UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

GRADUAL SENSITIVITY TYPES

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN
MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

DAMIÁN NICOLÁS ARQUEZ TRIGO

PROFESOR GUÍA:
ÉRIC TANTER

PROFESOR CO-GUÍA:
MATÍAS TORO IPINZA

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
FEDERICO OLMEDO BERÓN
RAIMIL CRUZ CONCEPCIÓN

Este trabajo ha sido parcialmente financiado por:
Fondecyt regular 1190058 y ANID - Programa Iniciativa Científica Milenio - Código ICN17_002

SANTIAGO DE CHILE
2021

Resumen

Los sistemas de tipos con sensibilidad son utilizados para razonar sobre la sensibilidad de computaciones. Esto es de particular interés en el campo de privacidad, especialmente en privacidad diferencial. Una particularidad de los tipos con sensibilidad es que, al ser una disciplina de tipado, restringen al programador a lidiar estáticamente con a veces complejos y usualmente restricciones conservadoras, impuestas por los tipos. Esto puede conducir rápidamente a una compleja experiencia de desarrollo.

El tipado gradual es un técnica efectiva para proveer al programador con una transición suave entre la flexibilidad de lenguajes dinámicamente tipados y la seguridad de lenguajes estáticamente tipados. Esto se logra permitiendo supocisiones optimistas durante el chequeo de tipos, que más tarde son monitoreadas y chequeadas en tiempo de ejecución. Por ejemplo, en un languaje gradualmente tipado, un programador puede empezar con un programa que es chequeado en una disciplina completamente dinámica, y a medida que el código se vuelve más estable, el programador puede agregar información de tipos con el fin de aprovechar las garantías entregadas por el chequeo de tipos estático. Nosotros establecemos la hipótesis de que el tipado gradual, y sus ventajas, pueden ser aplicadas en un context de tipos con sensibilidad. Tipos con sensibilidad gradual permitiría al programador moverse en un rango desde un programa con tipos simples a uno completamente anotado añadiendo información de sensibilidad.

En este trabajo, estudiamos la introducción del tipado gradual en la información de sensibilidad codificada en los tipos con sensibilidad. En particular, exploramos como la metodología *Abstracting Gradual Typing* (AGT) puede ser utilizada para lograr esta tarea. Primero, presentamos un lenguage con sensibilidad estáticamente tipado como un preámbulo a la introducción de tipado gradual. Discutimos las particularidades del lenguaje y establecemos dos propiedades en el contexto de sensibilidad: *type safety* y *soundness*. Luego, derivamos un lenguaje con sensibilidad gradual siguiendo paso a paso la metodología AGT y exploramos si satisface (1) las propiedades ya satisfechas por su contraparte estática, especialmente *soundness* con respecto a sensibilidad, y (2) una propiedad crucial en lenguajes gradualmente tipados, conocida como *gradual guarantee*.

Abstract

Sensitivity type systems are used to reason about the sensitivity of computations. This is of particular interest in the fields of privacy, specially differential privacy. One caveat of sensitivity types is that, being a typing discipline, they constrain programmers to statically deal with sometimes complex and often conservative restrictions imposed by types. This can quickly lead to a cumbersome developer experience.

Gradual typing is an effective approach to provide the programmer with a smooth transition between the flexibility of dynamically-typed languages and the safety of statically-typed ones. This is achieved by allowing optimistic assumptions during typechecking that are later monitored and checked during runtime. For instance, in a gradually-typed language, a programmer can start with a program that is checked in a full dynamic discipline, and as the code becomes stable the programmer can add type information in order to take advantage of the guarantees provided by static typechecking. We hypothesize that gradual typing, and its advantages, can be applied in a sensitivity types setting. Gradual sensitivity types would allow the programmer to range from a program with simple types to a fully-annotated one by adding sensitivity information.

In this work, we explore the introduction of gradual typing in the sensitivity information encoded in sensitivity types. In particular, we explore how the Abstracting Gradual Typing (AGT) methodology can be used to achieve this task. We first present a statically-typed sensitivity language as a preamble to the introduction of gradual typing. We discuss the particularities of the language and establish type safety and soundness in a sensitivity setting. We then derive a gradual sensitivity language by following step-by-step the AGT methodology and explore whether it satisfies (1) the properties already satisfied by its static counterpart, specially soundness with respect to sensitivity, and (2) a crucial property of gradually-typed languages, known as gradual guarantee.

Contents

1. Introduction **1**

2. Background **4**
- 2.1. Sensitivity Type Systems . . . 4
- 2.2. Gradual Typing . . . 6
- 2.3. Abstracting gradual typing . . . 8
- 2.4. Summary . . . 9

3. A Static Sensitivity Type System **11**
- 3.1. Syntax . . . 11
- 3.2. Static Semantics . . . 13
- 3.3. Dynamic Semantics . . . 18
- 3.4. Properties . . . 20
- 3.5. Summary . . . 25

4. A Gradual Sensitivity Type System **26**
- 4.1. Syntax and Meaning of Gradual Types . . . 26
- 4.2. Lifting the Type System . . . 30
  - 4.2.1. Lifting predicates . . . 30
  - 4.2.2. Lifting Type Functions . . . 31
- 4.3. Static Semantics . . . 33
- 4.4. Dynamic Semantics . . . 35
  - 4.4.1. Evidence for Consistent Subtyping . . . 35
  - 4.4.2. Intrinsic Terms . . . 39
  - 4.4.3. Reduction of Intrinsic Terms . . . 40
  - 4.4.4. Elaboration of Terms . . . 46
- 4.5. Properties . . . 49

5. Conclusions **54**

Bibliography **56**

A. Auxiliary definitions **60**
- A.1. A Static Sensitivity Type System . . . 60
- A.2. A Gradual Sensitivity Type System . . . 61

B. Properties of a Gradual Sensitivity Type System **62**

B.1. Preamble 62
B.2. Galois connections 72
B.3. Type Safety 75
B.4. Gradual Guarantee 84
B.4.1. Static Gradual Guarantee 84
B.4.2. Dynamic Gradual Guarantee 89
B.5. Soundness 95

Figures

3.1. Syntax of static sensitivity types 12
3.2. Type system of λ_s 14
3.3. Subtyping relation of static sensitivity types 15
3.4. Sensitivities scaling, join and meet 17
3.5. Dynamic semantics of λ_s 18
3.6. λ_s: Logical relations for metric preservation 22
4.1. Syntax of gradual sensitivities 27
4.2. Type system of λ_i 34
4.3. Algorithmic interior operator 36
4.4. Algorithmic consistent transitivity 38
4.5. Syntax and type system of $\lambda_{i\varepsilon}$ 41
4.6. Dynamic semantics of $\lambda_{i\varepsilon}$ 45
4.8. Inversion functions on evidences 47
4.9. Elaboration of $\lambda_{i\varepsilon}$ from λ_i 48
4.10. Logical relations for gradual sensitivity soundness 50
A.1. Sensitivity environment substitutions 60
A.2. Typing rules for closures and contexts 60
A.3. Gradual sensitivity environment substitutions 61
A.4. Typing rules for gradual closures and contexts 61
B.1. Precision of evaluation contexts 93

Chapter 1

Introduction

Privacy is a critical concern for software systems. Unfortunately, the use of traditional techniques, such as de-identification, is not enough to prevent privacy violations. The classical examples of this are the de-anonymization of user ratings of Netflix by using the Internet Movie Database (IMDb) as the source of background knowledge [1] and the re-identification of Governor William Weld's medical records [2]. Furthermore, Narayanan and Shmatikov have proposed a generic re-identification framework to target anonymized social networks graphs [3]. In this context, an emerging technique called differential privacy has received a lot of attention and it has become the standard approach for protecting privacy of individuals.

The goal of differential privacy is to gain knowledge from aggregated data without compromising any individual's privacy, i.e. revealing something particular about them. Informally, a computation is said to be differentially private if for two *similar* databases, the results of the computation are *close enough*. Two databases are considered to be similar if they differ in at most one individual's data. Furthermore, the result of the computations are close enough when they are indistinguishable to an external observer. Therefore, an attacker can not learn anything about the individual (whose data differ between databases) by analyzing the results. Differential privacy is a strong and formal statistical guarantee of privacy, that provides a mathematical definition of what it means for a computation over sensitive data to be private. Since its early formulation, many variants of differential privacy have been developed in the seek of interesting properties over the composition of differentially private algorithms [4–6].

Most differential privacy mechanisms achieve privacy by adding random noise to the outputs of computations. In order to determine how much an output must be perturbed, differential privacy uses a concept called *sensitivity*. Sensitivity is a measure of how much a computation can magnify the distance between two inputs. Quantifying how sensitive is a computation is a crucial part of differential privacy as it provides a lower bound on how much noise has to be introduced: if the added noise is too little, privacy may not be guaranteed; if it is too high, the result may no longer be useful.

Implementing differential privacy algorithms comes with a particular set of challenges: the *right amount* of noise have to be added in the *right places*. Worse yet, differential privacy is a probabilistic multi-run property, so developing test cases for differentially private algorithms is far from trivial. Hence, many efforts have been done in order to achieve automation or mechanization of differential privacy verification.

A particular set of approaches have used *program logics* [7–10] in order to prove correctness of differential privacy mechanisms and support advanced variants of differential privacy. However, these approaches are not suitable for automation. The other major approaches leverage typechecking and type systems in order to automate differential privacy verification. The first such approach, FUZZ [11], is based on linear type systems and is capable of measuring sensitivity and tracking privacy costs. FUZZ and its successor, DFUZZ [12], support automation and higher-order programming. Both languages use only one type system for sensitivity reasoning and privacy cost tracking, which has the advantage of being a simple approach. However, they are not able to support advanced variants of differential privacy.

Another approach based on type systems, HOARE^2 [13], uses relational refinement types to encode differential privacy. This improves on FUZZ-like systems being able to support more advanced variants of differential privacy, at the expense of limiting its automation support.

Latest developments, DUET [14] and JAZZ [15], have taken the approach of splitting their differential privacy language on two mutually embedded languages, each with its own type system: one for measuring sensitivity, namely a *sensitivity type system*; and another one for tracking privacy costs.

Inspired by FUZZ, DUET's sensitivity type system is based on linear types. Its multi-language design allows it to support more advanced variant of differential privacy at the expense of limited higher-order programming. The sensitivity language of JAZZ, SAX, make novel use of contextual linear types and a delayed type-and-effect discipline, which allows it to support advanced variants of differential privacy while having the same expressiveness for higher-order programming as FUZZ. Both sensitivity type systems are able to bound the sensitivity of functions and computations through typechecking.

One limitation of tracking sensitivity with types is that it imposes complexity on programmers, which can be prohibitive, especially in the early stages of software development. Just as in a sensitivity type system types are statically checked, there exist another discipline where types are dynamically checked. In fact, most programming languages today can be classified in two groups: statically-typed languages, such as Java, Scala or C♯; and dynamically typed languages, such as Python or Javascript. Both paradigms have their advantages and limitations. For example, statically-typed languages can prevent errors at runtime at the cost of conservatively rejecting programs that *may* go well. On the other hand, dynamically-typed languages are better suited for quick prototyping at the expense of possible runtime errors and extra runtime checks, that result in slower execution.

Many efforts have been done in combining the advantages of both paradigms [16–19]. One prominent approach is gradual typing [20], which provides the programmer with a smooth transition between dynamic and static checking within the same language by introducing a notion of imprecision on types. Hence, the programmer is able to choose which portions of the program are dynamically checked and which ones are statically checked.

Gradual typing has been studied in many settings such as subtyping [21, 22], references [23, 24], effects [25], ownership [26], information-flow typing [27, 28], refinement types [29], parametric polymorphism [30–34]. However, it has never been studied for a sensitivity type system.

In this work, we explore the gradualization of the sensitivity parts of a minimal statically-typed sensitivity language. We derive a gradual sensitivity language by following the Abstracting Gradual Typing (AGT) methodology [21], which provides a systematic approach for deriving gradual language using statically-typed languages as the starting point. This language allow a programmer to smoothly evolve a program with simple types by annotating it with sensitivity information, incrementally obtaining the advantages of sensitivity types. We prove that the gradual sensitivity language we present satisfies 3 key properties: **type safety**, i.e. a well-typed closed expression does not get stuck; **soundness**, which in the setting of sensitivity is defined as *metric preservation* [11] that captures the bound of how much two similar computations may change given a small input variation; and the **gradual guarantee** [35], which establishes that reducibility and typeability of programs is monotonic with respect to imprecision. The simultaneous satisfiability of the last two is of particular of interest for us since in related work [36, 37], when soundness is a multi-run hyperproperty [38] like metric preservation, conciliating it with the dynamic component of the gradual guarantee has proven to be challenging.

Contributions. To summarize, this work makes the following contributions:

- We present λ_s, a minimal statically-typed sensitivity language (§ 3). We explain the main mechanisms of a sensitivity language and work on several simplifications in order to ease the derivation of our gradual language.
- We derive a gradual sensitivity language, λ_i, by following the AGT methodology on λ_s (§ 4). We illustrate step-by-step how to apply the AGT methodology in a sensitivity types setting. This is the first application of both gradual typing and AGT to sensitivity types.
- We prove three key properties of the derived gradual language: type safety, soundness and the gradual guarantee (§ 4.5).

Chapter 2

Background

In this chapter we introduce the core ideas necessary to understand our work. The first section outlines the evolution of the mechanisms used in type systems in order to reason about sensitivity. We put special emphasis on the latest developed sensitivity language, Sax. Finally, in the last two sections, we present gradual typing and AGT as a methodology to derive gradual languages from statically-typed languages. The next chapters assume the reader's familiarity with these concepts.

2.1. Sensitivity Type Systems

Sensitivity captures how much a computation can magnify the distance between similar inputs. Formally, a function f is c-sensitive if and only if $|f(x) - f(y)| \leq c * |x - y|$, for all x, y. For example, $f(x) = x$ and $f(x) = -x$ are 1-sensitive functions, whereas $f(x) = 2x + 1$ is 2-sensitive. On the other hand, the function $f(x) = x * x$ would not be c-sensitive for any c, i.e., it is ∞-sensitive. In particular, for differential privacy, the sensitivity of a function provides a lower bound on how much the output must be perturbed in order to preserve privacy. Furthermore, reasoning about sensitivity is crucial for implementing and verifying differential privacy.

Developing correct differentially-private algorithms is particularly challenging as one has to introduce the *right amount* of noise in the *right parts* of the program. For the former, sensitivity plays a key role but quantifying it can rapidly become a non-trivial task when dealing with large programs. The latter is usually achieved by reasoning about privacy costs that flow through the program, which is as difficult as measuring sensitivity. Given the difficulty of verifying the correctness of differentially private algorithms, automation of this process has become an important area of research. Two major approaches have been explored: approaches using *program logics* [7–10] have mechanized the verification of differential privacy mechanism, but lack support for automation; and others have explored the verification of differential privacy by using type systems.

In general terms, a type system is a formalization of a modular static analysis performed to programs before they are executed. They are used to prevent type errors during runtime. Depending on how complex the types are, a type system can also encode more powerful guarantees. In particular, type systems can be used to verify differential privacy by encoding, among other things, sensitivity in its types.

The first approach that uses type systems to verify differential privacy is Fuzz [11], which uses linear types. The type system takes care of both sensitivity reasoning and privacy costs tracking. Focusing on the sensitivity parts of Fuzz, its type system makes use of linear types to bound the sensitivity of functions, which allows it to classify them as c-sensitive functions. Fuzz achieves this by using a typing judgment of the form $x :_c \tau_1 \vdash e : \tau_2$, meaning that e is a c-sensitive computation of type τ_2 with respect to the variable x of type τ_1. This notion generalizes to $x_1 :_{c_1} \tau_1, \ldots, x_n :_{c_n} \tau_n \vdash e : \tau$.

Both Fuzz and its successor DFuzz [12] use one type system that takes care of sensitivity measuring and privacy costs tracking. While this has the advantage of being a simple approach, it limits the type systems to not be able to support advanced variants of differential privacy, which are alternative formalizations that yield different composition properties [4–6].

HOARe[2] uses a relational refinement type system to encode differential privacy and, improving on previous type systems approaches, is capable of supporting advanced variants. However, it has limited support for automation.

Later, Duet [14] presents a novel approach to differential privacy type systems, splitting the language in two mutually embedded sub-languages, each with its own type system: one for reasoning about sensitivity and another one exclusively for privacy costs tracking. This approach allows Duet to support more advanced variants of differential privacy but at the expense of supporting less expressive higher-order programming than Fuzz.

Duet's successor, Jazz [15], is also built on a multi-language design which allows it to keep the reasoning about differential privacy through typechecking. In addition, its sensitivity language, Sax, makes use of contextual linear types, which supports fully-expressive higher-order programming. Sax presents a novel delayed sensitivity effects discipline, which increases the precision of the sensitivity analysis with respect to all previous work, specially for sum and product types. Product types, multiplicative and additive, encode pairs of resources. Their difference resides in how they are destructed: multiplicative pairs are destructed by pattern matching and both components can be used, and for additive pairs only one component can be used at the same time, by using projections. On the other hand, sum types encode alternative occurrences of resources and are introduced via inl and inr constructors. They are destructed via a case expression with one branch per constructor. In Sax, the expression $\mathsf{inl}(x + x)$ is 2-sensitive on x.

Sax separates the sensitivity tracking from the type environment, creating a separate *sensitivity environment*, Σ, that maps variables to sensitivities (positive real numbers). This results in a typing judgment of the form $\Gamma \vdash e : \tau; \Sigma$, where Γ maps variables to types. The delayed effect discipline of Jazz gets reflected in the syntax of function and sum types, $(x : \tau) \xrightarrow{\Sigma} \tau$ and $\tau \; {}^{\Sigma}\!\oplus^{\Sigma} \; \tau$, respectively. The annotated sensitivity environments are called latent effects and correspond to the sensitivity effect of the body of a function or the injected expression in a sum type. One important caveat of a function type $(x : \tau_1) \xrightarrow{\Sigma} \tau_2$ is that

x (the argument variable) may be present both in the latent effect and the result type (the result type may contain other latent effects). For instance, the expression $e = \mathsf{inl}^{\mathbb{R}}(x + x)$ is typechecked as $x : \mathbb{R} \vdash e : \mathbb{R}\ {}^{2x}\!\oplus^{\varnothing}\ \mathbb{R}; \varnothing$.

Soundness. For a sensitivity type system to be sound its types have to express an actual upper bound on the real sensitivity of a computation. For instance, in Sax the expression $x + x$ is 2-sensitive on x, so informally for any two similar inputs (values for x) at distance d, the result of the expression must not vary more than $2 * d$. In this example, it is easy to see that it is sound. However, for reasoning about more complex programs, soundness is defined as property called *metric preservation* [11]. This establishes that if a program has a predicted sensitivity, when closed by two similar inputs, the output will not vary more than predicted. We defer a more technical characterization of soundness to Chapter 3 with an actual type system in place.

2.2. Gradual Typing

Statically and dynamically typed languages have advantages and limitations of their own. For example, static checking provides static guarantees against runtime errors at the cost of conservatively rejecting programs that may go right. Furthermore, in some cases a static type discipline can result in a cumbersome developer experience since the programmer may have to fully annotate a program or deal with complex types. On the other hand, dynamic typing is better suited for quick prototyping and flexibility at the expense of extra checks during runtime (which may raise errors) and slower execution. Motivated by this duality, there is a lot of work done on trying to combine static and dynamic checking. One of the most important approaches is gradual typing [20] based on the notion of imprecision of types.

In a gradually typed language, static and dynamic checking are combined, providing the programmer the advantages of both paradigms through a smooth transition between both. This is achieved by introducing the unknown type, denoted $?$, which can be used to specify partially known types [20]. For example, the type $\text{Int} \rightarrow ?$ is the gradual type of functions whose domain is Int and the co-domain is statically unknown $?$.

A gradual typechecker optimistically treats the unknown as any type statically. Accounting for the optimistic judgments during typechecking, at runtime, a mechanism ensures a runtime type error is raised before performing an unsafe operation. For instance, $(\lambda x : ?.x + 1)\ \mathsf{false}$ is well-typed, but during runtime it will result in a runtime error before the addition is performed.

In summary, in a gradual language, if a portion of a program is annotated with static types, it gets verified at compile time, getting the benefits of static checking. Contrarily, if the portion is not annotated (by using the unknown type), it gets verified at runtime, getting the benefits of dynamic checking.

Gradual typing is about (im)precision. The key element for formalizing gradual typing is the notion of (programmer-controlled) precision on types [20, 24]. Type precision $\sqsubseteq$ is an ordering relation between gradual types where $G_1 \sqsubseteq G_2$ means that G_1 *represents* less static types than G_2. In order to introduce imprecision, most gradual languages define the unknown type $?$, which represents *any* possible type, i.e. $G \sqsubseteq ?$ for any G. The precision relation is defined also for constructs like function types where $G_1 \to G_2 \sqsubseteq G_1 \to ? \sqsubseteq ?$.

The notion of type precision can be naturally lifted to expressions. For example, the program $\lambda f : \mathbb{R} \to ?. f(1) + 2$ is less precise (or has less precise type information) than $\lambda f : \mathbb{R} \to \mathbb{R}. f(1) + 2$.

Cast as runtime checks. Gradual typing achieves its flexibility by treating imprecision optimistically, leveraging relaxed type predicates. For example, type *consistency*, noted $\sim$, is the relaxation of type equality [20, 21]. Recalling the previous example, $(\lambda x : ?. x + 1)\ \mathsf{false}$ is well-typed because $\mathrm{Bool} \sim ?$ and $? \sim \mathrm{Int}$ (but notice that $\mathrm{Bool} \nsim \mathrm{Int}$). However, it is important to notice that consistency is not a transitive relation, i.e. $\mathrm{Bool} \nsim \mathrm{Int}$. Therefore, a runtime mechanism is needed to prevent unsafe operations, e.g. adding a number with a boolean.

In a classic design of gradual languages, dynamic semantics are typically given by translating the source language to a cast calculus [20]. The translation insert casts at the boundaries between the static and dynamic portions of the programs, in order to raise an error before an optimistic assumption during typechecking is violated. For example, the expression $(\lambda x : ?. x + 1)\ \mathsf{false}$ would typically be translated to $(\lambda x : ?. \langle \mathrm{Int} \Leftarrow ? \rangle x + 1)\ \langle ? \Leftarrow \mathrm{Bool} \rangle \mathsf{false}$, where the cast $\langle \mathrm{Int} \Leftarrow ? \rangle$ ensures that the argument passed to the function is indeed of type Int, before proceeding with the addition.

Since type consistency is not a transitive relation, combination of casts is used as the mechanism to prevent unsafe operations. For instance, the expression presented before would reduce as:

$$\begin{aligned}
&(\lambda x : ?. \langle \mathrm{Int} \Leftarrow ? \rangle x + 1)\ \langle ? \Leftarrow \mathrm{Bool} \rangle \mathsf{false} \\
&\mapsto \langle \mathrm{Int} \Leftarrow ? \rangle (\langle ? \Leftarrow \mathrm{Bool} \rangle \mathsf{false}) + 1 \\
&\mapsto \mathsf{error}
\end{aligned}$$

When combining the two casts, $\langle \mathrm{Int} \Leftarrow ? \rangle$ and $\langle ? \Leftarrow \mathrm{Bool} \rangle$, an error is raised because types Int and Bool are not compatible.

Desirable properties of a gradual language. There are properties that every gradual language should aim to satisfy. In particular, Siek *et al.* [35] formalized a refined criteria of what it *means* for a language to be gradually typed. These are the following:

- **Type safety**: Well-typed expressions do not get stuck and they either reduce to values, diverge or halt with a runtime type error. Formally, if $\vdash e : G$ then either $t \Downarrow v$ and $\vdash v : G$, $e \Uparrow$, or $e \Downarrow \textbf{error}$, where $t \Uparrow$ denotes that e diverges.

- **Conservative extension of the static discipline**: A gradual type system is equivalent to its static counterpart on fully-annotated programs. Formally, $\vdash_s e : T$ if and only if $\vdash e : T$, where $\vdash_s$ denotes the typing judgment used in the static type system. Additionally, both reductions behave equivalently: $e \Downarrow_s v$ if and only if $e \Downarrow v$, where $\Downarrow_s$ denotes the big-step reduction relation for the static type system.

- **Embedding of the dynamic discipline**: Expressions from the corresponding dynamic language can be encoded into expressions of the gradual language, where all the binders and literals are annotated as $?$. Formally, let e be an expression from the dynamic language, then $\vdash \lceil e \rceil : ?$, where $\lceil \cdot \rceil$ is a function that annotates $?$ on every binder and literal. Additionally, the reduction of an expression in the dynamic language and its gradual counterpart behave equivalently: $e \Downarrow_d v$ if and only if $\lceil e \rceil \Downarrow \lceil v \rceil$.

- **Gradual guarantees**:

  - **Static gradual guarantee**: Expressions typeability is monotone with respect to imprecision, i.e. removing precision does not introduce new type errors. Formally, let e and e' be expressions such that $e \sqsubseteq e'$. If $\vdash e : G$ then $\vdash e' : G'$ and $G \sqsubseteq G'$, for some G'.

  - **Dynamic gradual guarantee**: Expressions reducibility is monotone with respect to imprecision, i.e. removing precision does not introduce new runtime errors. Formally, let e and e' be expressions such that $e \sqsubseteq e'$. If $e \Downarrow v$ then $e' \Downarrow v'$ and $v \sqsubseteq v'$, for some v'.

Classic design of gradual languages. The classical approach for designing gradual type systems, and used for most gradual languages, is an ad-hoc process. The runtime semantics are given by a translation to a cast calculus. However, as noted by Garcia *et al.* [21], there is no direct justification for how to choose or design a suitable cast calculus. An implication of this is that there is no guidance through the design process of the semantics of the gradual language, e.g. how should the unknown information be dealt with? Moreover, the correctness of the cast calculus with respect to the intentions of the source language is typically argued based on intuition.

2.3. Abstracting gradual typing

Abstracting Gradual Typing (AGT) is a systematic methodology for deriving gradual languages [21]. It uses abstract interpretation [39] at the type level to construct gradually typed languages using a statically typed language as the starting point. The application of the AGT methodology results in a gradual type system along with its runtime semantics, without the need for an intermediate cast calculus. In this section we present an overview of the AGT methodology. However, we defer a more detailed formalization to Chapter § 4, where we apply AGT step-by-step to language with a sensitivity type system.

AGT dictates the following steps for deriving a gradual language:

- **Deriving the static semantics**:
  1. Start from a statically-typed language and its type safety proof.
  2. Define the syntax and meaning of gradual types. This is done by defining a concretization function $\gamma : \text{GType} \rightarrow \mathcal{P}(\text{Type})$, that maps a gradual type to the set of static types it *represents*. For example, $?$ is mapped to the set of all possible static types, whereas $? \rightarrow \mathbb{R}$ is mapped to the set of all functions that return a real number.
  3. Existentially lift all type relations and type functions. This is done by exploiting the Galois connection obtained from the concretization function and its corresponding abstraction function, $\alpha : \mathcal{P}(\text{Type}) \rightharpoonup \text{GType}$, that takes any non-empty set of types and produces the most precise gradual type. The lifting is driven by plausibility: two gradual types are in a consistent relation if and only if *some* of the static types they represent are in the static relation.
  4. Re-write the static typing rules using the lifted relations and functions.
- **Deriving the dynamic semantics**:
  1. Define the syntax of *evidences* for consistent judgments, which is usually a pair of gradual types. Evidence represents *why* a consistent judgment holds. Evidence operations are defined by using a Galois connection as well (usually the same used when deriving the static semantics). Evidences are evolved during runtime in order to determine whether the use of a transitivity judgment holds.
  2. Derive the reduction rules mirroring the reasoning used in the type safety proof of the statically typed language. This exploits the correspondence between proof normalization and term reduction [40].

