \beta^{\cdots} \rangle$, then as they never produce runtime errors, the result follows immediately as both operation never fail.

The interesting case is $\epsilon = \langle a^{E_3}, E_4 \rangle$. By definition of transitivity $\langle E_{11}, \alpha^{E_{21}} \rangle \circ \langle a^{E_3}, E_4 \rangle = \langle E_{11}, E_{21} \rangle \circ \langle E_3, E_4 \rangle$. By Lemma E.30, $\langle E_{11}, E_{21} \rangle \vdash W, \Xi_1 \vdash G_1 \sim \Xi(\alpha)$ and $\langle E_3, E_4 \rangle \vdash W, \Xi_1 \vdash \Xi(\alpha) \sim G'$. Also we know by premise that $E_{21} \equiv E_{21}$, then by induction hypothesis $\langle E_{11}, E_{21} \rangle \circ \langle E_3, E_4 \rangle \iff \langle E_{12}, E_{22} \rangle \circ \langle E_3, E_4 \rangle$, and the result follows immediately.

- **Case $\alpha \notin \Xi$.** In this case $\epsilon$ has to have the form $\langle X, X \rangle$ (where $\rho_1(\epsilon) = \langle \text{lift}_{W, \Xi_1}(\alpha), \text{lift}_{W, \Xi_1}(\alpha) \rangle$), $\langle ?, ? \rangle$ or $\langle ?, \beta^{\cdots} \rangle$, (otherwise the transitivity always fail in both cases). For cases where $\epsilon = \langle ?, ? \rangle$ or $\epsilon = \langle ?, \beta^{\cdots} \rangle$, by the definition of transitivity, they never produce runtime errors, then the result follows immediately as both operation never fail.

If $\epsilon = \langle X, X \rangle$, by construction of evidence, $\alpha^{E_{21}} \subseteq \text{lift}_{W, \Xi_1}(\alpha) \subseteq \Xi, \Delta \vdash G \sim G'$.

*Case $(\epsilon_1 = \langle a^{E_{11}}, E_{12} \rangle)$. Then $\epsilon$ has the form $\langle E_3, E_4 \rangle$, where $\rho_1(\epsilon) = \langle E_{13}, E_{14} \rangle$. By the definition of transitivity we know that:

$$\langle a^{E_{11}}, E_{12} \rangle \circ \langle E_{13}, E_{14} \rangle \iff \langle E_{11}, E_{12} \rangle \circ \langle E_{13}, E_{14} \rangle$$

Then by the induction hypothesis with:

$$\langle E_{11}, E_{12} \rangle \vdash W, \Xi_1 \vdash W, \Xi_1(\alpha) \sim \rho(G)$$

$$\epsilon \vdash \Xi; \Delta \vdash G \sim G'$$
we know that:
\[ \langle E_{11}, E_{22} \rangle \circ \langle E_{13}, E_{14} \rangle \iff \langle E_{21}, E_{22} \rangle \circ \langle E_{23}, E_{24} \rangle \]
Then the result follows immediately.

**Case** \( \epsilon_1 = \langle E_{11} \rightarrow E_{121}, E_{211} \rightarrow E_{221} \rangle \). We analyze cases for \( \epsilon \):

- **Case** \( \epsilon = \langle ?, ? \rangle \) or \( \epsilon = \langle ?, \beta \cdot \cdot \cdot \rangle \), then transitivity never fails as explained in previous cases.
- **Case** \( \epsilon = \langle E_{31} \rightarrow E_{32}, E_{41} \rightarrow E_{42} \rangle \). Then \( \rho_1(\epsilon) = \langle E_{311} \rightarrow E_{321}, E_{411} \rightarrow E_{421} \rangle \). By definition of interior and meet, the definition of transitivity for functions, can be rewriten like this:

\[
\langle E_{411}, E_{311} \rangle \circ \langle E_{211}, E_{111} \rangle = \langle E_{13}, E_{i1} \rangle \quad \langle E_{121}, E_{221} \rangle \circ \langle E_{321}, E_{421} \rangle = \langle E_{412}, E_{41i} \rangle
\]

Also notice as the definition of interior is symmetrical (as consistency is symmetric), \( \langle E_{411}, E_{311} \rangle \circ \langle E_{211}, E_{111} \rangle = \langle E_{13}, E_{i1} \rangle \) can be computed as \( \langle E_{111}, E_{211} \rangle \circ \langle E_{311}, E_{411} \rangle = \langle E_{i1}, E_{i3} \rangle \).

Also \( \epsilon_1 = \epsilon_2 \) implies that \( \text{dom}(\epsilon_1) = \text{dom}(\epsilon_2) \) and \( \text{cod}(\epsilon_1) = \text{cod}(\epsilon_2) \). And that \( \text{dom}(\epsilon) \vdash \Xi; \Delta \vdash \text{dom}(G') \sim \text{dom}(G') \) is equivalent to:

\[
(\pi_2(\text{dom}(\epsilon)), \pi_1(\text{dom}(\epsilon))) \vdash \Xi; \Delta \vdash \text{dom}(G') \sim \text{dom}(G')
\]

where \( \text{cod}(\epsilon) \vdash \Xi; \Delta \vdash \text{cod}(G) \sim \text{cod}(G') \). The result holds by applying induction hypothesis on:

\[
\langle E_{111}, E_{211} \rangle \vdash \Xi; \Delta \vdash \text{dom}(G_I) \sim \text{dom}(\rho(G))
\]

\[
(\pi_2(\text{dom}(\epsilon)), \pi_1(\text{dom}(\epsilon))) \vdash \Xi; \Delta \vdash \text{dom}(G) \sim \text{dom}(G')
\]

- **Case** \( \epsilon = \langle E_{31} \rightarrow E_{32}, \alpha^{E_{4i} \rightarrow E_{4i}} \rangle \). Then \( \rho_1(\epsilon) = \langle E_{311} \rightarrow E_{321}, \alpha^{E_{4i} \rightarrow E_{4i}} \rangle \). We use a similar argument to the previous item noticing that

\[
\langle E_{411}, E_{311} \rangle \circ \langle E_{211}, E_{111} \rangle = \langle E_{13}, E_{i1} \rangle \quad \langle E_{121}, E_{221} \rangle \circ \langle E_{321}, E_{421} \rangle = \langle E_{412}, E_{41i} \rangle
\]

and if \( G' = \alpha \) by Lemma E.30

\[
\langle E_{31} \rightarrow E_{32}, E_{41} \rightarrow E_{42} \rangle \vdash \Xi; \Delta \vdash G \sim \alpha
\]

and if \( G' = ? \) by Lemma E.30

\[
\langle E_{31} \rightarrow E_{32}, E_{41} \rightarrow E_{42} \rangle \vdash \Xi; \Delta \vdash G \sim ?
\]

**Case** \( \epsilon_1 = \langle \forall X. E_{11}, \forall X. E_{211} \rangle \).

We proceed similar to the function case using induction hypothesis on the subtypes.

**Case** \( \epsilon_1 = \langle E_{11} \times E_{21}, E_{31} \times E_{41} \rangle \).

We proceed similar to the function case using induction hypothesis on the subtypes.

\[ \square \]

**Lemma E.28.** If \( \langle E_1, E_2 \rangle \vdash \Xi; \Delta \vdash G_1 \sim G_2 \), then

1. \( \forall G_3, \text{toGType}(E_2) \sqsubseteq G_3 \sqsubseteq G_2, \langle E_1, E_2 \rangle \vdash \Xi; \Delta \vdash G_1 \sim G_3 \), and
In this section we show that the logical relation is sound with respect to contextual approximation and therefore contextual equivalence. Figure 12 presents the syntax and static semantics of contexts.

**E.3 Contextual Equivalence**

In this section we show that the logical relation is sound with respect to contextual approximation and therefore contextual equivalence. Figure 12 presents the syntax and static semantics of contexts.

**Definition E.32 (Contextual Approximation and Equivalence).**

\[
\begin{align*}
\Xi; \Delta; \Gamma \vdash t_1 \leq \text{ctx} t_2 : G \triangleq \Xi; \Delta; \Gamma \vdash t_1 : G \land \forall C, \Xi' \vdash G'. \\
& \quad + C : (\Xi; \Delta; \Gamma \vdash G) \leadsto (\Xi' \vdash G') \Rightarrow ((\Xi' \vdash t_1 \downarrow \Rightarrow \Xi' \vdash t_2 \uparrow) \land \\
& \quad (\exists \Xi_1, \Xi' \vdash C[t_1] \leadsto * \Xi_1 \triangleright \text{error} \Rightarrow \exists \Xi_2, \Xi' \vdash C[t_2] \leadsto * \Xi_2 \triangleright \text{error}) \\
\Xi; \Delta; \Gamma \vdash t_1 \approx \text{ctx} t_2 : G \triangleq \Xi; \Delta; \Gamma \vdash t_1 \leq \text{ctx} t_2 : G \land \Xi; \Delta; \Gamma \vdash t_2 \leq \text{ctx} t_1 : G
\end{align*}
\]
$$\begin{align*}
C &::= [] \ | \ \epsilon C_u :: G \ | \ (C, t) \ | \ (t, C) \ | \ C \triangleright t \ C \ | \ \epsilon C :: G \ | \ op(\overline{t}, C, \overline{t}) \ | \ C [G] \ | \ \pi_l(C) \quad \text{(GSFe Contexts)} \\
C_u &::= \lambda x : G.C \ | \ \Lambda X.C \ | \ \langle C_u, u \rangle \ | \ \langle u, C_u \rangle \\
C_s &::= C \mid C_u
\end{align*}$$

Well-typed contexts

$$\begin{align*}
\frac{\Xi \subseteq \Xi'}{\vdash \,[\cdot] : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G)} \\
\frac{\vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G)}{
\vdash \lambda x : G.C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G)} \\
\frac{\vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G + \forall X.G')}{
\vdash \Delta X.C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + \forall X.G')} \\
\frac{\vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G) \quad \Xi; \Delta \vdash G \quad \Xi; \Delta \vdash G'}{\vdash \Delta; \Gamma + G \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G' + t : G_2) \\
\vdash \langle C, t \rangle : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + \forall X.G')} \\
\frac{\vdash \epsilon C_3 : G'' : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G'')}{\vdash \epsilon C_3 : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G'')} \\
\frac{\Xi'; \Delta'; \Gamma' + \overline{t_1} : G_1 \quad \vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_2)}{\Xi'; \Delta'; \Gamma' + \overline{t_2} : G_2 \quad \text{ty}(op) = (\overline{G_1}, G_3, \overline{G_2}) \quad \rightarrow G''} \\
\frac{\vdash \epsilon C_1 : G' : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G'')}{
\vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_1 + G_2) \\
\vdash \Delta X.G : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G' + [G''/X])} \\
\frac{\vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_1 \times G_2)}{
\vdash \pi_l(C) : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_1)} \\
\frac{\vdash \pi_l(C) : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_1)}{
\vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_1 \times G_2)} \\
\frac{\vdash C : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_1 \times G_2)}{
\vdash \pi_l(C) : (\Xi; \Delta; \Gamma + G) \rightsquigarrow (\Xi'; \Delta'; \Gamma' + G_1)}
\end{align*}$$

Fig. 12. GSFe: Syntax and Static Semantics - Contexts

**Theorem E.33 (Soundness w.r.t. Contextual Approximation).** If \(\Xi; \Delta; \Gamma \vdash t_1 \leq t_2 : G\) then \(\Xi; \Delta; \Gamma \vdash t_1 \leq_{ctx} t_2 : G\).

**Proof.** The proof follows the usual route of going through congruence and adequacy. \(\square\)


F GSF: IMPRECISE TERMINATION

In this section we present the proof of the fundamental property of the imprecise termination logical predicate in Figure 7, and the proof of Lemma 10.1.

Throughout these proofs we assume that liftₓ(T) = ̂T (we omit the Σ notation when obvious from the context).

**Proposition F.1.** Let t be a static term. If Σ; Δ; Γ ⊢ t : T ⊑ G, ε ⊨ Σ; Δ ⊢ T ∼ G, and ε = ⟨̂T, ̂T⟩, then Σ; Δ; Γ ⊨ e t : G : T ⊑ G.

**Proof.** By induction on the type derivation of t. Note that all the subterms of t are also static.

**Case (Eb).** If t = ε'b :: T then:

\[ \frac{ty(b) = B}{Σ; Δ; Γ ⊢ b : B} \]

**Case (El).** If t = ελ(λx : T₁,t') :: T₁' → T₂' then we know that:

\[ Σ; Δ; Γ ⊢ ελ(λx : T₁,t') :: T₁' → T₂' : t₂' : T₂' → T₂' \]

Then we have to prove that:

\[ Σ; Δ; Γ ⊨ ε(ελ(λx : T₁,t')) :: T₁' → T₂' : G : T₁' → T₂' ⊑ G \]

Then after the usual unfoldings we have to prove that, for all ρ ∈ D^Σ[Δ], γ ∈ G^Σρ[Γ] :

\[ ρ(ε)(ρ(ελ(λx : T₁,t')))(ρ(γ(t')))) : ρ(T₁') → ρ(T₂') : ρ(G) ∈ C^ρ[Σ ; T₁' → T₂' ⊑ G] \]

Suppose that T₁₁ = ρ(T₁₁), T₁₂ = ρ(T₁₂), T₁₂ = ρ(T₁₂), T₂₁ = ρ(T₂₁), and T₂₂ = ρ(T₂₂), then ρ(≤) = ⟨̂T₂₁ → ̂T₂₂, ̂T₂₁ → ̂T₂₂⟩, and ρ(ελ) = ⟨̂T₁₁ → ̂T₁₂, ̂T₁₂ → ̂T₁₂⟩. Then by Lemma E.31, ρ(ελ) → ρ(ε) = ρ(ελ) = ⟨̂T₁₁ → ̂T₁₂, ̂T₁₂ → ̂T₁₂⟩. Then we have to prove that:

\[ ρ(ελ)(λx : T₁,ρ(γ(t')))) : ρ(G) ∈ N^Σρ[Σ ; T₁' → T₂' ⊑ G] \]