In summary, using AGT one can derive the static and dynamic semantics of a gradual language, starting from a statically-typed language along with the Galois connection(s). Although there is not a generalized proof, it is conjectured that this gradual language should satisfy, by construction, the refined criteria for gradual typing [35]. However, it is still necessary to prove this criteria formally for each derived language using AGT.

2.4. Summary

In this chapter we reviewed several key concepts needed to understand our work. Section 2.1 introduced basic notions of sensitivity and the importance of measuring it for differential privacy. It also presented the main characteristics of five differential privacy type systems, with particular focus in the most recent two: Duet and Sax. These are of particular interest as they introduce the first sensitivity type systems as part of their multi-language design.

Section 2.2 introduces and motivates the usefulness of gradual typing. It explains the key notions and mechanisms to give a develop a gradually-typed language. Lastly, several properties are presented as desired properties in a gradual typing setting.

Section 2.3 explains the workings and benefits of the Abstracting Gradual Typing (AGT) methodology. Although, details of the formalization are deferred to Chapter 4, where we apply AGT step-by-step to derive a gradually-typed sensitivity language.

In the next chapter, we present a sensitivity language along with its static and dynamic semantics, largely based on Sax, and we discuss the main characteristics of this sensitivity language mechanisms.

Chapter 3

A Static Sensitivity Type System

In this chapter we present a statically-typed sensitivity language, λ_s, that will act as the static counterpart of our gradual language. We present the details of the syntax and the static and dynamic semantics of λ_s, with particular focus on the sensitivity reasoning mechanisms and the important insights for the upcoming gradualization. The syntax and typing rules are largely based on a core subset of Sax [15]. λ_s works using a full-fledged type-and-effect discipline. We also work on several simplifications in order to ease the gradualization process in the next chapter.

Whereas Sax uses big-step runtime semantics, AGT is formalized on languages using small-step semantics. Therefore, in order to match the AGT pre-requisites for deriving our gradual language in the next chapter, we define a small-step runtime semantics for λ_s.

Finally, we establish two very important properties: type safety, i.e. every well-typed expression is either a value or it can take a step to another expression whose type is a subtype of the original one; and soundness, which in the context of sensitivity corresponds to a property called *metric preservation* [11]. Intuitively, this metric preservation ensures that predictions on sensitivity are sound with respect to the evolution of programs.

3.1. Syntax

We begin by defining the syntax of sensitivity types and expressions. In a sensitivity type system, typechecking is used not only to infer the type of an expression but also its sensitivity effect, i.e. how the expression may change. The sensitivity effect of an expression is variable-wise, namely, an expression can change depending on multiple variables. Therefore, it is useful to define effects as mappings between variables and sensitivities representing how a variable change may affect the value of an expression. This notions are formalized in Figure 3.1 which presents the syntax of expressions, sensitivity environments and types.

$$
\begin{array}{llll}
r \in \mathbb{R} & & & \\
b \in \mathbb{B} & & & \\
x \in \mathrm{Var} & & & \\
e \in \mathrm{Expr} & ::= & r \mid \; e + e \mid e \leq e & \text{real numbers} \\
 & \mid & b & \text{(derived) booleans} \\
 & \mid & x \mid \; \lambda(x : \tau).e \mid e \; e & \text{functions} \\
 & \mid & \mathsf{tt} & \text{unit} \\
 & \mid & \mathsf{inl}^{\tau} e \mid \; \mathsf{inr}^{\tau} e \mid \mathsf{case} \; e \; \mathsf{of} \{ x \Rightarrow e \} \; \{ x \Rightarrow e \} & \text{sums} \\
 & \mid & e :: \mathrm{T} & \text{ascriptions} \\
s \in \mathrm{Sens} & \triangleq & \mathbb{R}^{+} \cup \{\infty\} & \text{sensitivities} \\
\Sigma \in \mathrm{SEnv} & \triangleq & \mathrm{Var} \rightharpoonup \mathrm{Sens} ::= sx + \cdots + sx & \text{sensitivity environments} \\
\tau \in \mathrm{Type} & ::= & \mathbb{R} \mid \mathbb{B} \mid \mathsf{unit} \mid (x : \tau) \xrightarrow{\Sigma} \tau & \text{types} \\
 & \mid & \tau \; {}^{\Sigma}\!\oplus^{\Sigma} \; \tau & \\
\mathrm{T} \in \mathrm{Type}_{\Sigma} & ::= & \tau ; \Sigma & \text{type-and-effects}
\end{array}
$$

Figure 3.1: Syntax of static sensitivity types

Expressions. Syntax of expressions support operations on real numbers: real numbers literals r, additions $e + e$ and comparisons $e \leq e$. Expressions also support functions, so e can be either a variable x, a lambda $\lambda(x : \tau).e$ or an application $e \; e$. Additionally, an expression can encode sums, i.e. expressions can be the unit literal tt, a sum constructor $\mathsf{inl}^{\tau} e$ or inr^{τ}, or a sum destructor $\mathsf{case} \; e \; \mathsf{of} \{ x \Rightarrow e \} \; \{ y \Rightarrow e \}$. The left-hand side branch of a case expression is used to destruct an inl constructor. Analogously, the right-hand side branch is used to destruct inr constructors. For simplicity expressions can be derived booleans where $\mathsf{true} = \mathsf{inl}^{\mathsf{unit}}\mathsf{tt}$ and $\mathsf{false} = \mathsf{inr}^{\mathsf{unit}}\mathsf{tt}$. Finally, and differently from Sax, every expression can be ascribed to a type-and-effect instead of just a type. Ascriptions play a key role as hooks in AGT, so in our plan to gradualize the sensitivity parts of λ_s it is crucial to support ascriptions by both types and effects.

Sensitivities and Sensitivity environments. A sensitivity s is either a positive real number (including zero) or the infinity symbol ∞. A sensitivity environment Σ is a mapping between variables and sensitivities. Similar to Sax, we write a sensitivity environment as a first-order polynomial, e.g. $\Sigma = 1x + 3y$ is a sensitivity environment where $\Sigma(x) = 1$ and $\Sigma(y) = 3$. Although sensitivity environments are in essence partial functions, as only in-scope variables can be accessed, we treat them as total functions: whenever a variable that is not within the domain of a sensitivity environment is accessed, we return 0 as the sensitivity. Extending from the previous example, $\Sigma(z) = 0$ because $z \notin dom(\Sigma)$. Later, this simplification is going to allow us to define the typing rules in a more natural manner.

Types. A type can be a real number type $\mathbb{R}$, a boolean type $\mathbb{B}$, a unit type unit, a function type $(x : \tau) \xrightarrow{\Sigma} \tau$ or a sum type $\tau \; {}^{\Sigma}\!\oplus^{\Sigma} \; \tau$. The annotated Σ in a function type is called the latent sensitivity effect and corresponds to the effect of executing the body of the lambda, i.e. applying the function. In a function type $(x : \tau_1) \xrightarrow{\Sigma} \tau_2$, x may be present in the latent effect Σ or in the result type τ_2 (it can contain other latent effects). Because of this, in order to typecheck a function, the name of the argument variable, x, is annotated beside the argument type τ_1. Similar to function types, the annotated sensitivity environments Σ_1 and Σ_2 in a sum type $\tau_1 \,{}^{\Sigma_1}\!\oplus^{\Sigma_2} \tau_2$ are called the latent sensitivity effects, which correspond to the sensitivity effect of executing the injected expression by the inl or inr, respectively.

Type-and-effects. A type-and-effect T is a pair of a type τ and a sensitivity environment Σ. It will be used in the typing rules to report the type and the sensitivity effect of an expression.

3.2. Static Semantics

The typing rules for λ_s are presented in Figure 3.2. The expression typing judgment $\Gamma \vdash e : \mathrm{T}$ uses a type environment Γ to track variables in scope and their corresponding type-and-effects. Although the sensitivity effect of a variable x is always $1x$ initially, later when reducing a program, x may be bounded to a different sensitivity environment. Therefore, we need to track not only the type of a variable but its type-and-effect. Additionally, for brevity we just call them type environments, even though technically they are mappings between variables and type-and-effects.

Constants. Rules (Trlit) and (Tunit) are standard and report no effect since there is no variables being accessed. Notice that because of treating sensitivity environments as total functions we can report an empty sensitivity environment $\varnothing$. Otherwise, the reported effect should have the form $0x_1 + \cdots + 0x_n$ for $x_i \in dom(\Gamma)$, i.e. every free variable is explicitly mapped to a 0 sensitivity.

Additions. Rule (Tplus) computes the aggregated sensitivity effect by adding the effects of the both sub-expressions. Addition of sensitivity environments is defined as the polynomial sum, e.g. $(1x + 2y) + (4x) = 5x + 2y$.

Comparisons. Rule (Tleq) is similar to (Tplus) but the resulting effect is scaled by ∞ because the distance between different booleans is considered to be infinite. This is an implication of encoding booleans using sum types. Sensitivity environment scaling is defined in Figure 3.4.

Variables. Rule (Tvar) is standard as the type-and-effect of a variable is extracted from the type environment.

$$\Gamma \in \mathrm{TEnv} \;\triangleq\; \mathrm{Var} \rightharpoonup \mathrm{Type}_{\Sigma} ::= \cdot \mid \Gamma, x : \mathrm{T} \quad \text{type environments}$$

$\boxed{\Gamma \vdash e : \mathrm{T}}$ **Well-typed expressions**

$$\textsc{(TRlit)}\ \frac{}{\Gamma \vdash r : \mathbb{R}; \varnothing} \qquad \textsc{(TUnit)}\ \frac{}{\Gamma \vdash \mathsf{tt} : \mathsf{unit}; \varnothing} \qquad \textsc{(TPlus)}\ \frac{\Gamma \vdash e_1 : \mathbb{R}; \Sigma_1 \qquad \Gamma \vdash e_2 : \mathbb{R}; \Sigma_2}{\Gamma \vdash e_1 + e_2 : \mathbb{R}; \Sigma_1 + \Sigma_2}$$

$$\textsc{(TLeq)}\ \frac{\Gamma \vdash e_1 : \mathbb{R}; \Sigma_1 \qquad \Gamma \vdash e_2 : \mathbb{R}; \Sigma_2}{\Gamma \vdash e_1 \leq e_2 : \mathbb{B}; \infty(\Sigma_1 + \Sigma_2)} \qquad \textsc{(TVar)}\ \frac{\Gamma(x) = \mathrm{T}}{\Gamma \vdash x : \mathrm{T}} \qquad \textsc{(TLam)}\ \frac{\Gamma, x : \tau_1; x \vdash e : \tau_2; \Sigma}{\Gamma \vdash \lambda(x : \tau_1).e : (x : \tau_1) \xrightarrow{\Sigma} \tau_2; \varnothing}$$

$$\textsc{(TApp)}\ \frac{\Gamma \vdash e_1 : (x : \tau_1) \xrightarrow{\Sigma_2} \tau_2; \Sigma \qquad \Gamma \vdash e_2 : \tau_1'; \Sigma_1 \qquad \tau_1' <: \tau_1}{\Gamma \vdash e_1\ e_2 : [\Sigma_1/x]\tau_2; \Sigma + [\Sigma_1/x]\Sigma_2} \qquad \textsc{(TInl)}\ \frac{\Gamma \vdash e : \tau_1; \Sigma_1}{\Gamma \vdash \mathsf{inl}^{\tau_2} e : \tau_1 \ {}^{\Sigma_1}\!\oplus^{\varnothing}\ \tau_2; \varnothing}$$

$$\textsc{(TInr)}\ \frac{\Gamma \vdash e : \tau_2; \Sigma_2}{\Gamma \vdash \mathsf{inr}^{\tau_1} e : \tau_1 \ {}^{\varnothing}\!\oplus^{\Sigma_2}\ \tau_2; \varnothing}$$

$$\textsc{(TCase)}\ \frac{\Gamma \vdash e_1 : \tau_{11} \ {}^{\Sigma_{11}}\!\oplus^{\Sigma_{12}}\ \tau_{12}; \Sigma_1 \qquad \Gamma, x : \tau_{11}; x \vdash e_2 : \tau_2; \Sigma_2 \qquad \Gamma, y : \tau_{12}; y \vdash e_3 : \tau_3; \Sigma_3}{\begin{array}{c}\Gamma \vdash \mathsf{case}\ e_1\ \mathsf{of}\ \{\, x \Rightarrow e_2 \,\}\ \{\, y \Rightarrow e_3 \,\} : \\ ([\Sigma_1 + \Sigma_{11}/x]\tau_2 \curlyvee [\Sigma_1 + \Sigma_{12}/y]\tau_3)\ ;\ (\Sigma_1 \curlyvee [\Sigma_1 + \Sigma_{11}/x]\Sigma_2 \curlyvee [\Sigma_1 + \Sigma_{12}/y]\Sigma_3)\end{array}}$$

$$\textsc{(TAscr)}\ \frac{\Gamma \vdash e : \mathrm{T} \qquad \mathrm{T} <: \mathrm{T}'}{\Gamma \vdash e :: \mathrm{T}' : \mathrm{T}'}$$

Figure 3.2: Type system of λ_s

Functions. Rule (TLam) type checks the body under an extended type environment where the variable is bound to a type-and-effect composed by the type annotated in the lambda and a sensitivity environment that reports a single usage of a variable. The resulting type is annotated with a latent effect Σ, computed as the effect of the body. Lambdas are pure values, so the effect of constructing one is the empty sensitivity environment.

Function applications. Rule (TApp) typechecks applications and allows for subtyping in the argument. Subtyping is defined in Figure 3.3 and is discussed in more detail later when explaining ascriptions. Since x may be free in Σ_2 and τ_2 (e.g. in latent effects), x needs to be substituted by the argument effect, Σ_1, in the resulting type-and-effect. A sensitivity environment substitution, $[\Sigma_1/x]\Sigma$, replaces all occurrence of x in Σ by Σ_1. For example, $[2y + z/x](3x + y) = 3(2y + z) + y = 7y + 3z$. The same notion can be generalized to types. The resulting type is the substituted type of the arrow type body, $[\Sigma_1/x]\tau_2$, and the resulting

$\boxed{\Sigma <: \Sigma}$

$$\Sigma_1 <: \Sigma_2 \Leftrightarrow \forall x \in dom(\Sigma_1) \cup dom(\Sigma_2).\Sigma_1(x) \leq \Sigma_2(x)$$

$\boxed{\tau <: \tau}$

$$\frac{\tau \in \{\mathbb{R}, \mathsf{unit}\}}{\tau <: \tau} \qquad \frac{\tau_{21} <: \tau_{11} \qquad \Sigma_{12} <: \Sigma_{22} \qquad \tau_{12} <: \tau_{22}}{(x : \tau_{11}) \xrightarrow{\Sigma_{12}} \tau_{12} <: (x : \tau_{21}) \xrightarrow{\Sigma_{22}} \tau_{22}}$$

$$\frac{\tau_{11} <: \tau_{21} \qquad \Sigma_{11} <: \Sigma_{21} \qquad \tau_{12} <: \tau_{22} \qquad \Sigma_{12} <: \Sigma_{22}}{\tau_{11} \ {}^{\Sigma_{11}}\!\oplus^{\Sigma_{12}} \ \tau_{12} <: \tau_{21} \ {}^{\Sigma_{21}}\!\oplus^{\Sigma_{22}} \ \tau_{22}}$$

$\boxed{\mathrm{T} <: \mathrm{T}}$

$$\frac{\tau_1 <: \tau_2 \qquad \Sigma_1 <: \Sigma_2}{\tau_1; \Sigma_1 <: \tau_2; \Sigma_2}$$

Figure 3.3: Subtyping relation of static sensitivity types

effect is computed as the sum of the effect of the function, Σ, and the substituted latent effect of the function $[\Sigma_1/x]\Sigma_2$. For example, let $e = (\lambda(x : \mathbb{R}).x + y)y$ an open expression where y is free, the typing derivation follows as:

$$(\textsc{Tapp})\ \frac{y : \mathbb{R}; y \vdash \lambda(x : \mathbb{R}).x + y \ : \ (x : \mathbb{R}) \xrightarrow{x+y} \mathbb{R}; \varnothing \qquad y : \mathbb{R}; y \vdash y \ : \ \mathbb{R}; y \qquad \mathbb{R} <: \mathbb{R}}{y : \mathbb{R}; y \vdash (\lambda(x : \mathbb{R}).x + y)y \ : \ 2y}$$

If no substitution were performed on the resulting sensitivity effect, then x would be free. Therefore, we compute the sensitivity effect by substituting x by the effect of the argument $[y/x](x + y) = 2y$. The same applies for the resulting type as its type can contain references to x too, e.g. when returning a function instead of a real number. Sensitivity environment substitution is formally defined in Appendix A (Figure A.1).

Sum injections. Rule (Tinl) considers inl expressions to be pure. Therefore, sum injections have no effect. Instead, the effect of the expression being injected is annotated in the sum type as a latent effect. The constructor is annotated with a type τ_2 to avoid non-determinism in the typing derivations. The right-hand side latent effect is empty as it will never be accessed or used. Rule (Tinr) is defined analogously.

Sum eliminations. Rule (Tcase) typechecks the sub-expressions e_2 and e_3 under extended type environments where x and y are bound to a proper type-and-effect, respectively. The binding of x and y are similar to rule (Tlam). The resulting type is the join of the two branches types, where x and y have been substituted by the cost of using e_1: The sensitivity effect of e_1, Σ_1, plus the left latent effect Σ_{11} for x, and analogously, Σ_{12} for y. The same approach is followed for the computed effect, but if neither x is used in e_2 or y in e_3, the cost of computing e_1 is not going to be paid for. Thus, we also join Σ_1 to the resulting sensitivity effect. Therefore, in the worst case (where x or y are not used in their respective expressions) the reported effect is at least Σ_1. The join operator, $\curlyvee$, is defined in Figure 3.4: joining two sensitivities result in the maximum of the two; the join of two sensitivity environments is the natural lifting of joining the sensitivities variable-wise; join of types is defined inductively and is contravariant on the argument of function types, so the meet operator $\curlywedge$ is used. This operator is analogously defined in Figure 3.4, where the meet of two sensitivities is their minimum. Consider the open expression $\mathsf{case}\ z\ \mathsf{of}\,\{\, x_1 \Rightarrow 0 \,\}\ \{\, x_2 \Rightarrow x_2 + x_2 \,\}$ and a type environment $\Gamma = x : \mathbb{R}; x, z : \mathbb{R}\ {}^{\infty x}\!\oplus^{x}\ \mathbb{R}; z$, then the type derivation follows as:

$$(\textsc{Tcase})\ \dfrac{\begin{array}{c}\Gamma \vdash z : \mathbb{R}\ {}^{\infty x}\!\oplus^{x}\ \mathbb{R}; z \\ \Gamma, x_1 : \mathbb{R}; x_1 \vdash 0 : \mathbb{R}; \varnothing \qquad \Gamma, x_2 : \mathbb{R}; x_2 \vdash x_2 + x_2 : \mathbb{R}; 2x_2\end{array}}{\Gamma \vdash \mathsf{case}\ z\ \mathsf{of}\,\{\, x_1 \Rightarrow 0 \,\}\ \{\, x_2 \Rightarrow x_2 + x_2 \,\} : \mathbb{R}; (z \curlyvee [z + \infty x / x_1]\varnothing \curlyvee [z + x/x_2]2x_2)}$$

Then, the resulting effect is $2z + 2x$. Also notice that if we now define the expression as $\mathsf{case}\ z\ \mathsf{of}\,\{\, x_1 \Rightarrow 0 \,\}\ \{\, x_2 \Rightarrow 1 \,\}$, then the resulting effect is z, the cost of reducing z to a value.

Ascriptions. Rule $(\textsc{Tascr})$ allows expressions to be annotated with a greater type-and-effect under a subtyping relation. Subtyping is supported only in the sensitivity parts of type-and-effects. A sensitivity effect is subtype of another if all sensitivities are less or equal than the other for each variable. Although two sensitivity environments can have different domains, when checking for subtyping we assume that they share their domains and if a variable is not present on one of them, its sensitivity is 0. For instance, $\varnothing <: 2x$ is equivalent to $0x <: 2x$. Latent effects are co-variant.

For example, consider the open expression $e = \lambda(x : \mathbb{R}).x + 2y$ and a type environment $\Gamma = y : \mathbb{R}; y$. Then, $e :: (x : \mathbb{R}) \xrightarrow{2x+3y} \mathbb{R}; 2y$ is well-typed as its type derivation follows:

$$(\textsc{Tascr})\ \dfrac{\Gamma \vdash e : (x : \mathbb{R}) \xrightarrow{x+2y} \mathbb{R}; \varnothing \qquad (x : \mathbb{R}) \xrightarrow{x+2y} \mathbb{R}; \varnothing <: (x : \mathbb{R}) \xrightarrow{2x+3y} \mathbb{R}; 2y}{\Gamma \vdash e :: (x : \mathbb{R}) \xrightarrow{2x+3y} \mathbb{R}; 2y : (x : \mathbb{R}) \xrightarrow{2x+3y} \mathbb{R}; 2y}$$

The subtyping relation holds because $x + 2y <: 2x + 3y$ and $\varnothing <: 2y$.

$\boxed{s\Sigma}$ $\boxed{s \curlyvee s}$ $\boxed{s \curlywedge s}$

$$\begin{aligned} s\varnothing &= \varnothing \\ s(\Sigma + s'x) &= s\Sigma + (s * s')x \end{aligned}$$

$$s_1 \curlyvee s_2 = \max(s_1, s_2)$$

$$s_1 \curlywedge s_2 = \min(s_1, s_2)$$

$\boxed{\Sigma \curlyvee \Sigma}$

$$\begin{aligned} \varnothing \curlyvee \varnothing &= \varnothing \\ (\Sigma_1 + s_1 x) \curlyvee (\Sigma_2 + s_2 x) &= (\Sigma_1 \curlyvee \Sigma_2) + (s_1 \curlyvee s_2)x && \text{where } x \notin \mathit{dom}(\Sigma_1 \curlyvee \Sigma_2) \\ (\Sigma_1 + s_1 x) \curlyvee \Sigma_2 &= (\Sigma_1 \curlyvee \Sigma_2) + s_1 x && \text{where } x \notin \mathit{dom}(\Sigma_2) \\ \Sigma_1 \curlyvee (\Sigma_2 + s_2 x) &= (\Sigma_1 \curlyvee \Sigma_2) + s_2 x && \text{where } x \notin \mathit{dom}(\Sigma_1) \end{aligned}$$

$\boxed{\Sigma \curlywedge \Sigma}$

$$\begin{aligned} \varnothing \curlywedge \varnothing &= \varnothing \\ (\Sigma_1 + s_1 x) \curlywedge (\Sigma_2 + s_2 x) &= (\Sigma_1 \curlywedge \Sigma_2) + (s_1 \curlywedge s_2)x && \text{where } x \notin \mathit{dom}(\Sigma_1 \curlywedge \Sigma_2) \\ (\Sigma_1 + s_1 x) \curlywedge \Sigma_2 &= (\Sigma_1 \curlywedge \Sigma_2) + s_1 x && \text{where } x \notin \mathit{dom}(\Sigma_2) \\ \Sigma_1 \curlywedge (\Sigma_2 + s_2 x) &= (\Sigma_1 \curlywedge \Sigma_2) + s_2 x && \text{where } x \notin \mathit{dom}(\Sigma_1) \end{aligned}$$

$\boxed{\tau \curlyvee \tau}$

$$\begin{aligned} \mathbb{R} \curlyvee \mathbb{R} &= \mathbb{R} \\ \mathsf{unit} \curlyvee \mathsf{unit} &= \mathsf{unit} \\ (x : \tau_{11}) \xrightarrow{\Sigma_{12}} \tau_{12} \curlyvee (x : \tau_{21}) \xrightarrow{\Sigma_{22}} \tau_{22} &= (x : (\tau_{11} \curlywedge \tau_{21})) \xrightarrow{\Sigma_{12} \curlyvee \Sigma_{22}} (\tau_{12} \curlyvee \tau_{22}) \\ \tau_{11} \,{}^{\Sigma_{11}}\!\oplus^{\Sigma_{12}}\, \tau_{12} \curlyvee \tau_{21} \,{}^{\Sigma_{21}}\!\oplus^{\Sigma_{22}}\, \tau_{22} &= (\tau_{11} \curlyvee \tau_{21}) \,{}^{\Sigma_{11} \curlyvee \Sigma_{21}}\!\oplus^{\Sigma_{12} \curlyvee \Sigma_{22}}\, (\tau_{12} \curlyvee \tau_{22}) \end{aligned}$$

$\boxed{\tau \curlywedge \tau}$

$$\begin{aligned} \mathbb{R} \curlywedge \mathbb{R} &= \mathbb{R} \\ \mathsf{unit} \curlywedge \mathsf{unit} &= \mathsf{unit} \\ (x : \tau_{11}) \xrightarrow{\Sigma_{12}} \tau_{12} \curlywedge (x : \tau_{21}) \xrightarrow{\Sigma_{22}} \tau_{22} &= (x : (\tau_{11} \curlyvee \tau_{21})) \xrightarrow{\Sigma_{12} \curlywedge \Sigma_{22}} (\tau_{12} \curlywedge \tau_{22}) \\ \tau_{11} \,{}^{\Sigma_{11}}\!\oplus^{\Sigma_{12}}\, \tau_{12} \curlywedge \tau_{21} \,{}^{\Sigma_{21}}\!\oplus^{\Sigma_{22}}\, \tau_{22} &= (\tau_{11} \curlywedge \tau_{21}) \,{}^{\Sigma_{11} \curlywedge \Sigma_{21}}\!\oplus^{\Sigma_{12} \curlywedge \Sigma_{22}}\, (\tau_{12} \curlywedge \tau_{22}) \end{aligned}$$

Figure 3.4: Sensitivities scaling, join and meet

3.3. Dynamic Semantics

The dynamic semantics of λ_s are defined using evaluation contexts [41] and are presented in Figure 3.5. We also present extensions to the syntax of the language.

$$
\begin{array}{lcll}
v \in \mathrm{Val} & ::= & r \mid \mathsf{tt} \mid \langle \lambda(x:\tau).e, \gamma\rangle \mid \mathsf{inl}^{\tau} v \mid \mathsf{inr}^{\tau} v & \text{values} \\
\gamma \in \mathrm{VEnv} & \triangleq & \mathrm{var} \rightharpoonup \mathrm{val} ::= \{x \mapsto v, \ldots, x \mapsto v\} & \text{substitutions} \\
e \in \mathrm{Expr} & ::= & \cdots \mid \langle \lambda(x:\tau).e, \gamma\rangle \mid \mathsf{ctx}(\gamma, e) & \text{closures and contexts} \\
E & ::= & \square \mid E + e \mid v + E \mid E \leq e \mid v \leq E \mid E\ e \mid v\ E & \text{evaluation contexts} \\
 & \mid & \mathsf{inl}^{\tau} E \mid\ \mathsf{inr}^{\tau} E \mid\ \mathsf{case}\ E\ \mathsf{of}\,\{\, x \Rightarrow e\,\}\ \{\, x \Rightarrow e\,\} & \\
 & \mid & E :: \mathrm{T} &
\end{array}
$$

$\boxed{e \xrightarrow{\gamma} e}$ **Notions of reduction**

$$
\begin{array}{lrcl}
(\textsc{Tr-plus}) & r_1 + r_2 & \longrightarrow & r_3 \\
 & & \text{where} & r_3 = r_1 [\![+]\!] r_2 \\
(\textsc{Tr-leq}) & r_1 \leq r_2 & \longrightarrow & b \\
 & & \text{where} & b = r_1 [\![\leq]\!] r_2 \\
(\textsc{Tr-var}) & x & \xrightarrow{\gamma} & \gamma(x) \\
(\textsc{Tr-lam}) & \lambda(x:\tau).e & \xrightarrow{\gamma} & \langle \lambda(x:\tau).e, \gamma\rangle \\
(\textsc{Tr-app}) & \langle \lambda(x:g_1).e, \gamma'\rangle\ v & \xrightarrow{\gamma} & \mathsf{ctx}(\gamma'_{ext}, e) \\
 & & \text{where} & \gamma'_{ext} = \gamma'[x \mapsto v] \\
(\textsc{Tr-case-1}) & \mathsf{case}\ \mathsf{inl}^{g_{12}} v\ \mathsf{of}\,\{\, x \Rightarrow e_2\,\}\ \{\, y \Rightarrow e_3\,\} & \xrightarrow{\gamma} & \mathsf{ctx}(\gamma_{ext}, e_2) \\
 & & \text{where} & \gamma_{ext} = \gamma[x \mapsto v] \\
(\textsc{Tr-case-2}) & \mathsf{case}\ \mathsf{inr}^{g_{11}} v\ \mathsf{of}\,\{\, x \Rightarrow e_2\,\}\ \{\, y \Rightarrow e_3\,\} & \xrightarrow{\gamma} & \mathsf{ctx}(\gamma_{ext}, e_3) \\
 & & \text{where} & \gamma_{ext} = \gamma[x \mapsto v] \\
(\textsc{Tr-ctx}) & \mathsf{ctx}(\gamma', v) & \longrightarrow & v \\
(\textsc{Tr-ascr}) & v :: \mathrm{T} & \longrightarrow & v
\end{array}
$$

$\boxed{e \overset{\gamma}{\longmapsto} e}$ **Reduction**

$$
(\mathrm{TR}{\rightarrow})\ \frac{e_1 \xrightarrow{\gamma} e_2}{e_1 \overset{\gamma}{\longmapsto} e_2}
\qquad
(\mathrm{TR}E)\ \frac{e_1 \overset{\gamma}{\longmapsto} e_2}{E[e_1] \overset{\gamma}{\longmapsto} E[e_2]}
\qquad
(\mathrm{TR}\textsc{ctx})\ \frac{e_1 \overset{\gamma'}{\longmapsto} e_2}{\mathsf{ctx}(\gamma', e_1) \overset{\gamma}{\longmapsto} \mathsf{ctx}(\gamma', e_2)}
$$

Figure 3.5: Dynamic semantics of λ_s

Values. Values can be real, boolean or unit literals, a closure or a sum constructor whose injected expression is also a value.