Let G₁ = dom^#(ρ(G)) and G₂ = cod^#(ρ(G)). We have to prove that for all υ' ∈ N^Σρ[Σ ; T₁' ⊑ dom^#(G)] it is true that:

\[ (ρ(ελ)(λx : T₁,ρ(γ(t')))) : G₁ → G₂) υ' ∈ C^ρ[Σ ; T₂' ⊑ cod^#(G)] \]

Let υ = ευ u :: G₁. By the reduction rules we know that:

\[ Σ ⊢ ρ(ελ)(λx : T₁,ρ(γ(t')))) : G₁ → G₂) υ' \]

Note that γ' = γ(x : T₁) → (ευ u :: dom(ελ)) u :: ρ(T₁) ∈ G^Σρ[Γ,x :: T₁]. Then by the induction hypothesis on t', with ρ, γ', and ⟨T₁₁, T₁₂⟩, we know that:

\[ (ρ(T₁₁, T₁₂))ρ(γ(t')) : ρ(T₂) \]

Note that ρ(γ(t'))((υ :: dom(ελ)) u :: ρ(T₁))/x = ρ(γ(t')). Then by Lemma F.7 the result holds.

**Case.** If t = ελAX.t' :: ∀X.T₁ then:

\[ \frac{Σ; Δ; Γ ⊢ ελAX.t' :: ∀X.T₁}{Σ; Δ; Δ; Γ ⊢ ελAX.t' :: ∀X.T₁ :: ∀X.T₁} \]

Then we have to prove that:

\[ Σ; Δ; Γ ⊨ ελAX.t' :: ∀X.T₁ :: G : ∀X.T₁ ⊑ G \]
Matías Toro, Elizabeth Labrada, and Éric Tanter

Then after the usual unfoldings we have to prove that, for some \( \rho \in D^\Sigma [\Delta], \gamma \in G_\rho^\Sigma [\Gamma] \):

\[
\rho(\epsilon)(\rho(\epsilon)\Delta X.\rho(y(t')) :: \forall X.\rho(T_1)) :: \rho(G) \in C_\rho^\Sigma [\forall X. T_1 \sqsubseteq G]
\]

Suppose that \( T'_1 = \rho(T_1) \), then \( \rho(\epsilon) = \langle \forall X. T'_1, \forall X. T'_1 \rangle \), and that \( \rho(\epsilon) = \langle \forall X. T'_2, \forall X. T'_1 \rangle \). Then by Lemma E.31, \( \rho(\epsilon) \circ \rho(\epsilon) = \langle \forall X. T'_2, \forall X. T'_1 \rangle \). Then we have to prove that

\[
(\rho(\epsilon)\Delta X.\rho(y(t'))) :: \rho(G) \in C_\rho^\Sigma [\forall X. T_1 \sqsubseteq G]
\]

Let some \( T' \) such that \( \Sigma \vdash T' \), posing \( G_1 = \text{schn}^\gamma(\rho(G)) \), then

\[
\Sigma \bullet (\rho(\epsilon)\Delta X.\rho(y(t'))) :: \forall X. G_1 [T''] \infer \Sigma, \alpha := T'\bullet (E_1[a^{T'}/X], E_1[T'/X])((\bar{T}'_2[a^{T'}/X], \bar{T}'_1[a^{T'}/X])\rho(y(t'))[\alpha^{T'}/X] :: G_1[\alpha/X]) :: G_1[T'/X]
\]

where \( E_1 = \text{lift}_{\bar{T}}(G_1) \), and \( \bar{T}' = \text{lift}_{\bar{T}}(T') \). Note that \( \text{schn}(\epsilon) \models \Sigma; \Delta, X \vdash T_1 \sim \text{schn}^\gamma(G) \). Now we have to prove that

\[
((\bar{T}'_2[a^{T'}/X], \bar{T}'_1[a^{T'}/X])\rho(y(t'))[\alpha^{T'}/X] :: G_1[\alpha/X]) \in C_{\rho'}^\Sigma [T_1 \sqsubseteq G_1]
\]

But note that \( \rho(y(t'))[\alpha^{T'}/X] = \rho'(y(t')) \), then we use induction hypothesis on \( t' \), with \( \rho', \gamma \), and \( \epsilon = \langle \bar{T}_2, \bar{T}_2 \rangle \), where \( \bar{T}_2 = \text{lift}_{\bar{T}}(T_2) \). Then

\[
(\rho'(\langle \bar{T}_2, \bar{T}_2 \rangle)) \rho'(y(t')) :: \rho'(T_2)) \in C_{\rho'}^\Sigma [T_2 \sqsubseteq T_2]
\]

and thus, posing \( \bar{T}''_2 = \rho'(\bar{T}_2) \)

\[
\Sigma' \bullet ((\bar{T}''_2, \bar{T}''_2) \rho'(y(t')) :: \rho'(T_2)) \infer \Sigma'' \bullet ((\bar{T}_3, \bar{T}''_2) u :: \rho'(T_2))
\]

and \( \langle \bar{T}_3, \bar{T}''_2 \rangle u :: \rho'(T_2) \in N_{\rho'}^\Sigma [T_2 \sqsubseteq T_2] \). Then by Lemma F.7, as \( \bar{T}''_2[a^{T'}/X] = \bar{T}''_2 \), then

\[
\langle \bar{T}_3, \bar{T}_1[a^{T'}/X] \rangle u :: G_1[\alpha/X] \in N_{\rho'}^\Sigma [T_1 \sqsubseteq G_1]
\]

and the result holds.

Case. If \( t = \langle u_1, u_2 \rangle \) then:

\[
\Sigma; \Delta; \Gamma \vdash u_1 : T_1 \quad \Sigma; \Delta; \Gamma \vdash u_2 : T_2 \\
\Sigma; \Delta; \Gamma \vdash \langle u_1, u_2 \rangle : T_1 \times T_2
\]

Then we have to prove that:

\[
\Sigma; \Delta; \Gamma \vdash \epsilon(u_1, u_2) :: G : T_1 \times T_2 \sqsubseteq G
\]

We know that \( p_1(\epsilon) \models \Sigma; \Delta \vdash T_1 \simeq \text{proj}_1^\#(G) \). Then by induction hypotheses we already know that:

\[
\Sigma; \Delta; \Gamma \vdash p_1(\epsilon) u_1 :: \text{proj}_1^\#(G) : T_1 \sqsubseteq \text{proj}_1^\#(G) \text{ and } \Sigma; \Delta; \Gamma \vdash p_2(\epsilon) u_2 :: \text{proj}_2^\#(G) : T_2 \sqsubseteq \text{proj}_2^\#(G) \text{. But the result follows directly by Prop E.13 and E.14 (compatibility of pairs).}
\]