Value environments. A substitution γ is a partial function that maps variables to the values they were instantiated with, i.e. in the application of a function or the elimination of an injected expression. An extended substitution $\gamma[x \mapsto v]$ stands for a new substitution in which the variable x is bound to the value v.

Runtime expressions. Similar to SAX, the runtime semantics of λ_s are defined using explicit substitution [42]. In order to support this in a small-step discipline, we introduce a new piece of syntax: `ctx`, called contexts. When using implicit substitution, denoted $[v/x]e$, every occurrence of x is immediately substituted by v and the actual substitution is deferred to the meta language. For instance, $[2/x](2x + 3 * x)$ immediately yields $2 * 2 + 3 * 2$, without the need of extra reduction steps. In contrast, a language with explicit substitution takes care of substituting each occurrence of the variable using reduction steps. In λ_s, when a variable is bounded to a value we create a new context with an extended substitution where the variable is mapped to the respecting value. For simplicity we define closures and contexts under the same meta-variable e but it is important to note that the (unreduced) body of a lambda will never contain a closure nor a context, since they are not part of the source language and they are constructed only during the reduction of an expression. Typing rules for closures and contexts are presented in the Appendix A.

Elimination rules. We present the elimination rules as notions of reduction, which capture how an expression and a substitution are transformed into a new pair of expression and substitution. Since a substitution is never explicitly extended upon reduction, the substitution on both sides of an elimination rule will always be the same. Thus, we simplify the notation writing the substitution (once) over the arrow in the elimination rule, i.e. $e \xrightarrow{\gamma} e$. Additionally, we do not annotate it whenever it is not relevant for the reduction rule. Rules (TR-PLUS) and (TR-LEQ) are eliminated for the value calculated by the meta language. Rule (TR-VAR) eliminates a variable by (explicitly) substituting it with the value provided in the substitution. Rule (TR-LAM) creates a closure that captures the current substitution. Rule (TR-APP) takes an application and produces a new context for reducing the body of the lambda e, where the substitution is the one captured by the closure extended with the value v provided for x. Rules (TR-CASE-LEFT) and (TR-CASE-RIGHT) work analogously to (TR-APP) except for the fact that they extend the current substitution. Rule (TR-CTX) takes care of dropping a substitution once the expression contained in the context has reached a value. Rule (TR-ASCR) eliminates ascriptions keeping only the underlying value.

Reduction rules. For reduction rules, we apply the same syntax simplifications as for elimination rules. Reduction rules, denoted $e \overset{\gamma}{\longmapsto} e$, capture the inductive mechanism used to reduce an expression using evaluation contexts E. The formulation of evaluation contexts is defined under a call-by-value discipline. Rule (TR$\rightarrow$) establish that if an expression e_1 can be eliminated to e_2, it also can be reduced to e_2. Rule (TRE) captures the inductive mechanism of reduction: an expression equivalent to $E[e_1]$ can reduce to another expression $E[e_2]$ if e_1 reduces to e_2. Rule (TRCTX) is a mechanism for handling explicit substitution: it operates similarly to (TRE) but allows the inner expression to reduce under a different substitution γ'.

Explicit substitution in action. The explicit substitution mechanism is mainly performed by a combination of the rules (TRctx) and (Tr-var). Suppose $e = (\lambda(x : \mathbb{R}).x + 1)\ 2$, then the reduction will perform as:

$$\begin{array}{lr}
(\lambda(x : \mathbb{R}).x + 1)\ 2 & \\
\mapsto \langle \lambda(x : \mathbb{R}).x + 1, \varnothing \rangle\ 2 & \text{by (Tr-lam), (TR}\rightarrow\text{) and (TR}E\text{)} \\
\mapsto \mathsf{ctx}(\{\, x \mapsto 2 \,\}, x + 1) & \text{by (Tr-app), (TR}\rightarrow\text{)} \\
\mapsto \mathsf{ctx}(\{\, x \mapsto 2 \,\}, 2 + 1) & \text{by (Tr-var), (TR}\rightarrow\text{) and (TRctx)} \\
\mapsto \mathsf{ctx}(\{\, x \mapsto 2 \,\}, 3) & \text{by (Tr-plus), (TR}\rightarrow\text{) and (TRctx)} \\
\mapsto 3 & \text{by (Tr-ctx), (TR}\rightarrow\text{)}
\end{array}$$

The reduction steps take care of substituting every instance of x, and when the reduction inside a context has reached a final value, the substitution is dropped along with the context, since a value has no free variables.

3.4. Properties

λ_s satisfy two very important properties: type safety, i.e. every well-typed closed expression is either a value or it can take a step to another expression whose type is a subtype of the original expressions' type; and soundness, which is going to be detailed later.

Proposition 1 (Type safety)**.** *Let $\cdot \vdash e : \mathrm{T}$. Then one of the following is true:*

1. *e is a value v.*
2. *$e \xrightarrow{\varnothing} e'$, where $\cdot \vdash e' : \mathrm{T}'$ and $\mathrm{T}' <: \mathrm{T}$.*

Proof. The proof is standard and follows from progress and preservation [43]. □

Soundness For a sensitivity type system to be sound, the predicted sensitivity of a program has to match or over-approximate the actual sensitivity. Intuitively, if a program is said to be s-sensitive, it cannot magnify the distance between two inputs by more than a factor of s. Formally, in a sensitivity types setting soundness is called *metric preservation* [11]. This property captures the maximum variation of an open expression if closed with two different (but related) substitutions, i.e. with respect to an input variation. To formalize the notion of input variation we (1) classify free variables as *directly sensitives* and *indirectly sensitives* depending on their sensitivity effect under a type environment Γ, and (2) define distance environments Δ that establish the variation of the free variables. Free variables are classified as following:

Definition 1 (Directly sensitive variables)**.** *A variable x is directly sensitive under a type environment Γ if and only if $x \in dom(\Gamma)$ and $\Gamma(x) = \tau; x$, for some τ.*

Definition 2 (Indirectly sensitive variables)**.** *A variable x is indirectly sensitive under a type environment Γ if and only if $x \in dom(\Gamma)$ and $\forall x_i \in dom(\Sigma).\ x_i \neq x$, where $\Gamma(x) = \tau; \Sigma$ for some τ.*

Suppose the expression $e = x + 2y$, then in order to typecheck this expression we must use a type environment, e.g. $\Gamma = x : \mathbb{R}; x, y : \mathbb{R}; 2x$. In the last example we are implicitly defining x a directly sensitive variable, as its sensitivity effect is the variable itself. Additionally, y is a indirectly sensitive variable since its effect depends entirely on other variables (x in this case). Typechecking the expression would yield $\Gamma \vdash e : \mathbb{R}; 5x$.

Distance environments. The syntax of distance environments is defined in Figure 3.6. Syntactically, a distance environment Δ is equivalent to a sensitivity environment. However, semantically they differ as a distance environment captures how much a directly sensitive variables may vary. Note that because of this, a distance environment will only contain a subset of all free-variables, as indirectly sensitive variables have no explicit variation, i.e. $y \notin dom(\Delta)$ if y is indirectly sensitive. This notion is formalized as follows:

Definition 3. *A distance environment Δ is well formed with respect to a type environment $tenv$, written $\Gamma \vdash \Delta$, if $\forall x \in dom(\Gamma)$ such that x is directly sensitive with respect to Γ, it follows that $x \in dom(\Delta)$.*

Logical relations. Similar to Sax, we establish metric preservation by using a logical relation. Logical relations are a proof method useful for enunciating properties that cannot be proven by traditional mechanisms like structural induction. They were first used for proving strong normalization [44], but to the date it has been applied in several scenarios, specially for hyperproperties such as *noninterference* [45], *relational parametricity* [46] or metric preservation in the case of Sax.

The logical relation for metric preservation is presented in Figure 3.6. We define three mutually recursive logical relations: for values, computations and substitutions. The logical relations for values ($\mathcal{V}_\Delta[\![\mathrm{T}]\!]$) and computations ($\mathcal{T}_\Delta[\![\mathrm{T}]\!]$) are indexed by a type-and-effect T. On the other hand, the logical relation for substitutions ($\mathcal{G}_\Delta[\![\Gamma]\!]$) is indexed by a type environment Γ. All three logical relations are also indexed by a distance environment Δ. Notation $(v_1, v_2) \in \mathcal{V}_\Delta[\![\mathrm{T}]\!]$ denotes that the value v_1 is related to value v_2 at type-and-effect T and at distance environment Δ. Likewise for computations and substitutions.

Related real numbers. Two numbers r_1 and r_2 are related if their distance, i.e. $|r_1 - r_2|$, is bounded by the maximum variation. The maximum variation is calculated as the dot product of the distance environment Δ and the sensitivity effect Σ. The dot product between two a distance environment and a sensitivity environment is defined inductively in Figure 3.6, but it is essentially the classic vectorial dot product. Notice that since $\Sigma(x) = 0$ when $x \notin dom(\Sigma)$, the definition of the dot product is correct and exhaustive.

Related unit literals. As expected, only the unit literal $\mathtt{tt}$ is related to itself.

$$\Delta \in \mathrm{DEnv} \triangleq \mathrm{Var} \rightharpoonup \mathrm{Sens} ::= sx + \cdots + sx \quad \text{distance environments}$$

$$\begin{aligned}
(r_1, r_2) \in \mathcal{V}_\Delta[\![\mathbb{R}; \Sigma]\!] &\iff |r_1 - r_2| \leq \Delta \cdot \Sigma \\
(v_1, v_2) \in \mathcal{V}_\Delta[\![\mathsf{unit}; \Sigma]\!] &\iff v_1 = \mathsf{tt} \wedge v_2 = \mathsf{tt} \\
(v_1, v_2) \in \mathcal{V}_\Delta[\![(x : \tau_1) \xrightarrow{\Sigma_2} \tau_2; \Sigma]\!] &\iff \forall \Gamma, \gamma_1, \gamma_2, v_1', v_2', \Sigma_1 : \\
&\quad (v_1', v_2') \in \mathcal{V}_\Delta[\![\tau_1; \Sigma_1]\!] \wedge (\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]. \\
&\quad (v_1\ v_1' \mid \gamma_1, v_2\ v_2' \mid \gamma_2) \in \mathcal{T}_\Delta[\![[\Sigma_1/x]\tau_2; \Sigma + [\Sigma_1/x]\Sigma_2]\!] \\
(v_1, v_2) \in \mathcal{V}_\Delta[\![\tau_1 \,{}^{\Sigma_1}\!\oplus^{\Sigma_2} \tau_2; \Sigma]\!] &\iff \Delta \cdot \Sigma < \infty \implies \forall \Gamma, \gamma_1, \gamma_2 : (\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]. \\
&\quad (useL(v_1) \mid \gamma_1, useL(v_2) \mid \gamma_2) \in \mathcal{T}_\Delta[\![\tau_1; \Sigma + \Sigma_1]\!] \vee \\
&\quad (useR(v_1) \mid \gamma_1, useR(v_2) \mid \gamma_2) \in \mathcal{T}_\Delta[\![\tau_2; \Sigma + \Sigma_2]\!] \\
(e_1 \mid \gamma_1, e_2 \mid \gamma_2) \in \mathcal{T}_\Delta[\![\tau_1; \Sigma_1]\!] &\iff (e_1 \xmapsto{\gamma_1}{}^{*} v_1 \wedge e_2 \xmapsto{\gamma_2}{}^{*} v_2) \implies (v_1, v_2) \in \mathcal{V}_\Delta[\![\tau_1; \Sigma_1]\!] \\
(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!] &\iff dom(\Gamma) = dom(\gamma_1) = dom(\gamma_2) \\
&\quad \wedge \forall x \in dom(\Gamma).\ (\gamma_1(x), \gamma_2(x)) \in \mathcal{V}_\Delta[\![\Gamma(x)]\!]
\end{aligned}$$

$$\begin{aligned}
useL(\mathsf{inl}^\tau v') &= v' \\
useR(\mathsf{inr}^\tau v') &= v'
\end{aligned}$$

$\boxed{\Delta \cdot \Sigma}$ **Dot product**

$$\begin{aligned}
\varnothing \cdot \Sigma &= 0 \\
(\Delta + sx) \cdot \Sigma &= (\Delta \cdot \Sigma) + s * \Sigma(x)
\end{aligned}$$

Figure 3.6: λ_s: Logical relations for metric preservation

Related closures. Two closures are related if, given related inputs, their computation under two related substitutions are related at the type-and-effect of the application. We quantify over related substitutions to reduce the applications of the closures as a inductive hypothesis mechanism, even though the closures and the arguments are already values.

Related sum injections. Different sum constructors (with the same type-and-effect) are considered to be related at distance ∞, i.e. when $\Delta \cdot \Sigma = \infty$. Otherwise, both sum constructors have to (1) be either inl or inr. And, (2) the computations of their injected expressions along with any two related substitutions have to be related on their respective type-and-effects. The sensitivity effect of the injected expressions is computed mirroring the typing rules. Similar to related closures, we also quantify over related substitutions to reduce the corresponding branch of the sum expression. Notice, however, that $useL(v_i)$ and $useR(v_i)$ are always values. We prefer to reason about their reduction in order to maintain consistency with the gradual version of these logical relations presented later, where $useL$ and $useR$ are not going to return values but expressions (not yet reduced).

Related computations. Since λ_s is a language with explicit substitution, instead of relating pairs of closed terms, we reason about pairs of related configurations, i.e. expressions enclosed by substitutions. Two configurations are related if they reduce to related values at the same type-and-effect. Although we could use the same approach as in related functions and sums, quantifying over related substitutions instead of taking them as arguments, we follow the classical approach where the quantification is done in the main property, i.e. metric preservation.

Related substitutions. Two substitutions are related with respect to a type environment Γ, if all their domains are the same. Additionally, the values held in the substitutions have to be related with respect to the expected type of that variable in Γ.

Combining the notion of sensitive variables and distance environments we can now announce the metric preservation property.

Theorem 2 (Metric preservation) If $\Gamma \vdash e : \mathrm{T}$, then $\forall \Delta, \gamma_1, \gamma_2$ such that $\Gamma \vdash \Delta$ and $(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$, $(e \mid \gamma_1, e \mid \gamma_2) \in \mathcal{T}_\Delta[\![\mathrm{T}]\!]$.

Proof. The proof is a particular case of metric preservation in a gradual setting, presented in the next chapter. □

Metric preservation in action. To illustrate how the logical relations work let us use the same example as before: $e = x + 2y$, $\Gamma = x : \mathbb{R}; x, y : \mathbb{R}; 2x$, $\mathrm{T} = \mathbb{R}; 5x$ and $\Gamma \vdash e : \mathrm{T}$. Let us pick an arbitrary distance environment $\Delta = 2x$. Notice how $dom(\Delta) = \{\, x \,\}$ as x is the only sensitive variable. Then, we have pick two related substitutions. In order to show how metric preservation captures the maximum variation of expressions, let us pick $\gamma_1 = \{\, x \mapsto 2, y \mapsto 4 \,\}$ and $\gamma_2 = \{\, x \mapsto 4, y \mapsto 8 \,\}$. Now, let us check that $(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$. We need to prove:

- $dom(\Gamma) = dom(\gamma_1) = dom(\gamma_2)$: Trivial. And,
- $\forall x \in dom(\Gamma).(\gamma_1(x), \gamma_2(x)) \in \mathcal{V}_\Delta[\![\Gamma(x)]\!]$, i.e.:
  - $(2, 4) \in \mathcal{V}_\Delta[\![\mathbb{R}; x]\!]$: As $\Delta = 2x$, then $\Delta \cdot (1x) = 2$. Finally, $|2 - 4| \leq 2$ holds. And,
  - $(4, 8) \in \mathcal{V}_\Delta[\![\mathbb{R}; 2x]\!]$: As $\Delta = 2x$, then $\Delta \cdot (2x) = 4$. Finally, $|4 - 8| \leq 4$ holds.

Now, given that $(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$, from theorem 2 we know that $(e \mid \gamma_1, e \mid \gamma_2) \in \mathcal{T}_\Delta[\![\mathbb{R}; 5x]\!]$. Let us see why:

- First, trivially $e \xmapsto{\gamma_1} 10$ and $e \xmapsto{\gamma_2} 20$. Then we have to prove that,
- $(10, 20) \in \mathcal{V}_\Delta[\![\mathbb{R}; 5x]\!]$: $\Delta \cdot (5x) = 10$. Finally, $|10 - 20| \leq 10$ holds.

In this example, the predicted sensitivity matches the actual sensitivity of the expression, $5x$. Nevertheless, given that a sensitivity type system is a conservative approximation, it may over-approximate the sensitivity of an expression. However, this approximation is still sound.

Consider a new example where $e = \mathsf{case}\ (\mathsf{inl}^{\mathbb{R}} 3z)\ \mathsf{of}\, \{\, x \Rightarrow x \,\}\ \{\, y \Rightarrow 6z \,\}$ and $\Gamma = z : \mathbb{R}; z$. The type-and-effect of e is $\Gamma \vdash e : \mathrm{T}$ with $\mathrm{T} = \mathbb{R}; \varnothing \curlyvee 3z \curlyvee 6z = 6z$. Notice that the typing rules over-approximate the sensitivity effect by joining the effect of the right branch, $6z$, whereas the expression will reduce through the left branch. Now, let us check soundness for e.

Let $\Delta = 3z$, $\gamma_1 = \{\, z \mapsto 5 \,\}$ and $\gamma_2 = \{\, z \mapsto 8 \,\}$. Again, let us see that $(\gamma_1, \gamma_2) \in \mathcal{G}_{\Delta}[\![\Gamma]\!]$:

- $dom(\Gamma) = dom(\gamma_1) = dom(\gamma_2)$: Trivial. And,
- $(5, 8) \in \mathcal{V}_{\Delta}[\![\mathbb{R}; z]\!]$: As $\Delta = 3z$, then $\Delta \cdot (1z) = 3$. Finally, $|5 - 8| \leq 3$ holds.

- Let us reduce e under the two substitutions: $e \xmapsto{\gamma_1} 15$ and $e \xmapsto{\gamma_2} 24$. Then, we have to prove that,
- $(15, 24) \in \mathcal{V}_{\Delta}[\![\mathbb{R}, 6z]\!]$: $\Delta \cdot (6z) = 18$. Finally, $|15 - 24| \leq 18$.

Notice that even though the substitutions are at the maximum distance allowed by Δ ($|5 - 8| = 3$ exactly), the over-approximation of the type system is still sound, i.e. satisfies metric preservation.

Of course, the previous examples show arbitrary values and they are not proofs of metric preservation. However, they are useful to exemplify the categorization of directly and indirectly sensitive variables as well as the semantic characterization of distance environments. In particular, they show how sensitivity types are a sound approximation of the actual sensitivity of the expression.

3.5. Summary

In this chapter we presented λ_s, a statically-typed sensitivity language, with the following characteristics (some of them inherited from Sax):

- **Support for real number operations, functions and sum types.** The syntax of λ_s is expressive enough to explore interesting results in Chapter § 4. We also support derived booleans, encoded by using sum types.
- **Type-and-effect discipline.** An expression e can be typechecked to a particular type and sensitivity effect, namely $\Gamma \vdash e : \mathrm{T}$, where T is a type-and-effect tuple $\tau;\Sigma$ and $\Gamma : \mathrm{Var} \rightharpoonup \mathrm{Type}_\Sigma$ is a mapping from variables to type-and-effects. Also, ascriptions by type-and-effect are supported in the syntax of the language.
- **Small-step dynamic semantics.** Since AGT is formalized in languages with small-step runtime semantics, we defined the reduction rules of λ_s using evaluation contexts.
- **Support for closures and higher-order programming.** Support for higher-order programming is an important feature in languages for verifying differential privacy. Since our work is meant to be a step towards gradual differential privacy languages, it is important that λ_s (and our gradual sensitivity language) supports higher-order programming as well.
- **Type safe and sound.** For the latter, we presented an interpretation of directly and indirectly sensitive free variables, based on their sensitivity effect.

In Chapter 4 we derive a gradually-typed sensitivity language by applying step-by-step the AGT methodology to λ_s. Finally, in addition to establishing important properties for a gradual language, we explore whether type safety and soundness are satisfied by the derived gradual language.

Chapter 4

A Gradual Sensitivity Type System

With a static sensitivity type system in place, we can start deriving a gradual language using the Abstracting Gradual Typing methodology (AGT) [21]. The first step is to define the syntax and give meaning to gradual types. The latter is achieved by defining a concretization function, $C : \mathrm{GType}_\Sigma \to \mathcal{P}(\mathrm{Type}_\Sigma)$, which connects gradual types with static types. Then, by finding a suitable abstraction function, $A : \mathcal{P}(\mathrm{Type}_\Sigma) \to \mathrm{GType}_\Sigma$, to establish a Galois connection [39], the static semantics of the static language can be lifted to the gradual setting. Finally, the dynamic semantics of the gradual language are derived by proof normalization of gradual typing derivations. In this chapter, we derive the gradual sensitivity language λ_i by applying the AGT methodology to λ_s, introduced in Chapter 3.

4.1. Syntax and Meaning of Gradual Types

We aim to gradualize only the sensitivity parts of the type system. Therefore, the "dynamic" end of the spectrum is simply typed. The cornerstone of a sensitivity language are sensitivities, so defining the syntax and meaning of gradual sensitivities is crucial.

Gradual sensitivities. Most gradual languages introduce the unknown type $?$ in the types syntax as a form of imprecision. Analogously, a natural way to introduce imprecision in a sensitivity language would be the unknown sensitivity $?$, that represents any sensitivity. However, as noted by Toro *et al.* [36], later when reasoning *why* a consistent subtyping judgment holds, the unknown sensitivity $?$ is not going to be expressive enough when trying to justify consistent transitivity judgments. Toro *et al.* solves this by using intervals to represent why consistent subtyping judgments hold. We follow the same approach by using intervals to abstract the meaning of gradual sensitivities. We defer the technical discussion of this decision to Subsection 4.4.1.

Syntax of gradual sensitivities is defined in Figure 4.1. A gradual sensitivity is defined as a valid interval of two sensitivities, i.e. positive real numbers or ∞ where the lower bound is less than or equal to the upper bound. This way, a gradual sensitivity captures the

$$
\begin{array}{rcll}
i \in \text{GSens} & ::= & [s, s] & \text{gradual sensitivities} \\
 & \mid & s & \text{(derived) fully-static sensitivities} \\
 & \mid & ? & \text{(derived) unknown sensitivities} \\
\Xi \in \text{GSEnv} & \triangleq & \text{Var} \rightharpoonup \text{GSens} ::= ix + \cdots + ix & \text{gradual sensitivity environments} \\
g \in \text{GType} & ::= & \mathbb{R} \mid \mathbb{B} \mid \mathsf{unit} \mid (x : g) \xrightarrow{\Xi} g & \text{gradual types} \\
 & \mid & g \ {}^{\Xi}\!\oplus^{\Xi}\ g & \\
G \in \text{GType}_{\Sigma} & ::= & g ; \Xi & \text{gradual type-and-effects}
\end{array}
$$

Figure 4.1: Syntax of gradual sensitivities

plausibility of a sensitivity being any number within the range. Furthermore, as a syntactic sugar we allow for the unknown sensitivity $?$ as a shorthand for the interval $[0, \infty]$, and a *fully precise* sensitivity s as a shorthand for the interval $[s, s]$. For example, a gradual sensitivity environment may be $\Xi = [0, 5]x + [0, \infty]y + [3, 3]z = [0, 5]x + ?y + 3z$.

Expressions e, values v, substitutions γ, type environments Γ and evaluation contexts E also have gradual types occurrences in them, but for readability their notation is not changed.

Similar to defining the syntax, a concretization function for gradual sensitivities naturally leads to the definitions of the concretization functions for sensitivity environments, types and type-and-effects.

Definition 4 (Sensitivity concretization)**.** *Let $C_i : GSens \to \mathcal{P}(Sens)$ be defined as follows:*

$$C_i([s_1, s_2]) = \{\, s \mid s_1 \leq s \leq s_2 \,\}$$

Definition 5 (Sensitivity environments concretization)**.** *Let $C_\Xi : GSEnv \to \mathcal{P}(SEnv)$ be defined as follows:*

$$C_\Xi(\varnothing) = \{\, \varnothing \,\} \qquad\qquad C_\Xi(\Xi + ix) = \{\, \Sigma + sx \mid \Sigma \in C_\Xi(\Xi) \wedge s \in C_i(i) \,\}$$

Definition 6 (Type concretization)**.** *Let $C_g : GType \to \mathcal{P}(Type)$ be defined as follows:*

$$C_g(\mathbb{R}) = \{\, \mathbb{R} \,\} \qquad\qquad C_g(\mathsf{unit}) = \{\, \mathsf{unit} \,\}$$

$$C_g((x : g_1) \xrightarrow{\Xi} g_2) = \{\, (x : \tau_1) \xrightarrow{\Sigma} \tau_2 \mid \tau_1 \in C_g(g_1) \wedge \tau_2 \in C_g(g_2) \wedge \Sigma \in C_\Xi(\Xi) \,\}$$

$$C_g(g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2}\ g_2) = \{\, \tau_1 \ {}^{\Sigma_1}\!\oplus^{\Sigma_2}\ \tau_2 \mid \tau_1 \in C_g(g_1) \wedge \Sigma_1 \in C_\Xi(\Xi_1) \wedge \Sigma_2 \in C_\Xi(\Xi_2) \wedge \tau_2 \in C_g(g_2) \,\}$$

Definition 7 (Type-and-effect concretization)**.** *Let $C : GType_\Sigma \to \mathcal{P}(Type_\Sigma)$ be defined as follows:*

$$C(g; \Xi) = \{\, \tau; \Sigma \mid \tau \in C_g(g) \wedge \Sigma \in C_\Xi(\Xi) \,\}$$

Once the concretization function is defined, the notion of *precision* can be directly derived from it [21]:

Definition 8 (Sensitivity precision). *$i_1 \sqsubseteq i_2$ if and only if $C_i(i_1) \subseteq C_i(i_2)$.*

Intuitively a gradual sensitivity i_1 is more precise than i_2 if the interval of i_1 is contained in the interval of i_2. For instance, $[2, 10] \sqsubseteq [1, 20]$ or $[10, 20] \sqsubseteq ?$ but $[0, 5] \not\sqsubseteq [2, 7]$.

Proposition 3. *The following definition of sensitivity precision is equivalent to definition 8. Let $i_1 = [s_{11}, s_{12}], i_2 = [s_{21}, s_{22}]$, then $i_1 \sqsubseteq i_2$ if and only if $s_{11} \geq s_{21}$ and $s_{12} \leq s_{22}$.*

Definition 9 (Sensitivity environment precision). *$\Xi_1 \sqsubseteq \Xi_2$ if and only if $C_\Xi(\Xi_1) \subseteq C_\Xi(\Xi_2)$.*

Proposition 4 (Sensitivity environment precision, inductively). *The following definition of sensitivity environment precision is equivalent to definition 9.*

$$\frac{}{\varnothing \sqsubseteq \varnothing} \qquad \frac{\Xi_1 \sqsubseteq \Xi_2 \qquad i_1 \sqsubseteq i_2}{\Xi_1 + i_1 x \sqsubseteq \Xi_2 + i_2 x}$$

Definition 10 (Type precision). *$g_2 \sqsubseteq g_2$ if and only if $C(g_2) \subseteq C(g_2)$.*

Proposition 5 (Type precision, inductively). *The following inductive definition of type precision is equivalent to definition 10.*

$$\frac{g \in \{\mathbb{R}, \mathsf{unit}, \mathbb{B}\}}{g \sqsubseteq g} \qquad \frac{g_1 \sqsubseteq g_1' \qquad \Xi \sqsubseteq \Xi' \qquad g_2 \sqsubseteq g_2'}{(x : g_1) \xrightarrow{\Xi} g_2 \sqsubseteq (x : g_1') \xrightarrow{\Xi'} g_2'}$$

$$\frac{g_1 \sqsubseteq g_1' \qquad \Xi_1 \sqsubseteq \Xi_1' \qquad \Xi_2 \sqsubseteq \Xi_2' \qquad g_2 \sqsubseteq g_2'}{g_1 \, {}^{\Xi_1}{\oplus}^{\Xi_2} \, g_2 \sqsubseteq g_1' \, {}^{\Xi_1'}{\oplus}^{\Xi_2'} \, g_2'}$$

Definition 11 (Precision). *$G_1 \sqsubseteq G_2$ if and only if $C(G_1) \subseteq C(G_2)$.*

Proposition 6 (Precision, inductively). *The following definition of precision is equivalent to definition 11.*

$$\frac{g_1 \sqsubseteq g_2 \qquad \Xi_1 \sqsubseteq \Xi_2}{g_1; \Xi_1 \sqsubseteq g_2; \Xi_2}$$

The next step is to define an abstraction function A, which produces a gradual type-and-effect from a non-empty set of static type-and-effects. The abstraction function must be both sound and optimal with respect to C: From a set of static type-and-effects, it produces the *most precise* gradual type-and-effects that over-approximates the given set. When A satisfy these two properties, $\langle C, A \rangle$ is said to form a Galois connection [39].

Similar to concretization, we first define an abstraction function for gradual sensitivities and all other definitions follow naturally. For deriving a gradual sensitivity from a set of static sensitivities we need to create a new interval in which the lower bound is less than or equal to any other sensitivity within the set. Similarly, the upper bound must be greater or equal to any sensitivity within the set. Therefore, the auxiliary functions min and max are used to find a suitable lower and upper bound, respectively. Abstraction functions for sensitivity environments, types and type-and-effects are defined inductively, using the previously defined abstractions. For convenience, we use the notation $\{\,\overline{\tau_i}\,\}$ to represent a set of type-and-effects labeled by an index i. We extend this notation to all type constructs and forms, including sensitivities and sensitivity environments.

Definition 12 (Sensitivity abstraction)**.** *Let $A_i : \mathcal{P}(Sens) \rightharpoonup GSens$ be defined as follows:*

$$A_i(S) = [\min(S), \max(S)]$$

Definition 13 (Sensitivity environments abstraction)**.** *Let $A_\Xi : \mathcal{P}(SEnv) \rightharpoonup GSEnv$ be defined as follows:*

$$A_\Xi(\{\,\varnothing\,\}) = \varnothing \qquad\qquad A_\Xi(\{\,\overline{\Sigma_i + s_i x}\,\}) = A_\Xi(\{\,\overline{\Sigma_i}\,\}) + A_i(\{\,\overline{s_i}\,\})x$$

Definition 14 (Type abstraction)**.** *Let $A_g : \mathcal{P}(Type) \rightharpoonup GType$ be defined as follows:*

$$A_g(\{\,\mathbb{R}\,\}) = \mathbb{R} \qquad\qquad A_g(\{\,\mathsf{tt}\,\}) = \mathsf{tt}$$

$$A_g(\{\,\overline{(x : \tau_{i1}) \xrightarrow{\Sigma_i} \tau_{i2}}\,\}) = (x : A_g(\{\,\overline{\tau_{i1}}\,\})) \xrightarrow{A_\Xi\{\,\overline{\Sigma_i}\,\}} A_g(\{\,\overline{\tau_{i2}}\,\})$$

$$A_g(\{\,\overline{\tau_{i1}\ {}^{\Sigma_{i1}}\!\oplus^{\Sigma_{i2}}\ \tau_{i2}}\,\}) = A_g(\{\,\overline{\tau_{i1}}\,\})\ {}^{A_\Xi(\{\,\overline{\Sigma_{i1}}\,\})}\!\oplus^{A_\Xi(\{\,\overline{\Sigma_{i2}}\,\})}\ A_g(\{\,\overline{\tau_{i2}}\,\})$$

Definition 15 (Abstraction)**.** *Let $A : \mathcal{P}(Type_\Sigma) \rightharpoonup GType_\Sigma$ be defined as follows:*

$$A(\{\,\overline{\tau_i; \Sigma_i}\,\}) = A_g(\{\,\overline{\tau_i}\,\}); A_\Xi(\{\,\overline{\Sigma_i}\,\})$$

The abstraction function A is both sound and optimal: From a set of static type-and-effects, it produces the *most precise* gradual type-and-effects that over-approximates the given set. The same applies for the other abstraction functions within their respecting domains. However, for brevity, we enounce only the Galois connections for gradual sensitivities and type-and-effects.

Proposition 7 (Galois connection for sensitivities)**.** *$\langle C_i, A_i\rangle$ is a Galois connection, i.e.:*

1. *(Soundness) for any non-empty set of static sensitivities $S = \{\,\overline{s}\,\}$, we have $S \subseteq C_i(A_i(S))$*
2. *(Optimality) for any gradual sensitivity i, we have $i \sqsubseteq A_i(C_i(i))$.*

Proposition 8 (Galois connection for type-and-effects)**.** *$\langle C, A\rangle$ is a Galois connection, i.e.:*

1. *(Soundness) for any non-empty set of static type-and-effects $S = \{\,\overline{\mathrm{T}}\,\}$, we have $S \subseteq C(A(S))$*

2. (Optimality) for any gradual type-and-effect G, we have $G \sqsubseteq A(C(G))$.

In summary, we defined the syntax and meaning of gradual sensitivities. For the latter, we defined concretization and abstraction functions for sensitivities, sensitivity environments, types and type-and-effects. Two important results come out of this: we now have a definition of precision for gradual constructs; and, as defined, concretization and abstraction functions form Galois connections. These results play a key role in the following sections.

4.2. Lifting the Type System

In order to derive the static semantics of λ_i, we lift type predicates (subtyping) and type functions (sensitivities and sensitivity environments addition, sensitivity environment scaling, sensitivity environment substitution and join). This lifting is obtained by leveraging the Galois connections through *existential* lifting. Intuitively, a lifted predicate between two gradual types holds if and only if there exists a concretization of the gradual types involved that satisfy the static predicate. For functions, the result is the abstraction of the collected results of the function applied to all elements of the concretization(s) of the gradual type(s).

4.2.1. Lifting predicates

Let $P \subseteq \mathcal{P}(\text{Sens}, \text{Sens})$ be some binary predicate on sensitivities. The lifted predicate $\widetilde{P} \subseteq \mathcal{P}(\text{GSens}, \text{GSens})$ is true for two gradual sensitivities i_1 and i_2 if there exist two sensitivities s_1 and s_2, represented by the two gradual sensitivities, that satisfy the static predicate P. The same notion is valid for other gradual constructs, such as sensitivity environments, types and type-and-effects. Although the equivalences are only conjectured, i.e. not formally proven, we also provide algorithmic definitions for all lifted predicates.

Definition 16 (Consistent subtyping for sensitivities)**.** *$i_1 \mathrel{\widetilde{\leq}} i_2$ if and only if $s_1 \leq s_2$, for some $s_1 \in C_i(i_1)$, $s_2 \in C_i(i_2)$*

Proposition 9. *The following definition of consistent sensitivity subtyping is equivalent to definition 16:*

$$\frac{s_1 \leq s_2}{[s_1, s_{12}] \mathrel{\widetilde{\leq}} [s_{21}, s_2]}$$

Definition 17 (Consistent subtyping for sensitivity environments)**.** *$\Xi_1 \mathrel{\widetilde{<:}} \Xi_2$ if and only if $\Sigma_1 <: \Sigma_2$ for some $\Sigma_1 \in C_\Xi(\Xi_1)$, $\Sigma_2 \in C_\Xi(\Xi_2)$.*

Proposition 10. *The following definition of consistent sensitivity environment subtyping is equivalent to definition 17: $\Xi_1 \mathrel{\widetilde{<:}} \Xi_2$ if and only if $\forall x.\ \Xi_1(x) \mathrel{\widetilde{\leq}} \Xi_2(x)$. Or inductively:*

$$\frac{}{\varnothing \mathrel{\widetilde{<:}} \varnothing} \qquad\qquad \frac{\Xi_1 \mathrel{\widetilde{<:}} \Xi_2 \qquad i_1 \mathrel{\widetilde{\leq}} i_2}{\Xi_1 + i_1 x \mathrel{\widetilde{<:}} \Xi_2 + i_2 x}$$

Definition 18 (Consistent subtyping). $g_1 \mathrel{\widetilde{<:}} g_2$ *if and only if* $\tau_1 <: \tau_2$ *for some* $\tau_1 \in C_g(g_1)$, $\tau_2 \in C_g(g_2)$.

Proposition 11. *The following definition is equivalent to definition 18:*

$$\frac{g \in \{\,\mathbb{R}, \mathsf{unit}\,\}}{g \mathrel{\widetilde{<:}} g} \qquad \frac{g_1' \mathrel{\widetilde{<:}} g_1 \qquad \Xi \mathrel{\widetilde{<:}} \Xi' \qquad g_2 \mathrel{\widetilde{<:}} g_2'}{(x : g_1) \xrightarrow{\Xi} g_2 \mathrel{\widetilde{<:}} (x : g_1') \xrightarrow{\Xi'} g_2'}$$

$$\frac{g_1 \mathrel{\widetilde{<:}} g_1' \qquad \Xi_1 \mathrel{\widetilde{<:}} \Xi_1' \qquad g_2 \mathrel{\widetilde{<:}} g_2' \qquad \Xi_2 \mathrel{\widetilde{<:}} \Xi_2'}{g_1 \; {}^{\Xi_1}\!\oplus^{\Xi_2} \; g_2 \mathrel{\widetilde{<:}} g_1' \; {}^{\Xi_1'}\!\oplus^{\Xi_2'} \; g_2'}$$

Definition 19 (Consistent subtyping for type-and-effects). $G_1 \mathrel{\widetilde{<:}} G_2$ *if and only if* $\mathrm{T}_1 <: \mathrm{T}_2$ *for some* $\mathrm{T}_1 \in C(G_1)$, $\mathrm{T}_2 \in C(G_2)$.

Proposition 12. *The following definition is equivalent to definition 19:*

$$\frac{g_1 \mathrel{\widetilde{<:}} g_2 \qquad \Xi_1 \mathrel{\widetilde{<:}} \Xi_2}{g_1; \Xi_1 \mathrel{\widetilde{<:}} g_2; \Xi_2}$$

4.2.2. Lifting Type Functions

In order to lift the type functions the abstraction function is required: the lifted function is defined as the abstraction of all possible results of the static function applied to the concretization of types. For example, consider a partial function $F : \mathrm{Type}_\Sigma \times \mathrm{Type}_\Sigma \rightharpoonup \mathrm{Type}_\Sigma$. The lifting of F, namely $\widetilde{F}$, is defined as $\widetilde{F}(G_1, G_2) = A(\widehat{F}(C(G_1), C(G_2)))$[1]. Note that the partiality of F leads to the notion of errors [21]. This notion extends to n-ary functions as well as to other abstractions and concretizations (not only type-and-effects).

In order to improve readability, we overload the name of the type functions to be lifted, so they can operate on static types as well on gradual types. For example, the addition of two gradual sensitivities is denoted $i_1 + i_2$ instead of $i_1 \mathbin{\widetilde{+}} i_2$. We also provide algorithmic definitions for lifted type functions.

Definition 20. $i_1 + i_2 = A_i(\{\, s_1 + s_2 \mid (s_1, s_2) \in C_i(i_1) \times C_i(i_2) \,\})$.

Proposition 13. *The following definition is equivalent to definition 20: Let* $i_1 = [s_1, s_2]$, $i_2 = [s_1', s_2']$, *then* $i_1 + i_2 = [s_1 + s_1', s_2 + s_2']$.

Definition 21. $\Xi_1 + \Xi_2 = A_i(\{\, \Sigma_1 + \Sigma_2 \mid (\Sigma_1, \Sigma_2) \in C_\Xi(\Xi_1) \times C_\Xi(\Xi_2) \,\})$.

Proposition 14. *The following definition is equivalent to definition 21:*

$$\varnothing + \varnothing = \varnothing$$
$$(\Xi_1 + i_1 x) + (\Xi_2 + i_2 x) = (\Xi_1 + \Xi_2) + (i_1 + i_2)x$$

Definition 22. $i_1 * i_2 = A_i(\{\, s_1 * s_2 \mid (s_1, s_2) \in C_i(i_1) \times C_i(i_2) \,\})$.

[1] $\widehat{F}$ is the notation for the collecting function of F

Proposition 15. *The following definition is equivalent to definition 22: Let* $i_1 = [s_1, s_2]$, $i_2 = [s'_1, s'_2]$, *then* $i_1 * i_2 = [s_1 * s'_1, s_2 * s'_2]$.

Definition 23. $i\Xi = A_i(\{\, s\Sigma \mid (s, \Sigma) \in C_i(i) \times C_\Xi(\Xi) \,\})$.

Proposition 16. *The following definition is equivalent to definition 23:*

$$i\varnothing = \varnothing$$
$$i(\Xi + i'x) = (i\Xi) + (i * i')x$$

Definition 24. $[\Xi_1/x]\Xi = A_i(\{\, [\Sigma_1/x]\Sigma \mid (\Sigma_1, \Sigma) \in C_\Xi(\Xi_1) \times C_\Xi(\Xi) \,\})$.

Proposition 17. *The following definition is equivalent to definition 24:*

$$[\Xi_1/x]\varnothing = \varnothing$$
$$[\Xi_1/x](\Xi + ix) = [\Xi_1/x]\Xi + i\Xi_1$$
$$[\Xi_1/x](\Xi + iy) = [\Xi_1/x]\Xi + iy$$

Definition 25. $[\Xi/x]g = A_i(\{\, [\Sigma/x]\tau \mid (\Sigma, \tau) \in C_\Xi(\Xi) \times C_g(g) \,\})$.

Proposition 18. *The following definition is equivalent to definition 25:*

$$[\Xi/x]\mathbb{R} = \mathbb{R}$$
$$[\Xi/x]\mathsf{unit} = \mathsf{unit}$$
$$[\Xi/x]((y : g_1) \xrightarrow{\Xi_2} g_2) = (y : [\Xi/x]g_1) \xrightarrow{[\Xi/x]\Xi_2} [\Xi/x]g_2$$
$$[\Xi/x](g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2}\ g_2) = [\Xi/x]g_1 \ {}^{[\Xi/x]\Xi_1}\!\oplus^{[\Xi/x]\Xi_2}\ [\Xi/x]g_2$$

Definition 26. $i_1 \curlyvee i_2 = A_i(\{\, s_1 \ \curlyvee s_2 \mid (s_1, s_2) \in C_i(i_1) \times C_i(i_2) \,\})$.

Proposition 19. *The following definition is equivalent to definition 26: Let* $i_1 = [s_{11}, s_{12}]$, $i_2 = [s_{21}, s_{22}]$, *then* $i_1 \curlyvee i_2 = [\max(s_{11}, s_{21}), \max(s_{12}, s_{22})]$.

Definition 27. $\Xi_1 \curlyvee \Xi_2 = A_\Xi(\{\, \Sigma_1 \curlyvee \Sigma_2 \mid (\Sigma_1, \Sigma_2) \in C_\Xi(\Xi_1) \times C_\Xi(\Xi_2) \,\})$.

Proposition 20. *The following definition is equivalent to definition 27:*

$$\varnothing \curlyvee \varnothing = \varnothing$$
$$(\Xi_1 + i_1x) \curlyvee (\Xi_2 + i_2x) = (\Xi_1 \curlyvee \Xi_2) + (i_1 \curlyvee i_2)x$$

Definition 28. $g_1 \curlyvee g_2 = A_\Xi(\{\, \tau_1 \curlyvee \tau_2 \mid (\tau_1, \tau_2) \in C_g(g_1) \times C_g(g_2) \,\})$.

Proposition 21. *The following definition is equivalent to definition 28:*

$$\mathbb{R} \curlyvee \mathbb{R} = \mathbb{R}$$
$$\mathsf{unit} \curlyvee \mathsf{unit} = \mathsf{unit}$$
$$(x : g_{11}) \xrightarrow{\Xi_{12}} g_{12} \curlyvee (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22} = (x : (g_{11} \curlywedge g_{21})) \xrightarrow{\Xi_{12} \curlyvee \Xi_{22}} (g_{12} \curlyvee g_{22})$$
$$g_{11} \ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\ g_{12} \curlyvee g_{21} \ {}^{\Xi_{21}}\!\oplus^{\Xi_{22}}\ g_{22} = (g_{11} \curlyvee g_{21}) \ {}^{\Xi_{11} \curlyvee \Xi_{21}}\!\oplus^{\Xi_{12} \curlyvee \Xi_{22}}\ (g_{12} \curlyvee g_{22})$$

Definition 29. $i_1 \curlywedge i_2 = A_i(\{\, s_1 \ \curlywedge s_2 \mid (s_1, s_2) \in C_i(i_1) \times C_i(i_2) \,\})$.

Proposition 22. *The following definition is equivalent to definition 29: Let* $i_1 = [s_{11}, s_{12}]$, $i_2 = [s_{21}, s_{22}]$*, then* $i_1 \curlywedge i_2 = [\min(s_{11}, s_{21}), \min(s_{12}, s_{22})]$.

Definition 30. $\Xi_1 \curlywedge \Xi_2 = A_\Xi(\{\, \Sigma_1 \curlywedge \Sigma_2 \mid (\Sigma_1, \Sigma_2) \in C_\Xi(\Xi_1) \times C_\Xi(\Xi_2) \,\})$.

Proposition 23. *The following definition is equivalent to definition 30:*

$$\varnothing \curlywedge \varnothing = \varnothing$$
$$(\Xi_1 + i_1 x) \curlywedge (\Xi_2 + i_2 x) = (\Xi_1 \curlywedge \Xi_2) + (i_1 \curlywedge i_2)x$$

Definition 31. $g_1 \curlywedge g_2 = A_\Xi(\{\, \tau_1 \curlywedge \tau_2 \mid (\tau_1, \tau_2) \in C_g(g_1) \times C_g(g_2) \,\})$.

Proposition 24. *The following definition is equivalent to definition 31:*

$$\mathbb{R} \curlywedge \mathbb{R} = \mathbb{R}$$
$$\mathsf{unit} \curlywedge \mathsf{unit} = \mathsf{unit}$$
$$(x : g_{11}) \xrightarrow{\Xi_{12}} g_{12} \curlywedge (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22} = (x : (g_{11} \curlyvee g_{21})) \xrightarrow{\Xi_{12} \curlywedge \Xi_{22}} (g_{12} \curlywedge g_{22})$$
$$g_{11} \;{}^{\Xi_{11}}{\oplus}^{\Xi_{12}}\; g_{12} \curlywedge g_{21} \;{}^{\Xi_{21}}{\oplus}^{\Xi_{22}}\; g_{22} = (g_{11} \curlywedge g_{21}) \;{}^{\Xi_{11} \curlywedge \Xi_{21}}{\oplus}^{\Xi_{12} \curlywedge \Xi_{22}}\; (g_{12} \curlywedge g_{22})$$

To summarize, we provided definitions for lifted type relations (subtyping) and functions. All definitions follow directly from the AGT methodology. Accounting for the pragmatics of the language, we provided algorithmic definitions for all lifted relations and functions. Whereas the predicate for consistent subtyping is denoted $\widetilde{<:}$, in contrast to static subtyping $<:$, all lifted functions are overloaded with their static counterpart in order to improve readability in upcoming sections. With all lifted type predicates and functions in place, we can now lift the type system.

4.3. Static Semantics

The type-system for λ_i is presented in Figure 4.2. The rules are obtained by replacing the static elements with their gradual counterpart just as the type functions and relations with their lifted counterparts. Note that by overloading the lifted type functions, only the types and the notation for subtyping are changed.

The type system of λ_i accepts optimistically judgments that may hold during runtime. For instance, consider the open expression $2x :: \mathbb{R}; ?x :: \mathbb{R}; x$ and a type environment $\Gamma = x : \mathbb{R}; x$. The type derivation of e follows as:

$$\dfrac{(\textsc{GAscr})\ \dfrac{\Gamma \vdash 2x \;:\; \mathbb{R}; 2x \qquad \mathbb{R}; 2x \mathrel{\widetilde{<:}} \mathbb{R}; ?x}{\Gamma \vdash 2x :: \mathbb{R}; ?x \;:\; \mathbb{R}; ?x} \qquad \mathbb{R}; ?x \mathrel{\widetilde{<:}} \mathbb{R}; x}{\Gamma \vdash 2x :: \mathbb{R}; ?x :: \mathbb{R}; x \;:\; \mathbb{R}; x}\ (\textsc{GAscr})$$

Notice how both subtyping judgments, $\mathbb{R}; ?x \mathrel{\widetilde{<:}} \mathbb{R}; x$ and $\mathbb{R}; 2x \mathrel{\widetilde{<:}} \mathbb{R}; ?x$, hold. This is

$\boxed{\Gamma \vdash e : G}$ **Well-typed gradual expressions**

(GRLIT)
$$\frac{}{\Gamma \vdash r : \mathbb{R}; \varnothing}$$

(GPLUS)
$$\frac{\Gamma \vdash e_1 : \mathbb{R}; \Xi_1 \qquad \Gamma \vdash e_2 : \mathbb{R}; \Xi_2}{\Gamma \vdash e_1 + e_2 : \mathbb{R}; \Xi_1 + \Xi_2}$$

(GLEQ)
$$\frac{\Gamma \vdash e_1 : \mathbb{R}; \Xi_1 \qquad \Gamma \vdash e_2 : \mathbb{R}; \Xi_2}{\Gamma \vdash e_1 \leq e_2 : \mathbb{B}; \infty(\Xi_1 + \Xi_2)}$$

(GVAR)
$$\frac{\Gamma(x) = G}{\Gamma \vdash x : G}$$

(GLAM)
$$\frac{\Gamma, x : g_1; x \vdash e : g_2; \Xi}{\Gamma \vdash \lambda(x : g_1).e : (x : g_1) \xrightarrow{\Xi} g_2; \varnothing}$$

(GAPP)
$$\frac{\Gamma \vdash e_1 : (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi \qquad \Gamma \vdash e_2 : g_1'; \Xi_1 \qquad g_1' \mathrel{\widetilde{<:}} g_1}{\Gamma \vdash e_1\ e_2 : [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2}$$

(GUNIT)
$$\frac{}{\Gamma \vdash \mathsf{tt} : \mathsf{unit}; \varnothing}$$

(GINL)
$$\frac{\Gamma \vdash e : g_1; \Xi_1}{\Gamma \vdash \mathsf{inl}^{g_2} e : g_1 \ {}^{\Xi_1}\!\oplus^{\varnothing}\ g_2; \varnothing}$$

(GINR)
$$\frac{\Gamma \vdash e : g_2; \Xi_2}{\Gamma \vdash \mathsf{inr}^{g_1} e : g_1 \ {}^{\varnothing}\!\oplus^{\Xi_2}\ g_2; \varnothing}$$

(GCASE)
$$\frac{\Gamma \vdash e_1 : g_{11} \ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\ g_{12}; \Xi_1 \qquad \Gamma, x : g_{11}; x \vdash e_2 : g_2; \Xi_2 \qquad \Gamma, y : g_{12}; y \vdash e_3 : g_3; \Xi_3}{\begin{array}{c}\Gamma \vdash \mathsf{case}\ e_1\ \mathsf{of}\ \{x \Rightarrow e_2\}\ \{y \Rightarrow e_3\} : \\ [\Xi_1 + \Xi_{11}/x]g_2 \curlyvee [\Xi_1 + \Xi_{12}/y]g_3; \Xi_1 \curlyvee [\Xi_1 + \Xi_{11}/x]\Xi_2 \curlyvee [\Xi_1 + \Xi_{12}/y]\Xi_3\end{array}}$$

(GASCR)
$$\frac{\Gamma \vdash e : G \qquad G \mathrel{\widetilde{<:}} G'}{\Gamma \vdash e :: G' : G'}$$

Figure 4.2: Type system of λ_i

because, given the removal of precision through the use of $?$, for the typechecker it is *plausible* that they might hold during runtime.