Case (Easc). Then \( t = \epsilon' t' : T \), and therefore:

\[
\Sigma; \Delta; \Gamma \vdash \epsilon' t' : T' \quad \epsilon' \vdash \Sigma; \Delta \vdash T' \simeq T \\
\Sigma; \Delta; \Gamma \vdash \epsilon' t' :: T : T
\]

By induction hypotheses we already know that \( \Sigma; \Delta; \Gamma \vdash \epsilon' t' :: T : T \subseteq T \), then the result follows directly by Prop E.9 (Compatibility of ascriptions).
Case (Epair). Then \( t = \langle t_1, t_2 \rangle \), and therefore:

\[
\frac{\Sigma; \Delta; \Gamma \vdash t_1 : G_1 \quad \Sigma; \Delta; \Gamma \vdash t_2 : G_2}{\Sigma; \Delta; \Gamma \vdash \langle t_1, t_2 \rangle : G_1 \times G_2}
\]

where \( G = G_1 \times G_2 \). Then we have to prove that:

\[
\Xi; \Delta; \Gamma \vdash \langle t_1, t_2 \rangle \leq \langle t_1, t_2 \rangle : G_1 \times G_2
\]

By induction hypotheses we already know that: \( \Xi; \Delta; \Gamma \vdash t_1 \leq t_1 : G_1 \) and \( \Xi; \Delta; \Gamma \vdash t_2 \leq t_2 : G_2 \). But the result follows directly by Prop E.7 (Compatibility of pairs).

Case (Ex). Then \( t = x \), and therefore:

\[
\frac{x : G \in \Gamma \quad \Xi; \Delta; \Gamma \vdash x : G}{\Sigma; \Delta; \Gamma \vdash x : G}
\]

Then we have to prove that \( \Xi; \Delta; \Gamma \vdash x \leq x : G \). But the result follows directly by Prop E.8 (Compatibility of variables).

Case (Eop). Then \( t = \text{op}(\overline{t}) \), and therefore:

\[
\frac{\Sigma; \Delta; \Gamma \vdash \overline{t} : \overline{G} \quad \text{ty}(\text{op}) = \overline{G} \rightarrow G}{\Sigma; \Delta; \Gamma \vdash \text{op}(\overline{t}) : G}
\]

Then we have to prove that: \( \Xi; \Delta; \Gamma \vdash \text{op}(\overline{t}) \leq \overline{t} : \overline{G} \). Then the result follows directly by Prop E.10 (Compatibility of app operator).

Case (Eapp). Then \( t = t_1 \, t_2 \), and therefore:

\[
\frac{\Sigma; \Delta; \Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Sigma; \Delta; \Gamma \vdash t_2 : T_{11}}{\Sigma; \Delta; \Gamma \vdash t_1 \, t_2 : T_{12}}
\]

where \( T = T_{12} \). Then we have to prove that:

\[
\Sigma; \Delta; \Gamma \vdash \varepsilon(t_1, t_2) : G : T_{12} \subseteq G
\]

By induction hypotheses we already know that: \( \Sigma; \Delta; \Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \subseteq T_{11} \subseteq G \) and \( \Sigma; \Delta; \Gamma \vdash t_2 : T_{11} \subseteq T_{11} \). Then the result follows directly by Prop E.11 (Compatibility of term application).

Case (EappG). Then \( t = t'[T_2] \), and therefore:

\[
\frac{\Sigma; \Delta; \Gamma \vdash t' : \forall X. T_1 \quad \Xi; \Delta \vdash T_2}{\Sigma; \Delta; \Gamma \vdash t'[T_2] : T_1[T_2/X]}
\]

where \( T = T_2[X/T] \). Then after the usual unfoldings we have to prove that, for some \( \rho \in D^X[\Delta], \gamma \in G^X_\rho[\Gamma] \):

\[
\rho(\varepsilon)(\rho(\gamma(t'))(\rho(T_2))) : \rho(G) \in C_\rho^X[T_1[T_2/X] \subseteq G]
\]

Note that \( \varepsilon_{\forall X. T} \vdash \Sigma; \Delta \vdash \forall X. T_1 \sim \forall X. T_1 \). Then by induction hypotheses we know that \( \varepsilon_{\forall X. \rho(T_1)}(\rho(\gamma(t'))) : \forall X. \rho(T_1) \in C_\rho^X[\forall X. T_1 \subseteq \forall X. T_1] \), let \( T'_1 = \rho(T_1) \), then

\[
\Sigma' \vdash \varepsilon_{\forall X. T_1'}(\rho(\gamma(t'))) : \forall X. T'_1 \quad \Sigma' \vdash \varepsilon_{\forall X. t''} : \forall X. T'_1'
\]

where \( \varepsilon_{\forall X. T_1'} \vdash \forall X. T'_1' \), and \( \varepsilon_{\forall X. t''} : \forall X. T'_1' \in C_\rho^X[\forall X. T_1' \subseteq \forall X. T_1'] \). If we instantiate the last interpretation with \( T' = \rho(T_2) \), then we know that

\[
\Sigma' \vdash \varepsilon_{\forall X. t''} : \forall X. \rho(T_1)[\rho(T_2)]
\]

where \( \varepsilon_{\forall X. t''} \vdash \forall X. \rho(T_1)[\rho(T_2)] \rightarrow^* \Sigma', \alpha := \rho(T_2) \mapsto \langle T_1'[\alpha/T]/X, \tilde{T}_1'[\tilde{T}/X]\rangle(\tilde{T}_1'[\tilde{T}/X])t''[\alpha \tilde{T}/X] : T_1'[\alpha/X]) \vdash T_1'[T'/X] \]

where \( T' = \text{lift}_{\Sigma'}(T'), \tilde{T}_1' = \text{lift}_{\Sigma'}(T_1') \), and

\[
(\langle T_1'[\alpha \tilde{T}/X], \tilde{T}_1'[\hat{T}/X]\rangle)t''[\alpha \tilde{T}/X] : T_1'[\alpha/X]) \in C_\rho^{\Sigma'}[T_1' \subseteq T_1']
\]
for $\Sigma'' = \Sigma', \alpha := T'$, and $\rho' = \rho, X \leftrightarrow \alpha$. Let $t'' = (\langle \tilde{T}_3', \tilde{T}_1'[\alpha^T/X], \tilde{T}_1'[\alpha^T/X] \rangle t'' \mid \alpha^T/X \vdash T_1'[\alpha/X])$, then $\Sigma'' \triangleright t'' \dashv\vdash \Sigma'' \triangleright \langle \tilde{T}_3, \tilde{T}_1'[\alpha^T/X] \rangle u : T_1'[\alpha/X]$, and $(\tilde{T}_3, \tilde{T}_1'[\alpha^T/X])u : T_1'[\alpha/X] \in N_\rho^\Sigma''[T_1 \subseteq T_1]$. Then we have to prove that

$$
\langle \tilde{T}_3, \tilde{T}_1'[\alpha^T/X] \rangle \circ (\tilde{T}_3', \tilde{T}_1'[\alpha^T/X], \tilde{T}_1'[\tilde{T}'/X])u : T_1'[T'/X] \in N_{\rho'}^\Sigma''[T_1[T_2/X] \subseteq T_1[T_2/X]]
$$

which follows from compositionality (Prop F.8). Then

$$
(\tilde{T}_3, \tilde{T}_1'[\tilde{T}'/X])u : T_1'[T'/X] \in N_\rho^\Sigma''[T_1[T_2/X] \subseteq T_1[T_2/X]]
$$

But $\rho(\varepsilon) = (\tilde{T}_1'[\tilde{T}'/X], \tilde{T}_1'[\tilde{T}'/X])$, and the result holds by Lemma F.7.