The type system of λ_i is equivalent to the type system of λ_s for fully-static expressions. We say a gradual type-and-effect is static if all the gradual sensitivities occurrences happen to be fully-static, i.e. of the form $[n, n]$. A fully-static expression is the natural lifting of the previous notion. Let $\vdash_s$ denote the typing judgment for λ_s.

Proposition 25 (Equivalence for fully-static expressions)**.** *Let e be a fully-static expression and G a static type ($G = \mathrm{T}$). $\cdot \vdash_s e : \mathrm{T}$ if and only if $\cdot \vdash e : \mathrm{T}$.*

Also, the static semantics of λ_i satisfy the static gradual guarantee, which states that typeability is monotone with respect to imprecision [35]. Precision on expressions, noted $e_1 \sqsubseteq e_2$, is the natural lifting of type precision to expressions.

Proposition 26 (Static gradual guarantee)**.** *Let e_1 and e_2 be two closed expressions such that $e_1 \sqsubseteq e_2$ and $\cdot \vdash e_1 : G_1$. Then, $\cdot \vdash e_2 : G_2$ and $G_1 \sqsubseteq G_2$.*

So far in this chapter we have defined the meaning of gradual sensitivities and lifted the typing rules, obtaining the static semantics of λ_i. These static semantics satisfy two important properties from the refined criteria for gradual typing [35]: they behave the same as their static counterpart on fully-annotated expressions; and typeability of gradual expressions is monotone with respect to imprecision. In the next section, we take the last step, which is to derive the dynamic semantics of λ_i.

4.4. Dynamic Semantics

For deriving the dynamic semantics of a gradual language, AGT introduces the concept of *evidence* that captures *why* a consistent judgment holds. Evidences are used to augment consistent judgments. Then, the dynamic semantics are derived by mimicking the type preservation argument of the static language, combining evidences through *consistent transitivity* to check whether an expression can reduce another step or should halt with a runtime error. We call these semantics $\lambda_{i\varepsilon}$.

Similar to the case of lifted predicates, we provide algorithmic definitions for evidence operators without providing a formal proof. However, we conjecture that the equivalence between the formal definitions and their algorithmic counterparts holds.

4.4.1. Evidence for Consistent Subtyping

Evidence represents the plausible static types that support some consistent judgment. Consider $\Xi_1 = ?x$ and $\Xi_2 = 10x$ and the valid consistent subtyping judgment $\Xi_1 \mathrel{\widetilde{<:}} \Xi_2$. Besides knowing *that* it holds, we also know *why* it holds: for any static evaluation of Ξ_1, the sensitivity of x must be less than or equal to 10. Therefore, we can refine the precise bounds on the set of static entities that support why consistent subtyping holds. This notion can be captured by a pair of two gradual types (type-and-effects in the current setting) $\varepsilon = \langle G, G \rangle$, where each type is at least as precise as the types involved in the consistent subtyping judgment. In the last example, the evidence for consistent subtyping would be initially computed as $\langle [0,10]x, [10,10]x \rangle \triangleright ?x \mathrel{\widetilde{<:}} 10x$ [2]. Definition 32 captures this formally [3].

Definition 32. $\varepsilon \triangleright G_1 \mathrel{\widetilde{<:}} G_2 \iff \varepsilon \sqsubseteq A^2(\{ \langle \mathrm{T}_1, \mathrm{T}_2 \rangle \mid \mathrm{T}_1 \in C(G_1) \wedge \mathrm{T}_2 \in C(G_2) \})$

Evidence is initially computed using a partial function called the *initial evidence operator* $\mathcal{I}_{<:}$. The initial evidence operator returns the most precise evidence that can be deduced from a given judgment. Although the initial evidence operator does not always coincide with the interior operator [21], in our language they are the same. Thus, we define the initial evidence operator exactly as the *interior operator* and we use these names interchangeably.

[2] Remember that $?$ and 10 are encoded as $[0,\infty]$ and $[10,10]$, respectively.

[3] $A^2(\{ \overline{\langle \mathrm{T}_{i1}, \mathrm{T}_{i2} \rangle} \}) = \langle A(\{ \overline{\mathrm{T}_{i1}} \}), A(\{ \overline{\mathrm{T}_{i2}} \}) \rangle$.

$\boxed{\mathcal{I}_{<:}(i,i)}$

$$\frac{s_1 \leq s_2 \curlywedge s_4 \qquad s_1 \curlyvee s_3 \leq s_4}{\mathcal{I}_{<:}([s_1,s_2],[s_3,s_4]) = \langle\, [s_1, s_2 \curlywedge s_4], [s_1 \curlyvee s_3, s_4] \,\rangle}$$

$\boxed{\mathcal{I}_{<:}(\Xi,\Xi)}$

$$\frac{}{\mathcal{I}_{<:}(\varnothing,\varnothing) = \langle\, \varnothing, \varnothing \,\rangle} \qquad \frac{\mathcal{I}_{<:}(\Xi_1,\Xi_2) = \langle\, \Xi_1', \Xi_2' \,\rangle \qquad \mathcal{I}_{<:}(i_1,i_2) = \langle\, i_1', i_2' \,\rangle}{\mathcal{I}_{<:}(\Xi_1 + i_1x, \Xi_2 + i_2x) = \langle\, \Xi_1' + i_1'x, \Xi_2' + i_2'x \,\rangle}$$

$\boxed{\mathcal{I}_{<:}(g,g)}$

$$\frac{}{\mathcal{I}_{<:}(\mathbb{R},\mathbb{R}) = \langle\, \mathbb{R}, \mathbb{R} \,\rangle} \qquad \frac{}{\mathcal{I}_{<:}(\mathbb{B},\mathbb{B}) = \langle\, \mathbb{B}, \mathbb{B} \,\rangle}$$

$$\frac{\mathcal{I}_{<:}(g_{21},g_{11}) = \langle\, g_{21}', g_{11}' \,\rangle \qquad \mathcal{I}_{<:}(\Xi_1,\Xi_2) = \langle\, \Xi_1', \Xi_2' \,\rangle \qquad \mathcal{I}_{<:}(g_{12},g_{22}) = \langle\, g_{12}', g_{22}' \,\rangle}{\mathcal{I}_{<:}((x : g_{11}) \xrightarrow{\Xi_1} g_{12}, (x : g_{21}) \xrightarrow{\Xi_2} g_{22}) = \langle\, (x : g_{11}') \xrightarrow{\Xi_1'} g_{12}', (x : g_{21}') \xrightarrow{\Xi_2'} g_{22}' \,\rangle}$$

$$\frac{\begin{array}{cc}\mathcal{I}_{<:}(g_{11},g_{21}) = \langle\, g_{11}', g_{21}' \,\rangle & \mathcal{I}_{<:}(g_{12},g_{22}) = \langle\, g_{12}', g_{22}' \,\rangle \\ \mathcal{I}_{<:}(\Xi_{11},\Xi_{21}) = \langle\, \Xi_{11}', \Xi_{21}' \,\rangle & \mathcal{I}_{<:}(\Xi_{21},\Xi_{22}) = \langle\, \Xi_{21}', \Xi_{22}' \,\rangle\end{array}}{\mathcal{I}_{<:}(g_{11} \;{}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\; g_{12}, g_{21} \;{}^{\Xi_{21}}\!\oplus^{\Xi_{22}}\; g_{22}) = \langle\, g_{11}' \;{}^{\Xi_{11}'}\!\oplus^{\Xi_{12}'}\; g_{12}', g_{21}' \;{}^{\Xi_{21}'}\!\oplus^{\Xi_{22}'}\; g_{22}' \,\rangle}$$

$\boxed{\mathcal{I}_{<:}(G,G)}$

$$\frac{\mathcal{I}_{<:}(g_1,g_2) = \langle\, g_1', g_2' \,\rangle \qquad \mathcal{I}_{<:}(\Xi_1,\Xi_2) = \langle\, \Xi_1', \Xi_2' \,\rangle}{\mathcal{I}_{<:}(g_1;\Xi_1, g_2;\Xi_2) = \langle\, g_1';\Xi_1', g_2';\Xi_2' \,\rangle}$$

Figure 4.3: Algorithmic interior operator

Definition 33 (Interior)**.**

$$\mathcal{I}_{<:}(G_1,G_2) = A^2(\{\, (\mathrm{T}_1,\mathrm{T}_2) \in C(G_1) \times C(G_2) \mid \mathrm{T}_1 <: \mathrm{T}_2 \,\})$$

In order to provide an algorithmic definition for the interior operator, we overload it to operate on smaller gradual constructs, specifically, gradual sensitivities, sensitivity environments and types (Figure 4.3).

Proposition 27. *The definition of the interior operator in Figure 4.3 is equivalent to definition 33.*

During runtime, evidences need to be *combined* in order to justify the use of transitivity judgments. For instance, although $[5,8]x \;\widetilde{<:}\; [2,6]x$ and $[2,6]x \;\widetilde{<:}\; [0,3]x$, the transitive judgement of both, $[5,8]x \;\widetilde{<:}\; [0,3]x$, does not hold, i.e. consistent subtyping is not a transitive relation. In the case where the combination of two evidences does not justify transitivity a runtime error is raised. Formally, the notion of combination of evidences is captured by the consistent transitivity operator, denoted $\circ^{<:}$ for a language with subtyping. For instance, suppose $\varepsilon_1 \triangleright 10x \;\widetilde{<:}\; ?x$ and $\varepsilon_2 \triangleright ?x \;\widetilde{<:}\; 5x$ [4]. Since $[10, 10]$ is not subtype of $[5, 5]$ $(10 \not\leq 5)$, then $\varepsilon_1 \circ^{<:} \varepsilon_2$ should be undefined and an error should be raised.

The consistent transitivity for a predicate P, denoted $\circ^{P}$, is defined by the abstract interpretation framework [21]. In particular, for subtyping it is defined as follows:

Definition 34 (Consistent subtyping transitivity)**.** *Suppose* $\varepsilon_{ab} \triangleright G_a \;\widetilde{<:}\; G_b$ *and* $\varepsilon_{bc} \triangleright G_b \;\widetilde{<:}\; G_c$*. Evidence for consistent subtyping transitivity is deduced as* $(\varepsilon_{ab} \circ^{<:} \varepsilon_{bc}) \triangleright G_a \;\widetilde{<:}\; G_c$*, where:*

$$\langle G_1, G_{12} \rangle \circ^{<:} \langle G_{21}, G_3 \rangle = A^2(\{\mathrm{T}_1, \mathrm{T}_3 \in C(G_1) \times C(G_3) \mid \exists \mathrm{T}_2 \in C(G_{12}) \cap C(G_{21}), \mathrm{T}_1 <: \mathrm{T}_2 \wedge \mathrm{T}_2 <: \mathrm{T}_3\})$$

Recalling Section 4.1, when we discussed why gradual sensitivities should be interpreted as intervals instead of a simple unknown sensitivity $?$, let us see how evidence evolves when intervals are not available. Suppose $\varepsilon_1 = \langle 3x, 5x \rangle$, $\varepsilon_2 = \langle 5x, ?x \rangle$ and $\varepsilon_3 = \langle 4x, 4x \rangle$. Let us compute $\varepsilon = (\varepsilon_1 \circ^{<:} \varepsilon_2) \circ^{<:} \varepsilon_3$ without using intervals:

$$\begin{aligned}
\varepsilon &= (\varepsilon_1 \circ^{<:} \varepsilon_2) \circ^{<:} \varepsilon_3 \\
&= (\langle 3x, 5x \rangle \circ^{<:} \langle 5x, ?x \rangle) \circ^{<:} \varepsilon_3 \\
&= (\langle 3x, ?x \rangle) \circ^{<:} \varepsilon_3 \\
&= \langle 3x, ?x \rangle \circ^{<:} \langle 4x, 4x \rangle \\
&= \langle 3x, 4x \rangle
\end{aligned}$$

Now, let us see how ε is computed in a setting with intervals:

$$\begin{aligned}
\varepsilon &= (\varepsilon_1 \circ^{<:} \varepsilon_2) \circ^{<:} \varepsilon_3 \\
&= (\langle 3x, 5x \rangle \circ^{<:} \langle 5x, ?x \rangle) \circ^{<:} \varepsilon_3 \\
&= (\langle 3x, [5, \infty]x \rangle) \circ^{<:} \varepsilon_3 \\
&= \langle 3x, [5, \infty]x \rangle \circ^{<:} \langle 4x, 4x \rangle && (C_{\Xi}([5, \infty]x) \cap C_{\Xi}(4x) = \varnothing) \\
&= \mathtt{undefined}
\end{aligned}$$

As expected, the combination using intervals yields an undefined result. However the setting without intervals does not fail because in the combination of ε_1 and ε_2 information is lost. From the beginning we can notice that the right-hand side sensitivity of ε_2 will never be less than 5. Nevertheless, the result of the combination, $\langle 3x, ?x \rangle$, has no way to encode such information, so later when combined with ε_3 the operation can not be refuted.

Following the same approach as for the interior operator, in order to provide an algorithmic definition, we overload the consistent transitivity operator to work on smaller gradual constructs (Figure 4.4).

[4] Notice that we are omitting types, both in judgments and in evidences, in order to improve readability as

$\boxed{\langle i, i \rangle \circ^{<:} \langle i, i \rangle}$

$$\frac{s_{11} \leq (s_{12} \curlywedge s_{14} \curlywedge s_{22}) \qquad (s_{13} \curlyvee s_{21} \curlyvee s_{23}) \leq s_{24}}{\langle [s_{11}, s_{12}], [s_{13}, s_{14}] \rangle \circ^{<:} \langle [s_{21}, s_{22}], [s_{23}, s_{24}] \rangle = \langle [s_{11}, s_{12} \curlywedge s_{14} \curlywedge s_{22}], [s_{13} \curlyvee s_{21} \curlyvee s_{23}, s_{24}] \rangle}$$

$\boxed{\langle \Xi, \Xi \rangle \circ^{<:} \langle \Xi, \Xi \rangle}$

$$\frac{}{\langle \varnothing, \varnothing \rangle \circ^{<:} \langle \varnothing, \varnothing \rangle = \langle \varnothing, \varnothing \rangle}$$

$$\frac{\langle \Xi_1, \Xi_2 \rangle \circ^{<:} \langle \Xi_3, \Xi_4 \rangle = \langle \Xi'_1, \Xi'_4 \rangle \qquad \langle i_1, i_2 \rangle \circ^{<:} \langle i_3, i_4 \rangle = \langle i'_1, i'_4 \rangle}{\langle \Xi_1 + i_1 x, \Xi_2 + i_2 x \rangle \circ^{<:} \langle \Xi_3 + i_3 x, \Xi_4 + i_4 x \rangle = \langle \Xi'_1 + i'_1 x, \Xi'_4 + i'_4 x, \rangle}$$

$\boxed{\langle g, g \rangle \circ^{<:} \langle g, g \rangle}$

$$\frac{g \in \{ \mathbb{R}, \mathsf{unit} \}}{\langle g, g \rangle \circ^{<:} \langle g, g \rangle = \langle g, g \rangle}$$

$$\frac{\begin{array}{c}\langle g_{41}, g_{31} \rangle \circ^{<:} \langle g_{21}, g_{11} \rangle = \langle g'_{41}, g'_{11} \rangle \\ \langle \Xi_{12}, \Xi_{22} \rangle \circ^{<:} \langle \Xi_{32}, \Xi_{42} \rangle = \langle \Xi'_{12}, \Xi'_{42} \rangle \qquad \langle g_{12}, g_{22} \rangle \circ^{<:} \langle g_{32}, g_{42} \rangle = \langle g'_{12}, g'_{42} \rangle\end{array}}{\begin{array}{c}\langle (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}, (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22} \rangle \circ^{<:} \langle (x : g_{31}) \xrightarrow{\Xi_{32}} g_{32}, (x : g_{41}) \xrightarrow{\Xi_{42}} g_{42} \rangle = \\ \langle (x : g'_{11}) \xrightarrow{\Xi'_{12}} g'_{12}, (x : g'_{41}) \xrightarrow{\Xi'_{42}} g'_{42} \rangle\end{array}}$$

$$\frac{\begin{array}{c}\langle g_{11}, g_{21} \rangle \circ^{<:} \langle g_{31}, g_{41} \rangle = \langle g'_{11}, g'_{41} \rangle \qquad \langle \Xi_{11}, \Xi_{21} \rangle \circ^{<:} \langle \Xi_{31}, \Xi_{41} \rangle = \langle \Xi'_{11}, \Xi'_{41} \rangle \\ \langle \Xi_{12}, \Xi_{22} \rangle \circ^{<:} \langle \Xi_{32}, \Xi_{42} \rangle = \langle \Xi'_{12}, \Xi'_{42} \rangle \qquad \langle g_{12}, g_{22} \rangle \circ^{<:} \langle g_{32}, g_{42} \rangle = \langle g'_{12}, g'_{42} \rangle\end{array}}{\begin{array}{c}\langle g_{11} \; {}^{\Xi_{11}}\!\oplus^{\Xi_{12}} \; g_{12}, g_{21} \; {}^{\Xi_{21}}\!\oplus^{\Xi_{22}} \; g_{22} \rangle \circ^{<:} \langle g_{31} \; {}^{\Xi_{31}}\!\oplus^{\Xi_{32}} \; g_{32}, g_{41} \; {}^{\Xi_{41}}\!\oplus^{\Xi_{42}} \; g_{42} \rangle = \\ \langle g'_{11} \; {}^{\Xi'_{11}}\!\oplus^{\Xi'_{12}} \; g'_{12}, g'_{41} \; {}^{\Xi'_{41}}\!\oplus^{\Xi'_{42}} \; g'_{42} \rangle\end{array}}$$

$\boxed{\langle G, G \rangle \circ^{<:} \langle G, G \rangle}$

$$\frac{\langle g_1, g_2 \rangle \circ^{<:} \langle g_3, g_4 \rangle = \langle g'_1, g'_4 \rangle \qquad \langle \Xi_1, \Xi_2 \rangle \circ^{<:} \langle \Xi_3, \Xi_4 \rangle = \langle \Xi'_1, \Xi'_4 \rangle}{\langle g_1; \Xi_1, g_2; \Xi_2 \rangle \circ^{<:} \langle g_3; \Xi_3, g_4; \Xi_4 \rangle = \langle g'_1; \Xi'_1, g'_4; \Xi'_4 \rangle}$$

Figure 4.4: Algorithmic consistent transitivity

Proposition 28. *The definition of consistent transitivity in Figure 4.4 is equivalent to definition 34.*

Equipped with a formal definition of evidences and operations for initially inferring (interior operator) and combining them (consistent transitivity operator), we can now augment the syntax of our gradual language with runtime information of why consistent judgments hold and whether they can be combined. Recalling the characteristics of gradual languages, evidences serve the purpose of accounting for the optimistic judgments during typechecking. Moving forward, we derive the runtime semantics by elaborating λ_i to a language with a richer syntax.

4.4.2. Intrinsic Terms

In order to avoid writing reduction rules on actual (bi-dimensional) derivation trees, Garcia *et al* [21] leverage the use of *intrinsic terms*, a flat representation of terms that are isomorphic to type derivations [47]. More specifically, the typing judgment $\Gamma \vdash e : G$ is now represented by an intrinsic term $t^G \in \mathbb{T}[G]$, where all the information contained in Γ is implicitly present in the syntax of t^G.

In addition to the use of intrinsic terms, and similar to Toro *et al* [48], we heavily rely on a type-directed translation that inserts explicit ascriptions to every inner-term ensuring that all top-level constructor types match.

AGT requires that when a term reduces to a new one it has to preserve its type (type-and-effect), but the static type system λ_s reduces under subtyping, i.e. the type of the resulting expression is a subtype of the original one. Preserving the types without ascriptions is impossible, since values have no sensitivity effect. For instance, by rule (Tr-ascr), the expression $2 :: \mathbb{R}; 2x$ reduces to 2, whose type-and-effect is $\mathbb{R}; \varnothing$. Therefore, we re-define the syntax of values (and terms) to work under the premise that every value and inner-term is ascribed.

The syntax of intrinsic terms is presented in Figure 4.5. We avoid writing the explicit type exponent whenever is not needed and can be inferred from the context, i.e. $t \in \mathbb{T}[G]$. We defer the formalization of elaboration of terms to Subsection 4.4.4.

In summary, the transformations made to the original gradual syntax are the following:

- **Intrinsic terms**. Every expression is translated to a term that carries its own typing derivation tree. This allows us to get rid of the type environment Γ in the typing rules. Although, making use of a type environment is still useful when reasoning about open terms for metric preservation.
- **Matching types**. The types in function application are matched through ascribing the argument to the type-and-effect expected by the function. Because of this transformation, the only typing rule where consistent subtyping is used is (I*G*ascr).

they are not necessary to illustrate the point.

- **Evidences**. Every ascription is augmented with evidence supporting the underlying subtyping judgment.
- **All values are ascribed**. In order to ease the proofs for type safety and soundness, we work on the premise that all values are ascribed simple values. Simple values u are introduced to the syntax in order to re-define values v as ascribed simple values. This is also reflected later on the reduction rules.

4.4.3. Reduction of Intrinsic Terms

We now focus on the reduction rules for $\lambda_{i\varepsilon}$. A caveat of gradual types is that even a program that typechecks may raise an error because of an optimistic judgment during typecheck might not hold on a runtime check. For instance, the program $(x + x) :: \mathbb{R}; ?x :: \mathbb{R}; 1x$ typechecks, as $?$ hides the real sensitivity of the expression statically, but should halt with an error during runtime since $2x$ is not a subtype of $1x$.

Reduction rules for $\lambda_{i\varepsilon}$ are given by proof normalization on the type safety proof of λ_s. However, because of the transformations during the elaboration of terms, particularly the ascribed values and matching types approaches, the type safety proof does not exactly mirror the evidence operations necessary to derive the reduction rules. For this reason, we need to reason about the type safety proof for an intermediate language with all ascription transformations mentioned before.

Consider the type safety proof of the application case in λ_s with ascribed values and matching types. The application expression will have the form

$$e = (\langle \lambda x : \tau_1'.e, \gamma' \rangle :: (x : \tau_1) \xrightarrow{\Sigma_2} \tau_2; \Sigma)\ (u :: \tau_1; \Sigma_1)$$

where both the closure and the value provided to it are ascribed, and the ascriptions have matching types.

First, to deal with the type safety proof of e, we need the typing rules for closures and contexts, as well as an auxiliary definition of well-formedness of type environments. Essentially, a type environment is well formed if for any variable $x \in dom(\gamma)$, $\Gamma(x)$ matches the type-and-effect of $\gamma(x)$. The actual definition of typing rules for closures and contexts, and well-formedness of type environments can be found in Appendix A (Figure A.2 and Definition 37, respectively).