**Case (Epair1).** Then $t = \pi_1(t')$, and therefore:

$$
\Sigma; \Delta; \Gamma \vdash t' : G_1 \times G_2 \\
\Sigma; \Delta; \Gamma \vdash \pi_1(t') : G_1
$$

where $G = G_1$. Then we have to prove that: $\Sigma; \Delta; \Gamma \vdash \pi_1(t') \leq \pi_1(t') : G_1$. By the induction hypothesis we obtain that: $\Sigma; \Delta; \Gamma \vdash \pi_1(t') \leq \pi_1(t') : G_1 \times G_2$. Then the result follows directly by Prop E.13 (Compatibility of access to the first component of the pair).

**Case (Epair2).** Then $t = \pi_2(t')$, and therefore:

$$
\Sigma; \Delta; \Gamma \vdash t' : G_1 \times G_2 \\
\Sigma; \Delta; \Gamma \vdash \pi_2(t') : G_2
$$

where $G = G_2$. Then we have to prove that: $\Sigma; \Delta; \Gamma \vdash \pi_2(t') \leq \pi_2(t') : G_2$. By the induction hypothesis we obtain that: $\Sigma; \Delta; \Gamma \vdash \pi_2(t') \leq \pi_2(t') : G_1 \times G_2$. Then the result follows directly by Prop E.14 (Compatibility of access to the second component of the pair).

\[\blacksquare\]

**Lemma F.2.** If $T \subseteq G$ and $\varepsilon \vdash \Sigma; \Delta \vdash T \sim G$ then $\varepsilon = (\text{lift}_\Sigma(T), \text{lift}_\Sigma(T))$.

**Proof.** By induction on the structure of the type $T$, and the definition of $\subseteq, \Sigma; \Delta \vdash \cdot \sim \cdot$ and $\text{lift}_\Sigma(\cdot)$.

\[\blacksquare\]

**Lemma F.3.** If $t \in C_\rho^\Sigma[T \subseteq G]$ and $\Sigma \subseteq \Sigma'$ then $t \in C_\rho^{\Sigma'}[T \subseteq G]$.

**Proof.** By the definition of $C_\rho^\Sigma[\cdot \subseteq \cdot]$.

\[\blacksquare\]

**Lemma F.4.** $I_\Sigma(G, G) = (\text{lift}_\Sigma(G), \text{lift}_\Sigma(G))$.

**Proof.** By induction on the structure of the type $G$, and the definition of $I_\Sigma(\cdot, \cdot)$ and $\text{lift}_\Sigma(\cdot)$.

\[\blacksquare\]

**Lemma F.5.** If $G_1 \subseteq G_2$ then $\text{lift}_\Sigma(G_1) \subseteq \text{lift}_\Sigma(G_2)$.

**Proof.** By induction on the structure of the type $G$, and the definition of $\subseteq$ and $\text{lift}_\Sigma(\cdot)$.

\[\blacksquare\]

**Lemma F.6.** $\langle \tilde{T}_1, \tilde{T}_2 \rangle \circ \langle \tilde{T}_3, \tilde{T}_4 \rangle = \langle \tilde{T}_1, \tilde{T}_3 \rangle$.

**Proof.** By induction on the structure of the evidences, noticing that every evidence is static, so you cannot gain precision on the resulting values (Optimality Lemma E.24).

\[\blacksquare\]

**Lemma F.7 (Lemma asc).** If $\langle \tilde{T}_1, \tilde{T}_2 \parallel \Sigma; \vdash \rho(T_1) \sim \rho(G), \langle \tilde{T}_2, \tilde{T}_3 \parallel \Sigma; \vdash \rho(G) \sim \rho(G'), T_3 \subseteq G', and \langle \tilde{T}_1, \tilde{T}_2 \rangle u : \rho(G) \in N_\rho^{\Sigma}[T_2 \subseteq G]$ then $\langle \tilde{T}_1, \tilde{T}_3 \rangle u : \rho(G') \in N_\rho^{\Sigma}[T_3 \subseteq G']$.
PROOF. We proceed by induction on evidences \( \langle \tilde{T}_1, \tilde{T}_2 \rangle \) and \( \langle \tilde{T}_2, \tilde{T}_3 \rangle \). For simplicity, we omit type substitution \( \rho \), when is not important.

Case \( \langle \tilde{T}_2 = \alpha \tilde{T}_2 \rangle \). Notice by inspection of the consistent transitivity rules and Lemma F.6, \( \langle \tilde{T}_1, \alpha \tilde{T}_2 \rangle \circ \langle \alpha \tilde{T}_2, \tilde{T}_3 \rangle = \langle \tilde{T}_1, \tilde{T}_2 \rangle \circ \langle \tilde{T}_2, \tilde{T}_3 \rangle = \langle \tilde{T}_1, \tilde{T}_3 \rangle \).

As \( \alpha \subseteq G \) and \( \langle \tilde{T}_1, \alpha \tilde{T}_2 \rangle \vdash \Sigma; \cdot \vdash T_1 \sim G \), by Lemma E.28, \( \langle \tilde{T}_1, \alpha \tilde{T}_2 \rangle \vdash \Sigma; \cdot \vdash T_1 \sim \alpha \). Then by Lemma E.30, \( \langle \tilde{T}_1, \tilde{T}_2 \rangle \vdash \Sigma; \cdot \vdash T_1 \sim \alpha, \) and \( \langle \tilde{T}_2, \tilde{T}_3 \rangle \vdash \Sigma; \cdot \vdash \Sigma(\alpha) \sim G', \) then the result follows by induction hypothesis on \( \langle \tilde{T}_1, \tilde{T}_2 \rangle \) and \( \langle \tilde{T}_2, \tilde{T}_3 \rangle \).

Case \( \langle \tilde{T}_3 = \alpha \tilde{T}_3 \rangle \). Notice by inspection of the consistent transitivity rules and Lemma F.6, \( \langle \tilde{T}_1, \tilde{T}_2 \rangle \circ \langle \tilde{T}_2, \alpha \tilde{T}_3 \rangle = \langle \tilde{T}_1, \alpha \tilde{T}_3 \rangle \). Then we have to prove that \( \langle \tilde{T}_1, \alpha \tilde{T}_3 \rangle u :: G' \in N^\Sigma_\rho (\alpha \subseteq G') \), which is equivalent to prove that \( \langle \tilde{T}_1, \alpha \tilde{T}_3 \rangle u :: G' \in N^\Sigma_\rho (T'_3 \subseteq G') \), where \( T'_3 \subseteq G' \). But also by Lemma F.6, \( \langle \tilde{T}_1, \tilde{T}_2 \rangle \circ \langle \tilde{T}_2, \tilde{T}_3 \rangle = \langle \tilde{T}_1, \tilde{T}_3 \rangle \).