The type safety proof essentially reduces a typing derivation $\mathcal{D}$ (of e) into a new one, $\mathcal{D}'$, and preserves the type of the original expression.

$$\mathcal{D} = (\text{TAPP})\ \frac{\mathcal{D}_1 \qquad (\text{TASCR})\ \dfrac{\Gamma \vdash u : \tau_1''; \Sigma_1'' \qquad \tau_1''; \Sigma_1'' <: \tau_1; \Sigma_1}{\Gamma \vdash u :: \tau_1; \Sigma_1\ :\ \tau_1; \Sigma_1}}{\Gamma \vdash (\langle \lambda x : \tau_1'.e, \gamma' \rangle :: (x : \tau_1) \xrightarrow{\Sigma_2} \tau_2; \Sigma)\ (u :: \tau_1; \Sigma_1)\ :\ [\Sigma_1/x]\tau_2; \Sigma + [\Sigma_1/x]\Sigma_2}$$

$$
\begin{array}{llll}
t \in \mathbb{T}[*] & ::= & \varepsilon r :: G \mid t + t \mid t \leq t & \text{terms} \\
& \mid & x \mid \varepsilon\ \lambda(x : g).t :: G \mid t\ t & \\
& \mid & \varepsilon \mathsf{tt} :: G \mid \varepsilon\ (\mathsf{inl}^{g} t) :: G \mid \varepsilon\ (\mathsf{inr}^{g} t) :: G & \\
& \mid & \mathsf{case}\ t\ \mathsf{of}\ \{\, x \Rightarrow t \,\}\ \{\, x \Rightarrow t \,\} & \\
& \mid & \varepsilon t :: G & \\
& \mid & \varepsilon \langle \lambda(x : g).t, \gamma \rangle :: G \mid \varepsilon \mathsf{ctx}(\gamma, t) :: G & \\
u \in \mathrm{SVal} & ::= & r \mid \langle \lambda(x : g).t, \gamma \rangle \mid \mathsf{tt} \mid \mathsf{inl}^{g} v \mid \mathsf{inr}^{g} v & \text{simple values} \\
v \in \mathrm{Val} & ::= & \varepsilon u :: G & \text{values}
\end{array}
$$

$\boxed{t \in \mathbb{T}[G]}$ **Well-typed intrinsic terms**

$$
\text{(IGRlit)}\ \frac{}{r \in \mathbb{T}[\mathbb{R}; \varnothing]}
\qquad
\text{(IGPlus)}\ \frac{t_1 \in \mathbb{T}[\mathbb{R}; \Xi_1] \qquad t_2 \in \mathbb{T}[\mathbb{R}; \Xi_2]}{t_1 + t_2 \in \mathbb{T}[\mathbb{R}; \Xi_1 + \Xi_2]}
\qquad
\text{(IGLeq)}\ \frac{t_1 \in \mathbb{T}[\mathbb{R}; \Xi_1] \qquad t_2 \in \mathbb{T}[\mathbb{R}; \Xi_2]}{t_1 \leq t_2 \in \mathbb{T}[\mathbb{B}; \infty(\Xi_1 + \Xi_2)]}
$$

$$
\text{(IGVar)}\ \frac{}{x^{G} \in \mathbb{T}[G]}
\qquad
\text{(IGLam)}\ \frac{t \in \mathbb{T}[g_2; \Xi]}{\lambda(x : g_1).t \in \mathbb{T}[(x : g_1) \xrightarrow{\Xi} g_2; \varnothing]}
$$

$$
\text{(IGApp)}\ \frac{t_1 \in \mathbb{T}[x : g_1 \xrightarrow{\Xi_2} g_2; \Xi] \qquad t_2 \in \mathbb{T}[g_1; \Xi_1]}{t_1\ t_2 \in \mathbb{T}[[\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2]}
\qquad
\text{(IGUnit)}\ \frac{}{\mathsf{tt} \in \mathbb{T}[\mathsf{unit}; \varnothing]}
\qquad
\text{(IGInl)}\ \frac{t \in \mathbb{T}[g_1; \Xi]}{\mathsf{inl}^{g_2} t \in \mathbb{T}[g_1 \,{}^{\Xi}{\oplus}^{\varnothing}\, g_2; \varnothing]}
$$

$$
\text{(IGInr)}\ \frac{t \in \mathbb{T}[g_2; \Xi]}{\mathsf{inr}^{g_1} t \in \mathbb{T}[g_1 \,{}^{\varnothing}{\oplus}^{\Xi}\, g_2; \varnothing]}
$$

$$
\text{(IGCase)}\ \frac{t_1 \in \mathbb{T}[g_{11} \,{}^{\Xi_{11}}{\oplus}^{\Xi_{12}}\, g_{12}; \Xi_1] \qquad t_2 \in \mathbb{T}[g_2; \Xi_2] \qquad t_3 \in \mathbb{T}[g_3; \Xi_3]}{\begin{array}{c}\mathsf{case}\ t_1\ \mathsf{of}\ \{\, x \Rightarrow t_2 \,\}\ \{\, y \Rightarrow t_3 \,\} \in \\ \mathbb{T}[[\Xi_1 + \Xi_{11}/x]g_2 \sqcup [\Xi_1 + \Xi_{12}/y]g_3; \Xi_1 \sqcup [\Xi_1 + \Xi_{11}/x]\Xi_2 \sqcup [\Xi_1 + \Xi_{12}/y]\Xi_3]\end{array}}
$$

$$
\text{(IGAscr)}\ \frac{t \in \mathbb{T}[G] \qquad \varepsilon \triangleright G \mathrel{\widetilde{<:}} G'}{\varepsilon t :: G' \in \mathbb{T}[G']}
$$

Figure 4.5: Syntax and type system of $\lambda_{i\varepsilon}$

where

$$\mathcal{D}_1 = (\textsc{Tascr})\ \frac{\mathcal{D}_2 \qquad (x:\tau_1') \xrightarrow{\Sigma_2'} \tau_2';\varnothing <: (x:\tau_1) \xrightarrow{\Sigma_2} \tau_2;\Sigma}{\Gamma \vdash \langle \lambda x:\tau_1'.e, \gamma' \rangle :: (x:\tau_1) \xrightarrow{\Sigma_2} \tau_2;\Sigma \ :\ (x:\tau_1) \xrightarrow{\Sigma_2} \tau_2;\Sigma}$$

$$\mathcal{D}_2 = (\textsc{Tclosure})\ \frac{\exists \Gamma' : \gamma' \vdash \Gamma' \qquad (\textsc{Tlam})\ \dfrac{\Gamma', x:\tau_1'; x \vdash e \ :\ \tau_2';\Sigma_2'}{\Gamma' \vdash \lambda x:\tau_1'.e \ :\ (x:\tau_1') \xrightarrow{\Sigma_2'} \tau_2';\varnothing}}{\Gamma \vdash \langle \lambda x:\tau_1'.e, \gamma' \rangle \ :\ (x:\tau_1') \xrightarrow{\Sigma_2'} \tau_2';\varnothing}$$

The adapted elimination rule, from (Tr-app) (Figure 3.5), follows:

$(\langle \lambda x:\tau_1'.e, \gamma' \rangle :: (x:\tau_1) \xrightarrow{\Sigma_2} \tau_2;\Sigma)\ (u :: \tau_1;\Sigma_1) \xrightarrow{\gamma} \mathsf{ctx}(\gamma_{ext}', e) :: [\Sigma_1/x]\tau_2;\Sigma + [\Sigma_1/x]\Sigma_2$

where $\gamma_{ext}' = \gamma'[x \mapsto u :: \tau_1';\Sigma_1]$.

And $\mathcal{D}'$ is derived as:

$$\mathcal{D}' = (\textsc{Tascr})\ \frac{\mathcal{D}_1' \qquad \tau_2'';\Sigma_2'' <: [\Sigma_1/x]\tau_2;\Sigma + [\Sigma_1/x]\Sigma_2}{\Gamma \vdash \mathsf{ctx}(\gamma_{ext}', e) :: [\Sigma_1/x]\tau_2;\Sigma + [\Sigma_1/x]\Sigma_2 \ :\ [\Sigma_1/x]\tau_2;\Sigma + [\Sigma_1/x]\Sigma_2}$$

where

$$\mathcal{D}_1' = (\textsc{Tctx})\ \frac{\exists \Gamma'' : \gamma_{ext}' \vdash \Gamma'' \qquad \Gamma'' \vdash e \ :\ \tau_2'';\Sigma_2''}{\Gamma \vdash \mathsf{ctx}(\gamma_{ext}', e) \ :\ \tau_2'';\Sigma_2''}$$

Then, we need to prove that:

- $\cdot \vdash u :: \tau_1';\Sigma_1 \ :\ \tau_1';\Sigma_1$, which in essence is proving that u can be ascribed to $\tau_1';\Sigma_1$, i.e. $\tau_1'';\Sigma_1'' <: \tau_1';\Sigma_1$. This is achieved by combining the knowledge from the subtyping judgments in the rule (Tascr): $\tau_1'';\Sigma_1'' <: \tau_1;\Sigma_1$ and $(x:\tau_1') \xrightarrow{\Sigma_2'} \tau_2';\varnothing <: (x:\tau_1) \xrightarrow{\Sigma_2} \tau_2;\Sigma$. Notice that whereas inferring the domain type of a function is fairly intuitive, the type-and-effect is not. We consider the type-and-effect domain of a function type-and-effect $(x:\tau_1) \xrightarrow{\Sigma_2} \tau_2;\Sigma$ to be $\tau_1;x$. This results in that $[\Sigma_1/x]\mathit{dom}((x:\tau_1') \xrightarrow{\Sigma_2'} \tau_2';\varnothing) <: [\Sigma_1/x]\mathit{dom}((x:\tau_1) \xrightarrow{\Sigma_2} \tau_2;\Sigma)$ gives us exactly the piece of knowledge, such as when combined with $\tau_1'';\Sigma_1'' <: \tau_1;\Sigma_1$, it proves that $\tau_1'';\Sigma_1'' <: \tau_1';\Sigma_1$. This is an important result as it will correspond with the gradual reduction rules.

- $\exists\Gamma'' : \gamma'_{ext} \vdash \Gamma''$: It suffices to pick $\Gamma'' = \Gamma', x : \tau'_1; \Sigma_1$. As we know that $\gamma' \vdash \Gamma'$, we only need to prove that $\cdot \vdash \gamma'_{ext}(x) : \Gamma''(x)$ which we already did.

- $\Gamma'' \vdash e : \tau''_2; \Sigma''_2$: By substitution lemma we know that the typing judgment holds (as $\Gamma'' = \Gamma', x : \tau'_1; \Sigma_1$) and $\tau''_2; \Sigma''_2 = [\Sigma_1/x]\tau'_2; [\Sigma_1/x]\Sigma'_2$.

- $[\Sigma_1/x]\tau'_2; [\Sigma_1/x]\Sigma'_2 <: [\Sigma_1/x]\tau_2; \Sigma + [\Sigma_1/x]\Sigma_2$: This follows directly from using the subtyping judgment in (TASCR), $(x : \tau'_1) \xrightarrow{\Sigma'_2} \tau'_2; \varnothing <: (x : \tau_1) \xrightarrow{\Sigma_2} \tau_2; \Sigma$, after applying a sensitivity substitution of x by Σ_1. Similar to the domain type-and-effect of a function, we define the type-and-effect co-domain of $(x : \tau_1) \xrightarrow{\Sigma_2} \tau_2; \Sigma$ as $\tau_2; \Sigma + \Sigma_2$. Thus, we can prove that $[\Sigma_1/x]cod((x : \tau'_1) \xrightarrow{\Sigma'_2} \tau'_2; \varnothing) <: [\Sigma_1/x]cod((x : \tau_1) \xrightarrow{\Sigma_2} \tau_2; \Sigma)$ and the sub-goal holds immediately.

Finally, we proved that the type-and-effect is preserved after one step, for the application case.

In the gradual language, the same notions apply. First, as intrinsic terms carry all the type information, instead of a well-formedness definition for type environments, now we need a well-formedness definition for substitutions with respect to a term:

Definition 35. *A substitution γ is well-formed with respect to an intrinsic term t if and only if:*

1. $FV(t) \subseteq \gamma$, *and*
2. $\forall x_i^G \in FV(t).\ \gamma(x_i) \in \mathbb{T}[G]$.

$$\mathcal{D} = (\text{IGAPP})\ \dfrac{\mathcal{D}_1 \qquad (\text{IGASCR})\ \dfrac{u \in \mathbb{T}[g''_1; \Xi''_1] \qquad \varepsilon_2 \triangleright g'_1; \Xi''_1 \mathrel{\widetilde{<:}} g_1; \Xi_1}{\varepsilon_2 u :: g_1; \Xi_1 \in \mathbb{T}[g_1; \Xi_1]}}{\begin{array}{c}(\varepsilon_1\langle\lambda(x : g''_1).t, \gamma'\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi)\ (\varepsilon_2 u :: g_1; \Xi_1) \\ \in \mathbb{T}[[\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2]\end{array}}$$

where

$$\mathcal{D}_1 = (\text{IGASCR})\ \dfrac{\mathcal{D}_2 \qquad \varepsilon_1 \triangleright (x : g'_1) \xrightarrow{\Xi'_2} g'_2; \varnothing \mathrel{\widetilde{<:}} (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi}{\varepsilon_1\langle\lambda(x : g'_1).t, \gamma'\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi \in \mathbb{T}[(x : g_1) \xrightarrow{\Xi_2} g_2; \Xi]}$$

$$\mathcal{D}_2 = (\text{IGCLOSURE})\ \dfrac{t \vdash \gamma' \qquad (\text{IGLAM})\ \dfrac{t \in \mathbb{T}[g'_2; \Xi'_2]}{\lambda(x : g'_1).t \in \mathbb{T}[(x : g'_1) \xrightarrow{\Xi'_2} g'_2; \varnothing]}}{\langle\lambda(x : g'_1).t, \gamma'\rangle \in \mathbb{T}[(x : g'_1) \xrightarrow{\Xi'_2} g'_2; \varnothing]}$$

- We need to extend γ' with a suitable value for x. Based on the static language, we can pass u but the type-and-effect will not match exactly with the one expected by the closure. So now we have to ascribe u to $g_1'; \Xi_1$ and produce an evidence that justifies the subtyping judgment. By inversion lemmas, we know that $idom(\varepsilon_1) \triangleright g_1; x \mathrel{\widetilde{<:}} g_1'; x$ and $icod(\varepsilon_1) \triangleright g_2'; \Xi_2' \mathrel{\widetilde{<:}} g_2; \Xi + \Xi_2$. Using consistent transitivity between ε_2 and $[\Xi_1/x]idom(\varepsilon_1)$, (if defined) we can justify $(\varepsilon_2 \circ^{<:} [\Xi_1/x]idom(\varepsilon_1)) \triangleright g_1''; \Xi_1'' \mathrel{\widetilde{<:}} g_1'; \Xi_1$.

- Notice that by extending γ' with $x \mapsto \varepsilon_2' u :: g_1'; \Xi_1$ we are implicitly stating that the sensitivity effect of x is now known and it corresponds to Ξ_1. However, all occurrences of x in (the types and sensitivity environments of) t are not yet updated. Therefore, we need to substitute all occurrences of x in the type-and-effects within t, i.e. $[\Xi_1/x]t$. It is important to note that the substitutions only occur in the type-and-effects and not in the terms itself, e.g. $[\Xi_1/x](x^{\mathbb{R};x} + y^{\mathbb{R};y}) = x^{\mathbb{R};\Xi_1} + y^{\mathbb{R};y}$, as the term substitution is explicitly handled by reduction. In the static setting this is implicitly handled by the existential predicates of type environments.

- We can now construct a term $t' = \mathsf{ctx}(\gamma'[x \mapsto (\varepsilon_2 \circ^{<:} [\Xi_1/x]idom(\varepsilon_1))u :: g_1'; \Xi_1], [\Xi_1/x]t)$ for handling the reduction of the lambda body. However, despite t' is well-typed, it does not preserve the type of the original term. To fix this, we can ascribe t' by the type-and-effect of the original term, $[\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2$, and justify the subtyping judgment with evidence $[\Xi_1/x]icod(\varepsilon_1)$.

- The final resulting term is derived as follows:

$$\varepsilon_{11}\mathsf{ctx}(\gamma'[x \mapsto v'], [\Xi_1/x]t) :: [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2$$

  where $v' = \varepsilon_2' u :: g_1'; \Xi_1$, $\varepsilon_{11} = [\Xi_1/x]icod(\varepsilon_1)$, $\varepsilon_2' = \varepsilon_2 \circ^{<:} [\Xi_1/x]idom(\varepsilon_1)$.

Figure 4.7 presents the reduction rules for $\lambda_{i\varepsilon}$. They are defined over configurations of terms and substitutions. However, as substitutions are not affected by reduction, we write the substitution just once in the reduction rules.

$$\begin{array}{lll} E & ::= & \square \mid E + t \mid v + E \mid E \le t \mid v \le E \mid E\ t \mid v\ E \quad \text{evaluation contexts} \\ & \mid & \mathsf{inl}^{\tau} E \mid \mathsf{inr}^{\tau} E \mid \mathsf{case}\ E\ \mathsf{of}\ \{x \Rightarrow t\}\ \{x \Rightarrow t\} \\ & \mid & \varepsilon E :: \mathrm{T} \end{array}$$

$\boxed{t \xmapsto{\gamma} t}$ **Reduction**

$$\text{(IGR}\rightarrow\text{)}\ \frac{t_1 \xrightarrow{\gamma} t_2}{t_1 \xmapsto{\gamma} t_2} \qquad \text{(IGRE)}\ \frac{t_1 \xmapsto{\gamma} t_2}{E[t_1] \xmapsto{\gamma} E[t_2]} \qquad \text{(IGRctx)}\ \frac{t_1 \xmapsto{\gamma'} t_2}{\mathsf{ctx}(\gamma', t_1) \xmapsto{\gamma} \mathsf{ctx}(\gamma', t_2)}$$

$$\text{(IGR}\rightarrow\text{err)}\ \frac{t_1 \xrightarrow{\gamma} \mathbf{error}}{t_1 \xmapsto{\gamma} \mathbf{error}} \qquad \text{(IGREerr)}\ \frac{t_1 \xmapsto{\gamma} \mathbf{error}}{E[t_1] \xmapsto{\gamma} \mathbf{error}} \qquad \text{(IGRctxerr)}\ \frac{t_1 \xmapsto{\gamma'} \mathbf{error}}{\mathsf{ctx}(\gamma', t_1) \xmapsto{\gamma} \mathbf{error}}$$

Figure 4.7: Dynamic semantics of $\lambda_{i\varepsilon}$

$\boxed{t \xrightarrow{\gamma} t}$ **Notions of reduction**

$$
\begin{array}{llcl}
(\text{IGR-PLUS}) & (\varepsilon_1 r_1 :: \mathbb{R}; \Xi_1) + (\varepsilon_2 r_2 :: \mathbb{R}; \Xi_2) & \longrightarrow & \varepsilon_3 r_3 :: \mathbb{R}; \Xi_1 + \Xi_2 \\
 & & \text{where} & r_3 = r_1 \llbracket + \rrbracket\, r_2 \\
 & & & \varepsilon_3 = \varepsilon_1 +_{\Xi} \varepsilon_2 \\
(\text{IGR-LEQ}) & (\varepsilon_1 r_1 :: \mathbb{R}; \Xi_1) \leq (\varepsilon_2 r_2 :: \mathbb{R}; \Xi_2) & \longrightarrow & \varepsilon_3 b :: \mathbb{B}; \infty(\Xi_1 + \Xi_2) \\
 & & \text{where} & b = r_1 \llbracket \leq \rrbracket\, r_2 \\
 & & & \varepsilon_3 = \varepsilon_1 \leq_{\Xi} \varepsilon_2 \\
(\text{IGR-VAR}) & x & \xrightarrow{\gamma} & \gamma(x) \\
(\text{IGR-LAM}) & \varepsilon\lambda(x : g_1').t :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi & \xrightarrow{\gamma} & \varepsilon\langle \lambda(x : g_1').t, \gamma\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi \\
(\text{IGR-APP}) & (\varepsilon_1\langle \lambda(x : g_1').t, \gamma'\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi)\ (\varepsilon_2 u :: g_1; \Xi_1) & & \\
 & & \xrightarrow{\gamma} & \begin{cases} \varepsilon_{11}\mathsf{ctx}(\gamma_{ext}, t_{body}) :: g_2'; \Xi + \Xi_2' & \\ \mathbf{error} & \text{if not defined} \end{cases} \\
 & & \text{where} & \gamma_{ext} = \gamma'[x \mapsto \varepsilon_2' u :: g_1'; \Xi_1] \\
 & & & t_{body} = [\Xi_1/x]t \\
 & & & \varepsilon_{11} = [\Xi_1/x]icod(\varepsilon_1) \\
 & & & \varepsilon_2' = \varepsilon_2 \circ^{<:} [\Xi_1/x]idom(\varepsilon_1) \\
 & & & g_2' = [\Xi_1/x]g_2 \\
 & & & \Xi_2' = [\Xi_1/x]\Xi_2 \\
(\text{IGR-CASE-1}) & \mathsf{case}(\varepsilon\mathsf{inl}^{g_{12}'}(\varepsilon_1 u :: g_{11}'; \Xi_{11}') :: g_{11}\ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\ g_{12}; \Xi_1) & & \\
 & \quad \mathsf{of}\ \{x \Rightarrow t_2^{g_2;\Xi_2}\}\ \{y \Rightarrow t_3^{g_3;\Xi_3}\} & & \\
 & & \xrightarrow{\gamma} & \begin{cases} \varepsilon_2'\mathsf{ctx}(\gamma_{ext}, t_{body}) :: g'; \Xi' \curlyvee \Xi_1 & \\ \mathbf{error} & \text{if not defined} \end{cases} \\
 & & \text{where} & \gamma_{ext} = \gamma[x \mapsto \varepsilon_1' u :: g_{11}; \Xi_x] \\
 & & & t_{body} = [\Xi_x/x]t_2 \\
 & & & \varepsilon_1' = \varepsilon_1 \circ^{<:} ileft(\varepsilon) \\
 & & & \varepsilon_2' = \mathcal{I}_{<:}(g_2'; \Xi_2', g'; \Xi') \curlyvee_{\Xi} \varepsilon \\
 & & & \Xi_x = \Xi_1 + \Xi_{11} \\
 & & & \Xi_y = \Xi_1 + \Xi_{12} \\
 & & & g_2' = [\Xi_x/x]g_2 \\
 & & & \Xi_2' = [\Xi_x/x]\Xi_2 \\
 & & & g' = [\Xi_x/x]g_2 \curlyvee [\Xi_y/y]g_3 \\
 & & & \Xi' = [\Xi_x/x]\Xi_2 \curlyvee [\Xi_y/y]\Xi_3 \\
(\text{IGR-CTX}) & \varepsilon\mathsf{ctx}(\gamma', v) :: g; \Xi & \longrightarrow & \varepsilon v :: g; \Xi \\
(\text{IGR-ASCR}) & \varepsilon(\varepsilon' u :: g'; \Xi') :: g; \Xi & \longrightarrow & \begin{cases} \varepsilon' \circ^{<:} \varepsilon u :: g; \Xi & \\ \mathbf{error} & \text{if not defined} \end{cases}
\end{array}
$$

Figure 4.6: Dynamic semantics of $\lambda_{i\varepsilon}$

Rules (IGr-var), (IGr-lam) and (IGr-ctx) present no novelty with respect to their static counterpart, except for the fact that they operate on ascribed terms. Rule (IGr-plus) follows the same pattern as (Tr-plus) and the evidence is computed using the inversion operator $+_{\Xi}$. The evidence operator $+_{\Xi}$ works only on the sensitivity environment parts of the evidences by adding them, reassembling the operations done at the ascriptions level. For example, $\langle \mathbb{R}; 2x, \mathbb{R}; 3x \rangle +_{\Xi} \langle \mathbb{R}; 4x, \mathbb{R}; 5x \rangle = \langle \mathbb{R}; 2x + 4x, \mathbb{R}; 3x + 5x \rangle = \langle \mathbb{R}; 6x, \mathbb{R}; 8x \rangle$. The formal definition of $+_{\Xi}$, and all inversion functions, are presented in Figure 4.8. Rule (IGr-leq) is analogous to (IGr-plus). The application rule, (IGr-app), is defined as explained before by using the inversion functions *idom* and *icod*.

Although the derivation process is analogous to (IGr-app), rule (IGr-case-1) is more complex. Evidence ε'_1, for the stored value, is computed using the inversion function *ileft* (instead of *idom* in (IGr-app)). Furthermore, no substitution is needed in the computation of ε'_1, since there is no free variables in its types. Since the body terms have no explicit evidences, in order to produce an evidence ε'_2 for the context term we use the interior operator. For ε'_2 to fully resemble the ascriptions operations we must integrate the information in the sensitivity environment parts of ε. This is done by using the inversion operator $\curlyvee_{\Xi}$, which is also defined in Figure 4.8. Rule (IGr-case-2), corresponding to the inr case, is left out as its definition is analogous to (IGr-case-1).

Finally, rule (IGr-ascr) eliminates ascriptions by keeping only the outer one. Consistent transitivity is performed in order to justify the new ascribed (simple) value.

So far, we have defined the runtime semantics of $\lambda_{i\varepsilon}$ that, by translation, also define the runtime semantics for λ_i. The reduction rules are fully derived by proof normalization against the type safety proof. Nevertheless, the type safety proof used belongs to a modified version of λ_s that preserve types and have matching types on applications (by using ascriptions). However, we argue that the derivation process still follows the AGT methodology. This seems to conclude the derivation of the dynamic semantics of λ_i. However, we still do not address the actual elaboration of terms and whether it preserves important properties such as typeability or precision.

4.4.4. Elaboration of Terms

In order to justify that reduction rules of $\lambda_{i\varepsilon}$ preserve the semantics of λ_i, we have to formalize the elaboration of terms and establish whether it preserves typeability and precision. Figure 4.9 shows the elaboration from gradual expressions to evidence-augmented intrinsic terms. Judgment $\Gamma \vdash e : G \rightsquigarrow_{\varepsilon} t^G$ denotes the elaboration of the intrinsic term t^G from the expression e, where e has type G under the type environment Γ.

Rules (ELplus), (ELleq) and (ELcase) work by translating each sub-expression pointwise. Rules (ELrlit), (ELlam), (ELunit), (ELinl) and (ELinr) insert ascriptions and justify them with evidences produced by the interior operator. The computation of these evidences never fails since both type-and-effects are always the same, so the insertion of these ascriptions are harmless. Rule (ELapp) ascribes the argument by the expected type while preserving its sensitivity effect. The interior operator is used again to produce the evidence

$\boxed{idom(\varepsilon)}$

$$idom(\langle (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}; \Xi_1, (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22}; \Xi_2 \rangle) = \langle g_{21}; x, g_{11}; x \rangle$$

$\boxed{icod(\varepsilon)}$

$$icod(\langle (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}; \Xi_1, (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22}; \Xi_2 \rangle) = \langle g_{12}; \Xi_1 + \Xi_{12}, g_{22}; \Xi_2 + \Xi_{22} \rangle$$

$\boxed{ileft(\varepsilon)}$

$$ileft(\langle g_{11} \ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1, g_{21} \ {}^{\Xi_{21}}\!\oplus^{\Xi_{22}} g_{22}; \Xi_2 \rangle) = \langle g_{11}; \Xi_1 + \Xi_{11}, g_{21}; \Xi_2 + \Xi_{21} \rangle$$

$\boxed{iright(\varepsilon)}$

$$iright(\langle g_{11} \ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1, g_{21} \ {}^{\Xi_{21}}\!\oplus^{\Xi_{22}} g_{22}; \Xi_2 \rangle) = \langle g_{12}; \Xi_1 + \Xi_{12}, g_{22}; \Xi_2 + \Xi_{22} \rangle$$

$\boxed{\varepsilon +_{\Xi} \varepsilon}$

$$\langle \mathbb{R}; \Xi_{11}, \mathbb{R}; \Xi_{12} \rangle +_{\Xi} \langle \mathbb{R}; \Xi_{21}, \mathbb{R}; \Xi_{22} \rangle = \langle \mathbb{R}; \Xi_{11} + \Xi_{21}, \mathbb{R}; \Xi_{12} + \Xi_{22} \rangle$$

$\boxed{\varepsilon \leq_{\Xi} \varepsilon}$

$$\langle \mathbb{R}; \Xi_{11}, \mathbb{R}; \Xi_{12} \rangle \leq_{\Xi} \langle \mathbb{R}; \Xi_{21}, \mathbb{R}; \Xi_{22} \rangle = \langle \mathbb{B}; \infty(\Xi_{11} + \Xi_{21}), \mathbb{B}; \infty(\Xi_{12} + \Xi_{22}) \rangle$$

$\boxed{\varepsilon \curlyvee_{\Xi} \varepsilon}$

$$\langle g_{11}; \Xi_{11}, g_{12}; \Xi_{12} \rangle \curlyvee_{\Xi} \langle g_{21}; \Xi_{21}, g_{22}; \Xi_{22} \rangle = \langle g_{11}; \Xi_{11} \curlyvee \Xi_{21}, g_{12}; \Xi_{12} \curlyvee \Xi_{22} \rangle$$

Figure 4.8: Inversion functions on evidences

with the difference that this computation may fail if g_1' is not a subtype of g_1. Finally, (ELascr) produces a initial evidence to justify the ascription. This ascription can also fail to compute if subtyping is not satisfied.

In order to conclude with the derivation of the dynamic semantics of $\lambda_{i\varepsilon}$ we establish that elaboration preserves typing. Notice that elaboration rules only insert trivial ascriptions and enrich derivations with evidence and ascriptions. As this derivations are represented as intrinsic terms, by construction, elaboration of terms trivially preserves typing.

Proposition 29 (Elaboration preserves typing)**.** *If* $\Gamma \vdash e : G$ *and* $\Gamma \vdash e : G \rightsquigarrow_{\varepsilon} t^G$*, then* $t^G \in \mathbb{T}[G]$.