As \( \alpha \subseteq G' \) and \( \langle \tilde{T}_2, \alpha \tilde{T}_3 \rangle \vdash \Sigma; \cdot \vdash G \sim G' \), by Lemma E.28, \( \langle \tilde{T}_2, \alpha \tilde{T}_3 \rangle \vdash \Sigma; \cdot \vdash G \sim \alpha \). Then by Lemma E.30, \( \langle \tilde{T}_2, \tilde{T}_3 \rangle \vdash \Sigma; \cdot \vdash G \sim T'_3 \), and by Lemma E.28, \( \langle \tilde{T}_2, \tilde{T}_3 \rangle \vdash \Sigma; \cdot \vdash G \sim G' \), then by induction hypothesis on \( \langle \tilde{T}_1, \tilde{T}_2 \rangle \) and \( \langle \tilde{T}_2, \tilde{T}_3 \rangle \) the result follows.

Case \( \langle \tilde{T}_1 = \alpha \tilde{T}_1 \rangle \). This case can never happen as there are no values where the left component of an evidence is a type name.

Case \( \langle \tilde{T}_1 = \tilde{T}_{11} \to \tilde{T}_{12} \rangle \). We know that \( \langle \tilde{T}_{11} \to \tilde{T}_{12}, \tilde{T}_{21} \to \tilde{T}_{22} \rangle (\lambda x : T_{11}.t') :: G \in N^\Sigma_\rho [T_{21} \to T_{22} \subseteq G] \)

Where \( \langle \tilde{T}_{11} \to \tilde{T}_{12}, \tilde{T}_{21} \to \tilde{T}_{22} \rangle \vdash \Sigma; \Delta \vdash T_{11} \to T_{12} \sim G \).

We have to prove that:

\[ \langle \tilde{T}_{11} \to \tilde{T}_{12}, \tilde{T}_{31} \to \tilde{T}_{32} \rangle (\lambda x : T_{11}.t') :: G' \in N^\Sigma_\rho [T_{31} \to T_{32} \subseteq G'] \]

Let \( G_1 \to G_2 = \text{cod}^\#(G') \to \text{dom}^\#(G') \) Then we have to proof that:

\( \forall \epsilon' u' :: G_1 \in N^\Sigma_\rho [T_{31} \subseteq G_1] \Rightarrow \)

\( (\langle T_{11} \to T_{12}, T_{31} \to T_{32} \rangle (\lambda x : T_{11}.t') : G_1 \to G_2) (\epsilon' u' :: G_1) \iff \)

\( \Sigma \triangleright (\langle \tilde{T}_{11} \to \tilde{T}_{12}, \tilde{T}_{31} \to \tilde{T}_{32} \rangle (\lambda x : T_{11}.t') :: G_1 \to G_2) (\epsilon' u' :: G_1) \rightarrow \)

Note that by the reduction rule of application terms, we obtain that:

\( \Sigma \triangleright (\langle \tilde{T}_{12}, T_{32} \rangle (t'((\epsilon' \circ ((T_{31}, T_{21}) \circ \langle \tilde{T}_{21}, \tilde{T}_{11} \rangle)) u' :: \tilde{T}_{11}/x) \rightarrow \)

Note that by Lemma F.6, \( (\langle T_{31}, T_{21} \rangle \circ \langle \tilde{T}_{21}, \tilde{T}_{11} \rangle) = \langle T_{31}, \tilde{T}_{11} \rangle \), and by Lemma E.19, \( \epsilon' \circ ((T_{31}, T_{21}) \circ \langle \tilde{T}_{21}, T_{11} \rangle) = (\epsilon' \circ (T_{31}, T_{21})) \circ (\langle \tilde{T}_{21}, T_{11} \rangle) \). Note that \( \epsilon' = \tilde{T}_{21} \circ (T_{31}, T_{21}) \vdash \Sigma; \cdot \vdash T_{u} \sim \text{dom}^\#(G') \) and \( \langle \tilde{T}_{31}, T_{21} \rangle \vdash \Sigma; \cdot \vdash \text{dom}^\#(G') \sim \text{dom}^\#(G) \), therefore we know that by the induction hypothesis that \( ((\epsilon' \circ (T_{31}, T_{21})) u' :: \text{dom}^\#(G) \in N^\Sigma_\rho [T_{21} \subseteq \text{dom}^\#(G)] \)

We instantiate \( N^\Sigma_\rho [T_{21} \to T_{22} \subseteq G] \) with \( \nu' = ((\epsilon' \circ (T_{31}, T_{21})) u' :: \text{dom}^\#(G) \in N^\Sigma_\rho [T_{21} \subseteq \text{dom}^\#(G)] \), and then we know that:

\( \Sigma \triangleright (\langle \tilde{T}_{11} \to T_{12}, \tilde{T}_{21} \to \tilde{T}_{22} \rangle (\lambda x : T_{11}.t') :: \text{dom}^\#(G) \to \text{cod}^\#(G)) (\epsilon' \circ (T_{31}, T_{21}) u' :: \text{dom}^\#(G)) \rightarrow \)

\( \Sigma \triangleright (\tilde{T}_{12}, \tilde{T}_{22}) (t'([(\epsilon' \circ (T_{31}, T_{21}) \circ (\tilde{T}_{21}, \tilde{T}_{11})) u' :: \tilde{T}_{11}/x] :: \text{dom}^\#(G) \)
The resulting term reduce to value \((\epsilon'' u'': \text{cod}^\#(G)) \in \mathcal{N}_\rho^\Sigma' [T_{22} \subseteq \text{cod}^\#(G)]\), for some \(\Sigma'\), such that \(\Sigma \subseteq \Sigma'\).

But note that by Lemmas E.19 and F.6,

\[\Sigma \triangleright (\tilde{T}_{12}, \tilde{T}_{32})(t'[\epsilon' \circ ((\tilde{T}_{31}, T_{21}) \circ (\tilde{T}_{21}, T_{11}))u' : T_{11})/x]) : G_2\]

\[= \Sigma \triangleright (\tilde{T}_{22}, \tilde{T}_{32})(\tilde{T}_{12}, \tilde{T}_{22})(t'[\epsilon' \circ ((\tilde{T}_{31}, T_{21}) \circ (\tilde{T}_{21}, T_{11}))u' : T_{11})/x]) : \text{cod}^\#(G) \supseteq G_2\]

\[\iff \Sigma' \triangleright (\tilde{T}_{22}, \tilde{T}_{32})(\epsilon'' u'': \text{cod}^\#(G)) : G_2\]

Then the result follows immediately by using induction hypothesis on evidences \(\epsilon''\) and \((\tilde{T}_{22}, \tilde{T}_{32})\).