$\boxed{\Gamma \vdash e \;:\; G \rightsquigarrow_{\varepsilon} t^{G}}$ **Elaboration of Intrinsic Terms**

(ELRlit)
$$\frac{\varepsilon = \mathcal{I}_{<:}(\mathbb{R};\varnothing,\mathbb{R};\varnothing)}{\Gamma \vdash r \;:\; \mathbb{R};\varnothing \rightsquigarrow_{\varepsilon} \varepsilon r :: \mathbb{R};\varnothing}$$

(ELPlus)
$$\frac{\Gamma \vdash e_1 \;:\; \mathbb{R};\Xi_1 \rightsquigarrow_{\varepsilon} t_1 \qquad \Gamma \vdash e_2 \;:\; \mathbb{R};\Xi_2 \rightsquigarrow_{\varepsilon} t_2}{\Gamma \vdash e_1 + e_2 \;:\; \mathbb{R};\Xi_1 + \Xi_2 \rightsquigarrow_{\varepsilon} t_1 + t_2}$$

(ELLeq)
$$\frac{\Gamma \vdash e_1 \;:\; \mathbb{R};\Xi_1 \rightsquigarrow_{\varepsilon} t_1 \qquad \Gamma \vdash e_2 \;:\; \mathbb{R};\Xi_2 \rightsquigarrow_{\varepsilon} t_2}{\Gamma \vdash e_1 \leq e_2 \;:\; \mathbb{B};\infty(\Xi_1 + \Xi_2) \rightsquigarrow_{\varepsilon} t_1 \leq t_2}$$

(ELVar)
$$\frac{\Gamma(x) = G}{\Gamma \vdash x \;:\; G \rightsquigarrow_{\varepsilon} x^{G}}$$

(ELLam)
$$\frac{\Gamma, x : g_1; x \vdash e \;:\; g_2;\Xi \rightsquigarrow_{\varepsilon} t \qquad \varepsilon = \mathcal{I}_{<:}((x : g_1) \xrightarrow{\Xi} g_2;\varnothing,(x : g_1) \xrightarrow{\Xi} g_2;\varnothing)}{\Gamma \vdash \lambda(x : g_1).e \;:\; (x : g_1) \xrightarrow{\Xi} g_2;\varnothing \rightsquigarrow_{\varepsilon} \varepsilon\lambda(x : g_1).t :: (x : g_1) \xrightarrow{\Xi} g_2;\varnothing}$$

(ELApp)
$$\frac{\Gamma \vdash e_1 \;:\; (x : g_1) \xrightarrow{\Xi_2} g_2;\Xi \rightsquigarrow_{\varepsilon} t_1 \qquad \Gamma \vdash e_2 \;:\; g_1';\Xi_1 \rightsquigarrow_{\varepsilon} t_2 \qquad \varepsilon_2 = \mathcal{I}_{<:}(g_1';\Xi_1,g_1;\Xi_1)}{\Gamma \vdash e_1\; e_2 \;:\; [\Xi_1/x]g_2;\Xi + [\Xi_1/x]\Xi_2 \rightsquigarrow_{\varepsilon} t_1\; (\varepsilon_2 t_2 :: g_1;\Xi_1)}$$

(ELUnit)
$$\frac{\varepsilon = \mathcal{I}_{<:}(\mathsf{unit};\varnothing,\mathsf{unit};\varnothing)}{\Gamma \vdash \mathsf{tt} \;:\; \mathsf{unit};\varnothing \rightsquigarrow_{\varepsilon} \varepsilon\mathsf{tt} :: \mathsf{unit};\varnothing}$$

(ELInl)
$$\frac{\Gamma \vdash e \;:\; g_1;\Xi_1 \rightsquigarrow_{\varepsilon} t \qquad \varepsilon = \mathcal{I}_{<:}(g_1 \;{}^{\Xi_1}\!\oplus^{\varnothing}\; g_2;\varnothing,g_1 \;{}^{\Xi_1}\!\oplus^{\varnothing}\; g_2;\varnothing)}{\Gamma \vdash \mathsf{inl}^{g_2} e \;:\; g_1 \;{}^{\Xi_1}\!\oplus^{\varnothing}\; g_2;\varnothing \rightsquigarrow_{\varepsilon} \varepsilon\mathsf{inl}^{g_2} t :: g_1 \;{}^{\Xi_1}\!\oplus^{\varnothing}\; g_2;\varnothing}$$

(ELInr)
$$\frac{\Gamma \vdash e \;:\; g_2;\Xi_2 \rightsquigarrow_{\varepsilon} t \qquad \varepsilon = \mathcal{I}_{<:}(g_1 \;{}^{\varnothing}\!\oplus^{\Xi_2}\; g_2;\varnothing,g_1 \;{}^{\varnothing}\!\oplus^{\Xi_2}\; g_2;\varnothing)}{\Gamma \vdash \mathsf{inr}^{g_1} e \;:\; g_1 \;{}^{\varnothing}\!\oplus^{\Xi_2}\; g_2;\varnothing \rightsquigarrow_{\varepsilon} \varepsilon\mathsf{inr}^{g_1} t :: g_1 \;{}^{\varnothing}\!\oplus^{\Xi_2}\; g_2;\varnothing}$$

(ELCase)
$$\frac{\begin{array}{c}\Gamma \vdash e_1 \;:\; g_{11} \;{}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\; g_{12};\Xi_1 \rightsquigarrow_{\varepsilon} t_1 \\ \Gamma, x : g_{11}; x \vdash e_2 \;:\; g_2;\Xi_2 \rightsquigarrow_{\varepsilon} t_2 \qquad \Gamma, y : g_{12}; y \vdash e_3 \;:\; g_3;\Xi_3 \rightsquigarrow_{\varepsilon} t_3\end{array}}{\begin{array}{c}\Gamma \vdash \mathsf{case}\; e_1\; \mathsf{of}\,\{\, x \Rightarrow e_2 \,\}\;\{\, y \Rightarrow e_3 \,\} \;: \\ [\Xi_1 + \Xi_{11}/x]g_2 \sqcup [\Xi_1 + \Xi_{12}/y]g_3;\Xi_1 \sqcup [\Xi_1 + \Xi_{11}/x]\Xi_2 \sqcup [\Xi_1 + \Xi_{12}/y]\Xi_3 \\ \rightsquigarrow_{\varepsilon} \mathsf{case}\; t_1\; \mathsf{of}\,\{\, x \Rightarrow t_2 \,\}\;\{\, y \Rightarrow t_3 \,\}\end{array}}$$

(ELAscr)
$$\frac{\Gamma \vdash e \;:\; G \rightsquigarrow_{\varepsilon} t \qquad \varepsilon = \mathcal{I}_{<:}(G, G')}{\Gamma \vdash e :: G' \;:\; G' \rightsquigarrow_{\varepsilon} \varepsilon t :: G'}$$

Figure 4.9: Elaboration of $\lambda_{i\varepsilon}$ from λ_i

4.5. Properties

The derived gradual language $\lambda_{i\varepsilon}$ satisfy several important properties. The first one is type safety: close terms do not get stuck, but they still can halt with a runtime error.

Proposition 30 (Type safety). *If $t \in \mathbb{T}[g; \Xi]$, then one of the following is true:*

- *t is a value v.*
- *$t \xmapsto{\varnothing} t'$ for some $t' \in \mathbb{T}[g; \Xi]$.*
- *$t \xmapsto{\varnothing}$ **error***

In addition, we also prove that the gradual type system is equivalent to the statically-typed system for fully-static expressions and that it satisfies the gradual guarantee [35]. The former and the static component of the gradual guarantee were previously stated in Section 4.3. First, the static semantics of λ_i and λ_s are equivalent for fully-static expressions, i.e. expressions whose gradual sensitivity occurrences are all fully-static.

Proposition 25 (Equivalence for fully-static expressions). *Let e be a fully-static expression and G a static type ($G = \mathrm{T}$). $\cdot \vdash_s e : \mathrm{T}$ if and only if $\cdot \vdash e : \mathrm{T}$.*

The static semantics of λ_s also satisfy the static gradual guarantee: typeability is monotone to imprecision.

Proposition 26 (Static gradual guarantee). *Let e_1 and e_2 be two closed expressions such that $e_1 \sqsubseteq e_2$ and $\cdot \vdash e_1 : G_1$. Then, $\cdot \vdash e_2 : G_2$ and $G_1 \sqsubseteq G_2$.*

Finally, we also prove that λ_i satisfies the dynamic component of the gradual guarantee: any program that reduces without error will continue to do so if precision is removed. In related work [36, 37], where soundness of the type system is an hyperproperty, simultaneous satisfiability of this proposition and soundness has resulted to be fairly challenging. For this reason, we have particular interest in formally proving both.

Proposition 31 (Dynamic gradual guarantee). *Suppose $t_{11} \sqsubseteq t_{12}$ and $\gamma_1 \sqsubseteq \gamma_2$. If $t_{11} \xmapsto{\gamma_1} t_{21}$ then $t_{12} \xmapsto{\gamma_2} t_{22}$ where $t_{21} \sqsubseteq t_{22}$.*

Soundness. Just as in the static setting, we state metric preservation by making use of logical relations (Figure 4.10).

For proving soundness we make extensive use of a compatibility lemma, which states that if two values are related, then the computations of ascribing both values to the same type-and-effect are also related. Lemma 32 captures this formally:

Lemma 32. *If $(v_1, v_2) \in \mathcal{V}_\Delta[\![G]\!]$ and $\varepsilon_i \ \triangleright\ G \ \widetilde{<:}\ G'$, then $\forall \Gamma, \gamma_1, \gamma_2 : (\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$ it follows that $(\varepsilon_1 v_1 :: G' \mid \gamma_1, \varepsilon_2 v_2 :: G' \mid \gamma_2) \in \mathcal{T}_\Delta[\![G']\!]$.*

$$\begin{aligned}
(v_1, v_2) \in \mathsf{Atom}[\![G]\!] &\iff v_1 \in \mathbb{T}[G] \land v_2 \in \mathbb{T}[G] \\
(v_1, v_2) \in \mathcal{V}_\Delta[\![\mathbb{R}; \Xi]\!] &\iff (v_1, v_2) \in \mathsf{Atom}[\![\mathbb{R}; \Xi]\!] \land \\
&\quad \neg\big(\Delta \cdot (\Xi'_1 \curlyvee \Xi'_2) \mathrel{\widetilde{<}} |u_1 - u_2|\big) \\
&\quad \text{where } v_i = \langle\, \mathbb{R}; \varnothing, \mathbb{R}; \Xi'_i \,\rangle\, u_i :: \mathbb{R}; \Xi \\
(v_1, v_2) \in \mathcal{V}_\Delta[\![\mathsf{unit}; \Xi]\!] &\iff (v_1, v_2) \in \mathsf{Atom}[\![\mathsf{unit}; \Xi]\!] \land \\
&\quad u_1 = \mathsf{tt} \land u_2 = \mathsf{tt} \\
&\quad \text{where } v_i = \varepsilon_i u_i :: \mathsf{unit}; \Xi \\
(v_1, v_2) \in \mathcal{V}_\Delta[\![g_1 \,{}^{\Xi_1}{\oplus}^{\Xi_2}\, g_2; \Xi]\!] &\iff (v_1, v_2) \in \mathsf{Atom}[\![g_1 \,{}^{\Xi_1}{\oplus}^{\Xi_2}\, g_2; \Xi]\!] \land \\
&\quad \Delta \cdot (\Xi'_1 \curlyvee \Xi'_2) \mathrel{\widetilde{\leq}} \infty \implies \big(\forall \Gamma, \gamma_1, \gamma_2 : (\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]. \\
&\quad (useL(v_1) \mid \gamma_1, useL(v_2) \mid \gamma_2) \in \mathcal{T}_\Delta[\![g_1; \Xi + \Xi_1]\!] \lor \\
&\quad (useR(v_1) \mid \gamma_1, useR(v_2) \mid \gamma_2) \in \mathcal{T}_\Delta[\![g_2; \Xi + \Xi_2]\!]\big) \\
&\quad \text{where } v_i = \langle\, G'_{i1}; \varnothing, G'_{i2}; \Xi'_i \,\rangle\, u_i :: g_1 \,{}^{\Xi_1}{\oplus}^{\Xi_2}\, g_2; \Xi \\
(v_1, v_2) \in \mathcal{V}_\Delta[\![(x : g_1) \xrightarrow{\Xi_2} g_2; \Xi]\!] &\iff (v_1, v_2) \in \mathsf{Atom}[\![(x : g_1) \xrightarrow{\Xi_2} g_2; \Xi]\!] \land \\
&\quad \big(\forall \Gamma, \gamma_1, \gamma_2, v'_1, v'_2, \Xi_1 : \\
&\quad (v'_1, v'_2) \in \mathcal{V}_\Delta[\![g_1; \Xi_1]\!] \land (\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!] \ . \\
&\quad (v_1\ v'_1 \mid \gamma_1, v_2\ v'_2 \mid \gamma_2) \in \mathcal{T}_\Delta[\![[\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2]\!]\big) \\
(t_1 \mid \gamma_1, t_2 \mid \gamma_2) \in \mathcal{T}_\Delta[\![G]\!] &\iff (t_1 \xmapsto{\gamma_1}{}^{*}\ v_1 \land t_2 \xmapsto{\gamma_2}{}^{*}\ v_2) \implies (v_1, v_2) \in \mathcal{V}_\Delta[\![G]\!] \\
(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!] &\iff dom(\gamma_1) = dom(\gamma_2) = dom(\Gamma) \land \\
&\quad \forall x \in dom(\Gamma).(\gamma_1(x), \gamma_2(x)) \in \mathcal{V}_\Delta[\![\Gamma(x)]\!]
\end{aligned}$$

$$\begin{aligned}
useL(v) &= ileft(\varepsilon)v' :: g_1; \Xi + \Xi_1 &\quad \text{if } v = \varepsilon\mathsf{inl}v' :: g_1 \,{}^{\Xi_1}{\oplus}^{\Xi_2}\, g_2; \Xi \\
useR(v) &= iright(\varepsilon)v' :: g_2; \Xi + \Xi_2 &\quad \text{if } v = \varepsilon\mathsf{inr}v' :: g_1 \,{}^{\Xi_1}{\oplus}^{\Xi_2}\, g_2; \Xi
\end{aligned}$$

Figure 4.10: Logical relations for gradual sensitivity soundness

Furthermore, in order to prove this lemma the logical relation has to reason about worst-case scenarios. Otherwise, if we would take optimistic assumptions, one could always ascribe two related values to a less sensitive type-and-effect (justified by gradual plausibility) and these terms should be related as well. However, this might not hold. For the same reason, we have to make use of the most precise information, so instead of using the sensitivity environment in the ascriptions, we leverage the information encoded in the evidences. Additionally, in order to reason about what evidences justify, typechecking is also needed. For this $\mathsf{Atom}[\![G]\!]$ is introduced and used in all values logical relations. Although integrating the type system within the logical relations can be prohibitive (given the conservative nature of a type system) we assume this cost in order to ease the proof work.

In summary, with respect to their static counterpart, the key differences are:

- In order to account for the optimistic assumptions of the gradual typechecker and gradual operations, such as $\widetilde{<:}$, we must reason about worst-case scenarios.
- We introduce the $\mathsf{Atom}[\![G]\!]$ relation that typechecks the values in the relation. This relation is used in all value relations.

We now discuss each relation in detail.

Related numbers. First, instead of using the information in the sensitivity environment in the ascription, Ξ, we use the most precise information, namely the one inside the evidences. Between the two sensitivity environments inside the evidences of v_1 and v_2, we compute the join in order to account for the worst case. Notice that $\Delta \cdot (\Xi'_1 \curlyvee \Xi'_2)$ returns a gradual sensitivity, i.e. an interval. Let us call it d. Reasoning about the worst case for the inequality is a bit challenging: the judgment cannot be directly lifted by using $\widetilde{\leq}$, e.g. $|u_1 - u_2| \mathrel{\widetilde{\leq}} d$, since $\widetilde{\leq}$ is already an optimistic operator. We actually want $|u_1 - u_2|$ to be less or equal than *any* of the values within d. In other words, we do *not* want to exist a value within d that is less than $|u_1 - u_2|$, so $\neg(d \mathrel{\widetilde{<}} |u_1 - u_2|)$. Notice that we make use of $\widetilde{<}$ (strict consistent *less than*) instead of $\widetilde{\leq}$, e.g. $[2,4] \mathrel{\widetilde{<}} 2$ is not true. For instance, let $v_1 = \langle \mathbb{R}; \varnothing, \mathbb{R}; [1,4]x \rangle\, 2 :: \mathbb{R}; [0,10]x$, $v_2 = \langle \mathbb{R}; \varnothing, \mathbb{R}; [1,5]x \rangle\, 5 :: \mathbb{R}; [0,10]x$ and $\Delta = 3x$. Consider the following proof for $(v_1, v_2) \in \mathcal{V}_\Delta[\![\mathbb{R}; [0,10]x]\!]$:

- $(v_1, v_2) \in \mathsf{Atom}[\![\mathbb{R}; [0,10]x]\!]$: Trivially both values typecheck.
- $\neg\big(\Delta \cdot (\Xi'_1 \curlyvee \Xi'_2) \mathrel{\widetilde{<}} |u_1 - u_2|\big)$:

$$\begin{aligned}
\neg\big((3x) \cdot ([1,4]x \curlyvee [1,5]x) \mathrel{\widetilde{<}} |2-5|\big) &\iff \neg\big((3x) \cdot ([1,5]x) \mathrel{\widetilde{<}} |2-5|\big) \\
&\iff \neg\big([3,15] \mathrel{\widetilde{<}} 3\big) \\
&\iff \neg(\bot) \\
&\iff \top
\end{aligned}$$

Informally, it is useful to think of Δ as the distance of the input, $\Xi'_1 \curlyvee \Xi'_2$ as the predicted sensitivity, and $\Delta \cdot (\Xi'_1 \curlyvee \Xi'_2)$ as the predicted output distance. For the predicted sensitivity to be sound, there must not exist *any* value within the predicted output distance, $[3,15]$, that is less than the actual distance, 3. As $[3,15] \mathrel{\widetilde{<}} 3$ is false, both values are related.

Related sums. The reasoning for this relation is similar to the rule for related numbers. Evidence is used in the same way, but now in order to account for the worst case, we need to take the best case on the hypothesis side of the implication as the consistent judgment is in a contravariant position. Therefore, the consistent operator $\widetilde{<:}$ is used directly, without the need for double negations. The rest is analogous to their static counterpart. Functions *useL* and *useR* now operate on ascribed values so *ileft* and *iright* are used.

Related unit literals, functions and substitutions. These relations present no novelty with respect to their static counterparts except for the fact that typechecking is now performed for value relations. Everything else is the natural lifting to a gradual setting.

Finally, in order to enounce metric preservation we need a notion of well-formedness of a type environment with respect to a terms:

Definition 36 (Type environment well-formedness)**.** *A type environment* Γ *is well-formed with respect to an intrinsic term* t*, denoted* $t \vdash \Gamma$ *if and only if* $FV(t) \subseteq dom(t)$ *and* $\forall x^G \in FV(t).\ \Gamma(x) = G$*.*

Now we can establish soundness for gradual terms: if an open intrinsic term type checks and we have a type environment Γ that closes it (at the type level), then for any two related substitutions γ_1, γ_2, the computations of the term enclosed by γ_1 and γ_2 are related.

Theorem 33 ((Gradual) metric preservation) If $t \in \mathbb{T}[G]$ and $t \vdash \Gamma$, then $\forall \Delta, \gamma_1, \gamma_2$ such that $\Gamma \vdash \Delta$ and $(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$, it follows that $(t \mid \gamma_1, t \mid \gamma_2) \in \mathcal{T}_\Delta[\![G]\!]$.

Gradual metric preservation in action. Let $t = \varepsilon_2(x + \varepsilon_1 1 :: \mathbb{R}; ?x) :: \mathbb{R}; [0,2]x$, where $\varepsilon_1 = \langle \mathbb{R}; \varnothing, \mathbb{R}; ?x \rangle$ and $\varepsilon_2 = \langle \mathbb{R}; 1x, \mathbb{R}; [1,2]x \rangle$. Consider $\Gamma = x : \mathbb{R}; x$, $\Delta = 2x$, $\gamma_1 = \{ x \mapsto \langle \mathbb{R}; \varnothing, \mathbb{R}; x \rangle\, 4 :: \mathbb{R}; x \}$ and $\gamma_2 = \{ x \mapsto \langle \mathbb{R}; \varnothing, \mathbb{R}; x \rangle\, 5 :: \mathbb{R}; x \}$. Let us establish that $(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$:

- $dom(\Gamma) = dom(\gamma_1) = dom(\gamma_2)$: Trivial.
- $(\langle \mathbb{R}; \varnothing, \mathbb{R}; x \rangle 4 :: \mathbb{R}; x, \langle \mathbb{R}; \varnothing, \mathbb{R}; x \rangle 5 :: \mathbb{R}; x) \in \mathcal{V}_\Delta[\![\mathbb{R}; x]\!]$: $\neg(2 \mathrel{\widetilde{\lesssim}} 1) \iff \top$.

By Proposition 33 we know that $(t \mid \gamma_1, t \mid \gamma_2) \in \mathcal{T}_\Delta[\![\mathbb{R}; [0,2]x]\!]$. Let us see why:

- $t \xrightarrow{\gamma_1}{}^{*} \varepsilon\, 5 :: \mathbb{R}; [0,2]x$, where $\varepsilon = \langle \mathbb{R}; \varnothing, \mathbb{R}; [1,2]x \rangle$. Analogously, $t \xrightarrow{\gamma_2}{}^{*} \varepsilon\, 6 :: \mathbb{R}; [0,2]x$.
- $(\varepsilon 5 :: \mathbb{R}; [0,2]x, \varepsilon 6 :: \mathbb{R}; [0,2]x) \in \mathcal{V}_\Delta[\![\mathbb{R}; [0,2]x]\!]$:

$$\begin{aligned}
\neg\big((2x)\cdot([1,2]x \curlyvee [1,2]x) \mathrel{\widetilde{\lesssim}} |5-6|\big) &\iff \neg\big((2x)\cdot([1,2]x) \mathrel{\widetilde{\lesssim}} |5-6|\big)\\
&\iff \neg\big([2,4] \mathrel{\widetilde{\lesssim}} 1\big)\\
&\iff \neg(\bot)\\
&\iff \top
\end{aligned}$$

Notice that if we had used the ascribed effect, $[0,2]x$, instead of the ones in the evidences the last judgment would have fail because $\neg\big((2x)\cdot([0,2]x) \mathrel{\widetilde{\lesssim}} |5-6|\big) \iff \bot$. Along with the pessimistic reasoning in the logical relations, this notion is the most challenging part of characterizing metric preservation for gradual sensitivity types.

In Appendix B we provide proofs for all the results presented in this section. We argue that type safety and soundness for λ_s are specific cases of their gradual counterparts, so their proofs are corollaries of the properties presented in this section.

Chapter 5

Conclusions

In this work we have presented λ_i, a gradual language with support for sensitivity reasoning. Gradual sensitivity types allow a programmer to smoothly evolve a program with simple types by incrementally adding sensitivity information. This, for example, enables the use of library code in contexts such as differentially private algorithms. By developing a gradual sensitivity language, we have proven that gradual typing can be applied to sensitivity typing. Furthermore, we proved that a sound gradual sensitivity type system is possible. We explained how to apply, step-by-step, the AGT methodology to λ_s, a language with sensitivity types, presenting a novel interpretation of gradual sensitivities. Also, given the formalization of λ_s, we showed an application of AGT to a language with explicit substitutions. Finally, we proved three important properties of λ_i: type safety, soundness and the gradual guarantee. For this, we presented a characterization for soundness of gradual sensitivity types. In particular, we found that the use of intervals in evidences, proposed by Toro *et al* [36], was enough for the derived gradual language to be sound, without the need for *ad-hoc* changes to the runtime semantics.

One of the key challenges of this work was to design small-step semantics for λ_s (in contrast to the big-step semantics of SAX) that were type safe, specially in the presence of explicit substitution. Regarding the application of AGT, several simplifications and transformation techniques were applied to mitigate challenges in the proof work. In particular, a challenging task was to define the logical relations of metric preservation in the context of a gradual language, where plausibility has to be carefully managed in order to avoid making unsound assumptions.

Whereas λ_i satisfies many interesting properties, other properties from the refined criteria for gradual typing [35] are left for future work. However, we conjecture that the most challenging proof work was in proving both soundness and the dynamic gradual guarantee. Additionally, our gradual language is limited in the sense that it only presents a subset of the functionality originally found in SAX. In particular, λ_i does not provide support for product types which are highly used in the context of differential privacy and data manipulation. Although we argue that this subset is enough to capture the main challenges of gradual sensitivity types, in order to aim for a practical implementation, extensions must be done.

In relation to differential privacy, we believe that a semi-gradual differential privacy language could be developed by using the multi-language design of DUET or JAZZ. Our gradual sensitivity type system could be embedded into a privacy type system by adding gradual-to-static ascriptions at the boundaries of both disciplines. Another interesting track for future work is to add support for mutable references. We believe it is worthwhile exploring whether tension would be found between soundness and the dynamic gradual guarantee, in a context with mutable references, similar to Toro *et al* [36] where soundness was also a hyperproperty.

Bibliography

[1] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pp. 111–125, 2008.

[2] D. Barth-Jones, "The 're-identification' of governor william weld's medical information: A critical re-examination of health data identification risks and privacy protections, then and now," *SSRN Electronic Journal*, 06 2012.

[3] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," *Proceedings - IEEE Symposium on Security and Privacy*, 04 2009.

[4] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, p. 211–407, Aug. 2014.

[5] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," in *Theory of Cryptography* (M. Hirt and A. Smith, eds.), (Berlin, Heidelberg), pp. 635–658, Springer Berlin Heidelberg, 2016.

[6] I. Mironov, "Rényi differential privacy," *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, Aug 2017.

[7] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella Béguelin, "Probabilistic relational reasoning for differential privacy," in *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, (New York, NY, USA), p. 97–110, Association for Computing Machinery, 2012.

[8] G. Barthe, B. Köpf, F. Olmedo, and S. Zanella-Béguelin, "Probabilistic relational reasoning for differential privacy," *ACM Trans. Program. Lang. Syst.*, vol. 35, Nov. 2013.

[9] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P.-Y. Strub, "Proving differential privacy via probabilistic couplings," in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, (New York, NY, USA), p. 749–758, Association for Computing Machinery, 2016.

[10] T. Sato, G. Barthe, M. Gaboardi, J. Hsu, and S.-y. Katsumata, "Approximate span liftings: Compositional semantics for relaxations of differential privacy," in *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–14, 2019.

[11] J. Reed and B. C. Pierce, "Distance makes the types grow stronger: A calculus for differential privacy," in *Proceedings of the 15th ACM SIGPLAN Conference on Functional Programming (ICFP 2010)*, (New York, NY, USA), p. 157–168, Association for Computing Machinery, Sept. 2010.

[12] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce, "Linear depen-

dent types for differential privacy," in *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, (New York, NY, USA), p. 357–370, Association for Computing Machinery, 2013.

[13] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, A. Roth, and P.-Y. Strub, "Higher-order approximate relational refinement types for mechanism design and differential privacy," *SIGPLAN Not.*, vol. 50, p. 55–68, Jan. 2015.

[14] J. P. Near, D. Darais, C. Abuah, T. Stevens, P. Gaddamadugu, L. Wang, N. Somani, M. Zhang, N. Sharma, A. Shan, and D. Song, "Duet: An expressive higher-order language and linear type system for statically enforcing differential privacy," *Proc. ACM Program. Lang.*, vol. 3, Oct. 2019.

[15] M. Toro, D. Darais, C. Abuah, J. Near, F. Olmedo, and Éric Tanter, "Contextual linear types for differential privacy," 2020.