**Case** \((\tilde{T}_1 = \forall X.\tilde{T}_1)\). We know that:

\[\forall X.\tilde{T}_1, \forall X.\tilde{T}_2(\Lambda X. \rho(t_1)) : \forall X. \rho(T_2) \in \mathcal{N}_{\rho}^\Sigma [\forall X. T_1 \subseteq G]\]

where \((\forall X.\tilde{T}_1, \forall X.\tilde{T}_2) \vdash \Sigma; \Delta \vdash \forall X. \rho(T_1) \sim \rho(G)\).

We have to prove that:

\[\forall X.\tilde{T}_1, \forall X.\tilde{T}_3(\Lambda X. \rho(t_1)) : \forall X. \rho(T_3) \in \mathcal{N}_{\rho}^\Sigma [\forall X. T_3 \subseteq G']\]

Let \(\forall X. G'_1 = \forall X. \text{schm}^\#(G')\). Then for any \(T'\), such that \(\Sigma \vdash T'\), as

\[\Sigma \triangleright (\forall X.\tilde{T}_1, \forall X.\tilde{T}_2)(\Lambda X. \rho(t_1)) : \forall X. \rho(G'_1)[T] \longrightarrow\]

\[\Sigma, \alpha := T' \triangleright \epsilon'((\tilde{T}_1[\alpha^{T_1}/X], \tilde{T}_3[\alpha^{T_2}/X])\rho(t_1)[\alpha^{T_1}/X] : \rho(G'_1)[\alpha/X]) \:: \rho(G'_1)[T'/X]\]

Then we have to prove that

\[((\tilde{T}_1[\alpha^{T_1}/X], \tilde{T}_3[\alpha^{T_2}/X])\rho(t_1)[\alpha^{T_1}/X] : \rho(G'_1)[\alpha/X]) \in C_{\rho}^\Xi'[T_3 \subseteq G'_1]\]

where \(\Xi' = \Xi, \alpha := T', \text{and} \rho' = \rho[X \mapsto \alpha]\). Note that \((\tilde{T}_1[\alpha^{T_1}/X], \tilde{T}_3[\alpha^{T_2}/X]) = (\tilde{T}_1[\alpha^{T_1}/X], \tilde{T}_2[\alpha^{T_2}/X])\circ (\tilde{T}_2[\alpha^{T_1}/X], \tilde{T}_3[\alpha^{T_2}/X]).\)

Let \(\forall X. G_1 = \forall X. \text{schm}^\#(G)\), we instantiate \(\mathcal{N}_{\rho}^\Sigma [\forall X. T_1 \subseteq G]\) with \(T'\), so:

\[\Sigma \triangleright (\forall X.\tilde{T}_1, \forall X.\tilde{T}_2)(\Lambda X. \rho(t_1)) : \forall X. \rho(G'_1)[T] \longrightarrow\]

\[\Sigma, \alpha := T' \triangleright \epsilon'((\tilde{T}_1[\alpha^{T_1}/X], \tilde{T}_2[\alpha^{T_2}/X])\rho(t_1)[\alpha^{T_1}/X] : \rho(G_1)[\alpha/X]) \:: \rho(G_1)[T'/X]\]

and

\[((\tilde{T}_1[\alpha^{T_1}/X], \tilde{T}_2[\alpha^{T_2}/X])\rho(t_1)[\alpha^{T_1}/X] : \rho(G_1)[\alpha/X]) \in C_{\rho}^\Xi'[T_2 \subseteq G_1]\]

therefore

\[\Sigma' \triangleright (\tilde{T}_1[\alpha^{T_1}/X], \tilde{T}_2[\alpha^{T_2}/X])\rho(t_1)[\alpha^{T_1}/X] : \rho(G_1)[\alpha/X] \longrightarrow^*\]

\[\Sigma'' \triangleright (\tilde{T}_u, \tilde{T}_2[\alpha^{T_2}/X])u : \rho(G_1)[\alpha/X]\]

for some \(\tilde{T}_u\), such that \((\tilde{T}_u, \tilde{T}_2[\alpha^{T_2}/X])u : \rho'(G_1) \in \mathcal{N}_{\rho}^\Sigma[T_2 \subseteq G_1].\) Then using analogous arguments as for the function case, as \((\tilde{T}_u, \tilde{T}_2[\alpha^{T_2}/X]) \circ (\tilde{T}_2[\alpha^{T_2}/X], \tilde{T}_3[\alpha^{T_3}/X]) = (\tilde{T}_u, \tilde{T}_3[\alpha^{T_3}/X]),\) then by induction hypothesis using \(\rho'\) (as \(\tilde{T}_1 = \text{lift}_{\Sigma}(\rho(T_1))\), then \(\tilde{T}_1[\alpha^{T_1}/X] = \text{lift}_{\Sigma'}(\rho'(T_1)),\) for \(i \in \{2, 3\}\)) the result holds immediately.

\[\square\]
\textbf{Proposition F.8 (Compositionality).} Let $\rho' = \rho[X \mapsto \alpha]$ and $\hat{T}' = \text{lifte}_2(\rho(T'))$, $\Sigma(\alpha) = \rho(T')$, $I(\text{lifte}_2(\rho(T)))$, $\text{lifte}_2(\rho(T)) = (\hat{T}, \hat{T})$, $\varepsilon = (\hat{T}[\alpha^{T'}/X], \hat{T}[\alpha^{T'}/X])$, $\varepsilon^{-1} = (\hat{T}[\alpha^{T'}/X], \hat{T}[\alpha^{T'}/X])$, such that $\varepsilon + \Sigma + \rho(T[\alpha/X]) \sim \rho(T[T'/X])$, and $\varepsilon^{-1} + \Sigma + \rho(T[T'/X]) \sim \rho(T[\alpha/X])$ then

1. $\varepsilon'u :: \rho'(T) \in N^\Sigma_\rho[T \in T] \Rightarrow (\varepsilon' \circ \varepsilon)u :: \rho'(T[T'/X]) \in N^\Sigma_\rho[T[T'/X] \in T[T'/X]]$

2. $\varepsilon'u :: \rho'(T[T'/X]) \in N^\Sigma_\rho[T[T'/X] \in T[T'/X]] \Rightarrow (\varepsilon' \circ \varepsilon^{-1})u :: \rho'(T) \in N^\Sigma_\rho[T \subseteq T']$

\textbf{Proof.} As everything is static, then we proceed analogous to the compositionality proof for static terms, proving (1) and (2) by induction on $T$. For instance:

\textit{Case ((1), $T = X$).} Let $v = (\hat{T}_1, \alpha^{\hat{T}'})u :: \alpha$. Then we know that

$$\langle \hat{T}_1, \alpha^{\hat{T}'} \rangle u :: \alpha \in N^\Sigma_\rho[X \in X]$$

which is equivalent to

$$\langle \hat{T}_1, \alpha^{\hat{T}'} \rangle u :: \alpha \in N^\Sigma_\rho[\alpha \subseteq \alpha]$$

As $\Sigma(\alpha) = \rho(T')$, we know that:

$$\langle \hat{T}_1, \hat{T}' \rangle u :: \rho(T') \in N^\Sigma_\rho[T' \subseteq T']$$

And as the value does not have $X$ free,

$$\langle \hat{T}_1, \hat{T}' \rangle u :: \rho(T') \in N^\Sigma_\rho[T' \subseteq T']$$

Then $\varepsilon + \Sigma + \alpha \sim \rho(T')$, and $\varepsilon$ has to have the form $\varepsilon = \langle \alpha^{T'}, \hat{T}' \rangle$. Therefore by Lemma E.31:

$$\langle \hat{T}_1, \alpha^{\hat{T}'} \rangle \circ \langle \alpha^{\hat{T}'}, \hat{T}' \rangle = \langle \hat{T}_1, \hat{T}' \rangle$$

and then we have to prove that

$$\langle \hat{T}_1, \hat{T}' \rangle u :: \rho(T') \in N^\Sigma_\rho[T' \subseteq T']$$

which we already know, and the result holds.

\textit{Case ((2), $T = X$).} Let $v = (\hat{T}_1, \hat{T}' \rangle u :: \rho(T')$. Then we know that

$$\langle \hat{T}_1, \hat{T}' \rangle u :: \rho(T') \in N^\Sigma_\rho[T' \subseteq T']$$

and as $X$ is not free:

$$\langle \hat{T}_1, \hat{T}' \rangle u :: \rho(T') \in N^\Sigma_\rho[T' \subseteq T']$$

As $\langle \hat{T}_1, \hat{T}' \rangle \circ \langle \hat{T}', \alpha^{\hat{T}'} \rangle = \langle \hat{T}_1, \alpha^{\hat{T}'} \rangle$, then we have to prove that

$$\langle \hat{T}_1, \alpha^{\hat{T}'} \rangle u :: \alpha \in N^\Sigma_\rho[X \subseteq X]$$

which is equivalent to prove that

$$\langle \hat{T}_1, \hat{T}' \rangle u :: G' \in N^\Sigma_\rho[T' \subseteq G']$$

where $T' \subseteq G'$. But the result holds immediately by premise and Lemma F.7 using $\langle \hat{T}'' \rangle \vdash \Sigma; \Delta \vdash T' \sim G$, where $\hat{T}'' = \text{lifte}_2(T')$.

\qed
In this section we show the proof of the cheap theorem presented in the paper.

**Definition G.1.** Let $X(t, \alpha)$ a predicate that holds if and only if in each evidence of term $t$, if $\alpha$ is present, then it appears on both sides of the evidence and in the same structural position. This predicate is defined inductively as follows:

$$
\forall \varepsilon \in t, X(\varepsilon, \alpha) \\
X(t, \alpha)
$$

where

$$
\begin{align*}
X((\alpha E, \alpha E'), \alpha) & \quad \alpha \notin FTN(E_1) \cup FTN(E_2) \\
X((E_1, E_2), \alpha) & \quad X((E_1, E_3), \alpha) X((E_2, E_4), \alpha) \\
X((E_1, E_3), \alpha) X((E_2, E_4), \alpha) & \quad X((E_1 \times E_2, \alpha) X((\forall \varepsilon X(E_1, \forall X.E_2), \alpha)
\end{align*}
$$

**Lemma G.2.** $\forall W \in S[\Xi], \rho, \gamma, ((W, \rho) \in D[\Lambda] \land (W, \gamma) \in G_\rho[\Gamma])$, such that $\forall \nu \in \text{cod} (\gamma_1), X(\nu, \alpha)$. If $X(\rho(\gamma_1(t_1)), \alpha)$, then $\Xi \triangleright \rho(\gamma_1(t_1)) \iff \Xi' \triangleright t'_1$ and $X(t'_1, \alpha)$

**Proof.** By induction on the structure of $t_1$. The proof is direct by looking at the inductive definition of construction of evidences (interior), noticing that $\forall G, \exists(X, G) = \exists(G, X) = \langle X, X \rangle$. Then by inspection of consistent transitivity we know that, for any evidence of a value $\langle E_1, E_2 \rangle$

$$
\langle E_1, E_2 \rangle \circ (\alpha E, \alpha E') = \langle E_1', \alpha E' \rangle \land E_1' \neq \alpha^* \iff E_2 = \alpha E'' \land E_1 \neq \alpha^*
$$

but if that is the case $\neg(\langle (E_1, E_2), \alpha \rangle)$, which contradicts the premise.

**Theorem 10.2.** Let $\nu \triangleright \Lambda X. \lambda x : ? t$ for some $t$, such that $\triangleright \nu : \forall X. ? \rightarrow X$. Then for any $\triangleright \nu' : G$, we either have $\nu \triangleright [G] \nu' \downarrow \text{error}$ or $\nu \triangleright [G] \nu' \uparrow$.

**Proof.** Let $\triangleright \nu \sim \nu : \forall X. ? \rightarrow X, \triangleright \nu' \sim \nu_1 : ?$. Because $\triangleright \nu \triangleright : \forall X. ? \rightarrow X$ and $\triangleright \nu : ?$, by the fundamental property (Theorem 8.1) we know that

$$
(W_0, \nu_1, \nu_1) \in V_0[\forall X. ? \rightarrow X]
$$

Let $\nu_1 = \nu \triangleright (\forall X. \lambda x : ? t) : \forall X. ? \rightarrow X$, where $\varepsilon \triangleright \cdot : \forall X. ? \rightarrow X \rightarrow \forall X. ? \rightarrow X$, and therefore $\varepsilon = (\forall X. ? \rightarrow X, \forall X. ? \rightarrow X)$.

Note that by the reduction rules we know that

$$
\Xi \triangleright \nu [G] \iff \exists' \triangleright \exists \triangleright (\varepsilon_1(\lambda x : ? t') : ? \rightarrow \alpha) : ? \rightarrow G
$$

for some $t'$, where $\varepsilon_1 = (\varepsilon \triangleright \alpha \triangleright, \varepsilon \triangleright, \varepsilon \triangleright) \rightarrow \alpha \triangleright$.

By definition of $V_0[\forall X. ? \rightarrow X]$ if we pick $G_1 = G_2 = G$, and some $R$, then for some $W_i$ we know that $(W_1, \nu_1, \nu_2) \in V_{X \rightarrow \alpha} ? \rightarrow X$, where $\nu_i = \varepsilon_2(\lambda x : ? t') : ? \rightarrow \alpha$.

Also, by the reduction rules we know that $\Xi \triangleright \nu [\forall X. ? \rightarrow G] \iff \exists' \triangleright \varepsilon_1(\lambda x : ? t') : ? \rightarrow \alpha$.

As $\text{dom} (\varepsilon_1) = (\text{dom} (\nu_1) : ? t) : ? \rightarrow \alpha \triangleright$.

Also we know that $X(\nu_1, \alpha)$. Then by Lemma G.2, if $\Xi \triangleright t' [\nu_1] \rightarrow \exists' \triangleright \nu_1', \nu_1 \triangleright X(\nu_1', \alpha)$, but that is a contradiction because if $(W_4, \nu_1', \nu_2') \in V_\rho [\alpha]$, then $\neg X(\nu_1', \alpha)$ and the result holds.