[16] E. Meijer and P. Drayton, "Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages," 01 2004.

[17] K. E. Gray, R. B. Findler, and M. Flatt, "Fine-grained interoperability through mirrors and contracts," in *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, (New York, NY, USA), p. 231–245, Association for Computing Machinery, 2005.

[18] G. L. Steele, "An overview of common lisp," in *Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, LFP '82, (New York, NY, USA), p. 98–107, Association for Computing Machinery, 1982.

[19] N. Feinberg, S. Keene, R. O. Mathews, and P. Withington, "Dylan programming: an object-oriented and dynamic language," 1996.

[20] J. Siek and W. Taha, "Gradual typing for functional languages," in *Proceedings of the Scheme and Functional Programming Workshop*, pp. 81–92, Sept. 2006.

[21] R. Garcia, A. M. Clark, and É. Tanter, "Abstracting gradual typing," in *Proceedings of the 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2016)* (R. Bodík and R. Majumdar, eds.), (St Petersburg, FL, USA), pp. 429–442, ACM Press, Jan. 2016. See erratum: https://www.cs.ubc.ca/ rxg/agt-erratum.pdf.

[22] J. Siek and W. Taha, "Gradual typing for objects," in *Proceedings of the 21st European Conference on Object-oriented Programming (ECOOP 2007)* (E. Ernst, ed.), no. 4609 in Lecture Notes in Computer Science, (Berlin, Germany), pp. 2–27, Springer-Verlag, July 2007.

[23] D. Herman, A. Tomb, and C. Flanagan, "Space-efficient gradual typing," *Higher-Order and Sympolic Computation*, vol. 23, pp. 167–189, June 2010.

[24] J. G. Siek, M. M. Vitousek, M. Cimini, S. Tobin-Hochstadt, and R. Garcia, "Monotonic references for efficient gradual typing," in *Proceedings of the 24th European Symposium on Programming Languages and Systems (ESOP 2015)* (J. Vitek, ed.), vol. 9032 of *Lecture Notes in Computer Science*, (London, UK), pp. 432–456, Springer-Verlag, Mar. 2015.

[25] F. Bañados Schwerter, R. Garcia, and É. Tanter, "A theory of gradual effect systems," in *Proceedings of the 19th ACM SIGPLAN Conference on Functional Programming (ICFP*

2014), (Gothenburg, Sweden), pp. 283–295, ACM Press, Sept. 2014.

[26] I. Sergey and D. Clarke, "Gradual ownership types," in *Proceedings of the 21st European Symposium on Programming Languages and Systems (ESOP 2012)* (H. Seidl, ed.), vol. 7211 of *Lecture Notes in Computer Science*, (Tallinn, Estonia), pp. 579–599, Springer-Verlag, 2012.

[27] T. Disney and C. Flanagan, "Gradual information flow typing," in *International Workshop on Scripts to Programs*, 2011.

[28] L. Fennell and P. Thiemann, "Gradual security typing with references," in *Proceedings of the 26th Computer Security Foundations Symposium (CSF)*, pp. 224–239, June 2013.

[29] N. Lehmann and É. Tanter, "Gradual refinement types," in *Proceedings of the 44th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2017)*, (Paris, France), pp. 775–788, ACM Press, Jan. 2017.

[30] A. Ahmed, R. B. Findler, J. G. Siek, and P. Wadler, "Blame for all," in *Proceedings of the 38th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2011)*, (Austin, Texas, USA), pp. 201–214, ACM Press, Jan. 2011.

[31] A. Ahmed, D. Jamner, J. G. Siek, and P. Wadler, "Theorems for free for free: Parametricity, with and without types," *Proceedings of the ACM on Programming Languages*, vol. 1, pp. 39:1–39:28, Sept. 2017.

[32] Y. Igarashi, T. Sekiyama, and A. Igarashi, "On polymorphic gradual typing," *Proceedings of the ACM on Programming Languages*, vol. 1, pp. 40:1–40:29, Sept. 2017.

[33] L. Ina and A. Igarashi, "Gradual typing for generics," in *Proceedings of the 26th ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2011)*, (Portland, Oregon, USA), pp. 609–624, ACM Press, Oct. 2011.

[34] N. Xie, X. Bi, and B. C. d. S. Oliveira, "Consistent subtyping for all," in *Proceedings of the 27th European Symposium on Programming Languages and Systems (ESOP 2018)* (A. Ahmed, ed.), vol. 10801 of *Lecture Notes in Computer Science*, (Thessaloniki, Greece), pp. 3–30, Springer-Verlag, Apr. 2018.

[35] J. G. Siek, M. M. Vitousek, M. Cimini, and J. T. Boyland, "Refined criteria for gradual typing," in *1st Summit on Advances in Programming Languages (SNAPL 2015)*, vol. 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Asilomar, California, USA), pp. 274–293, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, May 2015.

[36] M. Toro, R. Garcia, and É. Tanter, "Type-driven gradual security with references," *ACM Transactions on Programming Languages and Systems*, vol. 40, pp. 16:1–16:55, Nov. 2018.

[37] M. Toro, E. Labrada, and É. Tanter, "Gradual parametricity, revisited (with appendix)," 2018. arXiv:1807.04596 [cs.PL].

[38] M. R. Clarkson and F. B. Schneider, "Hyperproperties," vol. 18, p. 1157–1210, Sept. 2010.

[39] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Conference Record*

of the 4th ACM Symposium on Principles of Programming Languages (POPL 77), (Los Angeles, CA, USA), pp. 238–252, ACM Press, Jan. 1977.

[40] W. A. Howard, "The formulae-as-types notion of construction," in *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism* (H. Curry, H. B., S. J. Roger, and P. Jonathan, eds.), Academic Press, 1980.

[41] M. Felleisen, R. B. Findler, and M. Flatt, *Semantics Engineering with PLT Redex.* The MIT Press, 1st ed., 2009.

[42] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy, "Explicit substitutions," in *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, (New York, NY, USA), p. 31–46, Association for Computing Machinery, 1989.

[43] B. C. Pierce, *Types and programming languages.* Cambridge, MA, USA: MIT Press, 2002.

[44] W. W. Tait, "Intensional interpretations of functionals of finite type i," *Journal of Symbolic Logic*, vol. 32, no. 2, p. 198–212, 1967.

[45] J. A. Goguen and J. Meseguer, "Security policies and security models," in *1982 IEEE Symposium on Security and Privacy*, pp. 11–11, 1982.

[46] J. C. Reynolds, "Towards a theory of type structure," in *Programming Symposium* (B. Robinet, ed.), (Berlin, Heidelberg), pp. 408–425, Springer Berlin Heidelberg, 1974.

[47] A. Church, "A formulation of the simple theory of types," *J. Symb. Log.*, vol. 5, pp. 56–68, 1940.

[48] M. Toro, E. Labrada, and É. Tanter, "Gradual parametricity, revisited," *Proceedings of the ACM on Programming Languages*, vol. 3, pp. 17:1–17:30, Jan. 2019.

Appendix A

Auxiliary definitions

A.1. A Static Sensitivity Type System

This section present auxiliary functions and operators definitions that were not included in the Chapter § 3.

$\boxed{[\Sigma/x]\Sigma}$ **Sensitivity environment substitution**

$$\begin{aligned}
[\Sigma_1/x]\varnothing &= \varnothing \\
[\Sigma_1/x](\Sigma + sy) &= [\Sigma_1/x]\Sigma + sy \\
[\Sigma_1/x](\Sigma + sx) &= [\Sigma_1/x]\Sigma + s\Sigma_1
\end{aligned}$$

$\boxed{[\Sigma/x]\tau}$ **Sensitivity environment substitution (on types)**

$$\begin{aligned}
[\Sigma/x]\mathbb{R} &= \mathbb{R} \\
[\Sigma/x]\mathsf{unit} &= \mathsf{unit} \\
[\Sigma/x]((y : \tau_1) \xrightarrow{\Sigma_2} \tau_2) &= (y : [\Sigma/x]\tau_1) \xrightarrow{[\Sigma/x]\Sigma_2} [\Sigma/x]\tau_2 \\
[\Sigma/x](\tau_1 \,{}^{\Sigma_1}\!\oplus^{\Sigma_2}\, \tau_2) &= [\Sigma/x]\tau_1 \,{}^{[\Sigma/x]\Sigma_1}\!\oplus^{[\Sigma/x]\Sigma_2}\, [\Sigma/x]\tau_2
\end{aligned}$$

Figure A.1: Sensitivity environment substitutions

$\boxed{\Gamma \vdash e : \mathrm{T}}$ **Well-typed expressions**

$$(\textsc{TClosure})\ \frac{\exists \Gamma' : \gamma' \vdash \Gamma' \qquad \Gamma' \vdash e : \mathrm{T}}{\Gamma \vdash \langle e, \gamma' \rangle : \mathrm{T}} \qquad\qquad (\textsc{Tctx})\ \frac{\exists \Gamma' : \gamma' \vdash \Gamma' \qquad \Gamma' \vdash e : \mathrm{T}}{\Gamma \vdash \mathsf{ctx}(\gamma', e) : \mathrm{T}}$$

Figure A.2: Typing rules for closures and contexts

Definition 37. *A type environment is well-formed with respect to a substitution, denoted* $\gamma \vdash \Gamma$*, if and only if:*

1. *$dom(\gamma) \subseteq dom(\Gamma)$, and*
2. *$\forall x_i \in dom(\gamma).\ \cdot \vdash \gamma(x_i) : \Gamma(x_i)$.*

A.2. A Gradual Sensitivity Type System

$\boxed{[\Xi/x]\Xi}$ **Gradual sensitivity environment substitution**

$$\begin{aligned}
[\Xi_1/x]\varnothing &= \varnothing \\
[\Xi_1/x](\Xi + iy) &= [\Xi_1/x]\Xi + iy \\
[\Xi_1/x](\Xi + ix) &= [\Xi_1/x]\Xi + i\Xi_1
\end{aligned}$$

$\boxed{[\Xi/x]\tau}$ **Gradual sensitivity environment substitution (on gradual types)**

$$\begin{aligned}
[\Xi/x]\mathbb{R} &= \mathbb{R} \\
[\Xi/x]\mathsf{unit} &= \mathsf{unit} \\
[\Xi/x]((y : g_1) \xrightarrow{\Xi_2} g_2) &= (y : [\Xi/x]g_1) \xrightarrow{[\Xi/x]\Xi_2} [\Xi/x]g_2 \\
[\Xi/x](g_1 \ {}^{\Xi_1}{\oplus}^{\Xi_2}\ g_2) &= [\Xi/x]g_1 \ {}^{[\Xi/x]\Xi_1}{\oplus}^{[\Xi/x]\Xi_2}\ [\Xi/x]g_2
\end{aligned}$$

Figure A.3: Gradual sensitivity environment substitutions

$\boxed{\Gamma \vdash t^G : G}$ **Well-typed expressions**

$$(\text{I}G\textsc{closure})\ \frac{t \in \mathbb{T}[G] \qquad t \vdash \gamma'}{\langle t, \gamma' \rangle \in \mathbb{T}[G]} \qquad\qquad (\text{I}G\textsc{ctx})\ \frac{e \in \mathbb{T}[G] \qquad t \vdash \gamma'}{\mathsf{ctx}(\gamma', t) \in \mathbb{T}[G]}$$

Figure A.4: Typing rules for gradual closures and contexts

Appendix B

Properties of a Gradual Sensitivity Type System

B.1. Preamble

This section presents necessary definitions and propositions in order to prove the propositions in the following sections.

The precision relation for intrinsic terms is defined modulo α-renaming for pairs of constructs like lambdas or case terms that introduce fresh variables not necessarily with the same name.

Definition 38 (Expressions precision)**.**

$$(G{\sqsubseteq_{\mathbb{R}}})\ \frac{}{r \sqsubseteq r} \qquad (G{\sqsubseteq_{+}})\ \frac{e_{11} \sqsubseteq e_{21} \qquad e_{12} \sqsubseteq e_{22}}{e_{11} + e_{12} \sqsubseteq e_{21} + e_{22}} \qquad (G{\sqsubseteq_{\leq}})\ \frac{e_{11} \sqsubseteq e_{21} \qquad e_{12} \sqsubseteq e_{22}}{e_{11} \leq e_{12} \sqsubseteq e_{21} \leq e_{22}}$$

$$(G{\sqsubseteq_{x}})\ \frac{}{x \sqsubseteq x} \qquad (G{\sqsubseteq_{\lambda}})\ \frac{g_{11} \sqsubseteq g_{21} \qquad e_1 \sqsubseteq e_2}{\lambda(x : g_{11}).e_1 \sqsubseteq \lambda(x : g_{21}).e_2} \qquad (G{\sqsubseteq_{@}})\ \frac{e_{11} \sqsubseteq e_{21} \qquad e_{12} \sqsubseteq e_{22}}{e_{11}\ e_{12} \sqsubseteq e_{21}\ e_{22}}$$

$$(G{\sqsubseteq_{\mathsf{unit}}})\ \frac{}{\mathsf{tt} \sqsubseteq \mathsf{tt}} \qquad (G{\sqsubseteq_{\mathsf{inl}}})\ \frac{g_{12} \sqsubseteq g_{22} \qquad e_{11} \sqsubseteq e_{21}}{\mathsf{inl}^{g_{12}} e_{11} \sqsubseteq \mathsf{inl}^{g_{22}} e_{21}} \qquad (G{\sqsubseteq_{\mathsf{inr}}})\ \frac{g_{11} \sqsubseteq g_{21} \qquad e_{12} \sqsubseteq e_{22}}{\mathsf{inr}^{g_{11}} e_{12} \sqsubseteq \mathsf{inr}^{g_{21}} e_{22}}$$

$$(G{\sqsubseteq_{\mathsf{case}}})\ \frac{e_{11} \sqsubseteq e_{21} \qquad e_{12} \sqsubseteq e_{22} \qquad e_{13} \sqsubseteq e_{23}}{\mathsf{case}\ e_{11}\ \mathsf{of}\ \{\, x \Rightarrow e_{12} \,\}\ \{\, y \Rightarrow e_{13} \,\} \sqsubseteq \mathsf{case}\ e_{21}\ \mathsf{of}\ \{\, x \Rightarrow e_{22} \,\}\ \{\, y \Rightarrow e_{23} \,\}}$$

$$(G{\sqsubseteq_{::}})\ \frac{e_{11} \sqsubseteq e_{21} \qquad G_{12} \sqsubseteq G_{22}}{e_{11} :: G_{12} \sqsubseteq e_{21} :: G_{22}} \qquad (G{\sqsubseteq_{\langle\lambda,\gamma\rangle}})\ \frac{\gamma_1 \sqsubseteq \gamma_2 \qquad e_1 \sqsubseteq e_2}{\langle e_1, \gamma_1 \rangle \sqsubseteq \langle e_2, \gamma_2 \rangle}$$

$$(G{\sqsubseteq_{\mathsf{ctx}}})\ \frac{\gamma_1 \sqsubseteq \gamma_2 \qquad e_1 \sqsubseteq e_2}{\mathsf{ctx}(\gamma_1, e_1) \sqsubseteq \mathsf{ctx}(\gamma_2, e_2)}$$

Definition 39 (Substitutions precision).

$$(IG\sqsubseteq_{\varnothing})\ \frac{}{\varnothing \sqsubseteq \varnothing} \qquad (IG\sqsubseteq_{\gamma})\ \frac{\gamma_1 \sqsubseteq \gamma_2 \quad v_1 \sqsubseteq v_2}{\gamma_1[x \mapsto v_1] \sqsubseteq \gamma_2[x \mapsto v_2]}$$

Definition 40 (Terms precision).

$$(IG\sqsubseteq_{\mathbb{R}})\ \frac{}{r^{\mathbb{R};\varnothing} \sqsubseteq r^{\mathbb{R};\varnothing}} \qquad (IG\sqsubseteq_{+})\ \frac{t_{11} \sqsubseteq t_{21} \quad t_{12} \sqsubseteq t_{22}}{t_{11} + t_{12} \sqsubseteq t_{21} + t_{22}}$$

$$(IG\sqsubseteq_{\leq})\ \frac{t_{11} \sqsubseteq t_{21} \quad t_{12} \sqsubseteq t_{22}}{t_{11} \leq t_{12} \sqsubseteq t_{21} \leq t_{22}} \qquad (IG\sqsubseteq_{x})\ \frac{G_1 \sqsubseteq G_2}{x^{G_1} \sqsubseteq x^{G_2}}$$

$$(IG\sqsubseteq_{\lambda})\ \frac{g_{11} \sqsubseteq g_{21} \quad t^{G_{12}} \sqsubseteq t^{G_{22}}}{\lambda(x : g_{11}).t^{G_{12}} \sqsubseteq \lambda(x : g_{21}).t^{G_{22}}} \qquad (IG\sqsubseteq_{@})\ \frac{t_{11} \sqsubseteq t_{21} \quad t_{12} \sqsubseteq t_{22}}{t_{11}\ t_{12} \sqsubseteq t_{21}\ t_{22}}$$

$$(IG\sqsubseteq_{\mathsf{unit}})\ \frac{}{\mathsf{tt} \sqsubseteq \mathsf{tt}} \qquad (IG\sqsubseteq_{\mathsf{inl}})\ \frac{g_{12} \sqsubseteq g_{22} \quad t_{11} \sqsubseteq t_{21}}{\mathsf{inl}^{g_{12}} t_{11} \sqsubseteq \mathsf{inl}^{g_{22}} t_{21}} \qquad (IG\sqsubseteq_{\mathsf{inr}})\ \frac{g_{11} \sqsubseteq g_{21} \quad t_{12} \sqsubseteq t_{22}}{\mathsf{inr}^{g_{11}} t_{12} \sqsubseteq \mathsf{inr}^{g_{21}} t_{22}}$$

$$(IG\sqsubseteq_{\mathsf{case}})\ \frac{t_{11} \sqsubseteq t_{21} \quad t_{12} \sqsubseteq t_{22} \quad t_{13} \sqsubseteq t_{23}}{\mathsf{case}\ t_{11}\ \mathsf{of}\,\{\,x \Rightarrow t_{12}\,\}\ \{\,y \Rightarrow t_{13}\,\} \sqsubseteq \mathsf{case}\ t_{21}\ \mathsf{of}\,\{\,x \Rightarrow t_{22}\,\}\ \{\,y \Rightarrow t_{23}\,\}}$$

$$(IG\sqsubseteq_{::})\ \frac{\varepsilon_1 \sqsubseteq \varepsilon_2 \quad t_{11} \sqsubseteq t_{21} \quad G_{12} \sqsubseteq G_{22}}{\varepsilon_1 t_{11} :: G_{12} \sqsubseteq \varepsilon_2 t_{21} :: G_{22}} \qquad (IG\sqsubseteq_{\langle\lambda,\gamma\rangle})\ \frac{\gamma_1 \sqsubseteq \gamma_2 \quad t_1 \sqsubseteq t_2}{\langle t_1, \gamma_1\rangle \sqsubseteq \langle t_2, \gamma_2\rangle}$$

$$(IG\sqsubseteq_{\mathsf{ctx}})\ \frac{\gamma_1 \sqsubseteq \gamma_2 \quad t_1 \sqsubseteq t_2}{\mathsf{ctx}(\gamma_1, t_1) \sqsubseteq \mathsf{ctx}(\gamma_2, t_2)}$$

Definition 41 (Type environments precision).

$$(G\sqsubseteq_{\cdot})\ \frac{}{\cdot \sqsubseteq \cdot} \qquad (G\sqsubseteq_{\Gamma})\ \frac{\Gamma_1 \sqsubseteq \Gamma_2 \quad G_1 \sqsubseteq G_2}{\Gamma_1, x : G_1 \sqsubseteq \Gamma_2, x : G_2}$$

Proposition 34. *If $i_1 \sqsubseteq i_2$ and $i_3 \sqsubseteq i_4$ then $i_1 * i_3 \sqsubseteq i_2 * i_4$*

Proof. Follows directly from the Propositions 15 and 3. ☐

Proposition 35. *If $i_1 \sqsubseteq i_2$ and $\Xi_1 \sqsubseteq \Xi_2$ then $i_1\Xi_1 \sqsubseteq i_2\Xi_2$*

Proof. Let $\Xi'_1 = i_1\Xi_1$, $\Xi'_2 = i_2\Xi_2$. Then, for any x, $\Xi'_1(x) = i_1 * \Xi_1(x)$ and $\Xi'_2(x) = i_2 * \Xi_2(x)$. Finally, by Proposition 34, $\forall x, \Xi'_1(x) \sqsubseteq \Xi'_2(x)$, which is equivalent to $\Xi'_1 \sqsubseteq \Xi'_2$. ☐

Proposition 36. *If $i_1 \sqsubseteq i_2$ and $i_3 \sqsubseteq i_4$ then $i_1 + i_3 \sqsubseteq i_2 + i_4$.*

Proof. Follows directly from the Propositions 13 and 3. ☐

Proposition 37. *If $\Xi_1 \sqsubseteq \Xi_2$ and $\Xi_3 \sqsubseteq \Xi_4$ then $\Xi_1 + \Xi_3 \sqsubseteq \Xi_2 + \Xi_4$.*

Proof. Follows directly by Propositions 4 and 36. ☐

Proposition 38. *If $i_1 \sqsubseteq i_2$ and $i_3 \sqsubseteq i_4$, then $i_1 \curlyvee i_3 \sqsubseteq i_2 \curlyvee i_4$*

Proof. Let $i_i = [s_{i1}, s_{i2}]$. We know that

1. $s_{11} \geq s_{21}$ and $s_{12} \leq s_{22}$.
2. $s_{31} \geq s_{41}$ and $s_{32} \leq s_{42}$.
3. $\max(s_{11}, s_{31}) \geq \max(s_{21}, s_{41})$ and $\max(s_{12}, s_{32}) \leq \max(s_{22}, s_{42})$.
4. $i_1 \curlyvee i_3 = [\max(s_{11}, s_{31}), \max(s_{12}, s_{32})]$ and $i_2 \curlyvee i_4 = [\max(s_{21}, s_{41}), \max(s_{22}, s_{42})]$.

Finally, $i_1 \curlyvee i_3 \sqsubseteq i_2 \curlyvee i_4$. ☐

Lemma 39. *If $\Xi_1 \sqsubseteq \Xi_2$ and $\Xi_3 \sqsubseteq \Xi_4$, then $\Xi_1 \curlyvee \Xi_3 \sqsubseteq \Xi_2 \curlyvee \Xi_4$*

Proof. Let $x \in dom(\Xi_1 \curlyvee \Xi_3)$. We know that

1. By Proposition 4, $\Xi_1(x) \sqsubseteq \Xi_2(x)$ and $\Xi_3(x) \sqsubseteq \Xi_4(x)$.
2. By Proposition 38, $\Xi_1(x) \curlyvee \Xi_3(x) \sqsubseteq \Xi_2(x) \curlyvee \Xi_4(x)$.

Finally, by Proposition 4, $\Xi_1 \curlyvee \Xi_3 \sqsubseteq \Xi_2 \curlyvee \Xi_4$. ☐

Proposition 40. *If $g_1 \sqsubseteq g_2$ and $g_3 \sqsubseteq g_4$, then $g_1 \curlyvee g_3 \sqsubseteq g_2 \curlyvee g_4$*

Proof. By induction on the definition of $g_1 \curlyvee g_3$. All cases follow trivially (no premises) or by inductive hypotheses and Proposition 39. The function type case follows by induction hypotheses, Propositions 39 and 43. ☐

Proposition 41. *If $i_1 \sqsubseteq i_2$ and $i_3 \sqsubseteq i_4$ and $i_1 \curlywedge i_3$ is defined, then $i_1 \curlywedge i_3 \sqsubseteq i_2 \curlywedge i_4$.*

Proof. Let $i_i = [s_{i1}, s_{i2}]$. We know that

1. $s_{11} \geq s_{21}$ and $s_{12} \leq s_{22}$.
2. $s_{31} \geq s_{41}$ and $s_{32} \leq s_{42}$.
3. $\min(s_{11}, s_{31}) \geq \min(s_{21}, s_{41})$ and $\min(s_{12}, s_{32}) \leq \min(s_{22}, s_{42})$.
4. $i_1 \curlywedge i_3 = [\min(s_{11}, s_{31}), \min(s_{12}, s_{32})]$ and $i_2 \curlywedge i_4 = [\min(s_{21}, s_{41}), \min(s_{22}, s_{42})]$.

Finally, $i_1 \curlywedge i_3 \sqsubseteq i_2 \curlywedge i_4$. ☐

Proposition 42. *If $\Xi_1 \sqsubseteq \Xi_2$ and $\Xi_3 \sqsubseteq \Xi_4$ and $\Xi_1 \curlywedge \Xi_3$ is defined, then $\Xi_1 \curlywedge \Xi_3 \sqsubseteq \Xi_2 \curlywedge \Xi_4$*

Proof. Follows directly from the Propositions 23 and 41. □

Proposition 43. *If $g_1 \sqsubseteq g_2$ and $g_3 \sqsubseteq g_4$ and $g_1 \curlywedge g_3$ is defined, then $g_1 \curlywedge g_3 \sqsubseteq g_2 \curlywedge g_4$*

Proof. By induction on the definition of $g_1 \curlywedge g_3$. All cases follow trivially (no premises) or by inductive hypotheses and Proposition 42. The function case follow by induction hypotheses, and Propositions 42 and 40. □

Proposition 44. *If $G_1 \sqsubseteq G_2$ and $G_3 \sqsubseteq G_4$ and $G_1 \curlywedge G_3$ is defined, then $G_1 \curlywedge G_3 \sqsubseteq G_2 \curlywedge G_4$*

Proof. By Propositions 43 and 42. □

Proposition 45. *If $\Xi_1 \sqsubseteq \Xi_2$ and $\Xi_3 \sqsubseteq \Xi_4$ then $[\Xi_3/x]\Xi_1 \sqsubseteq [\Xi_4/x]\Xi_2$.*

Proof. By induction on the definition of substitution.

Case $\Xi_1 = \varnothing$ and $\Xi_1 = \varnothing$. Trivial.

Case $\Xi_1 = \Xi_1' + i_1 y$ and $\Xi_2 = \Xi_2' + i_2 y$, where $y \neq x$. By induction hypothesis.

Case $\Xi_1 = \Xi_1' + i_1 y$ and $\Xi_2 = \Xi_2' + i_2 y$, where $y = x$. We know that $\Xi_1' \sqsubseteq \Xi_2'$, $i_1 \sqsubseteq i_2$, $[\Xi_3/x]\Xi_1 = \Xi_1' + i_1 * \Xi_3$, $[\Xi_4/x]\Xi_2 = \Xi_2' + i_2 * \Xi_4$. By Proposition 35 and 45, $[\Xi_3/x]\Xi_1 \sqsubseteq [\Xi_4/x]\Xi_2 = \Xi_2' + i_2 * \Xi_4$.

□

Proposition 46. *If $g_1 \sqsubseteq g_2$ and $\Xi_1 \sqsubseteq \Xi_2$ then $[\Xi_1/x]g_1 \sqsubseteq [\Xi_2/x]g_2$.*

Proof. By induction on the type constructors:

Case $\{\mathbb{R}, \mathsf{unit}\}$: $[\Xi_1/x]g_1 = g_1$ and $[\Xi_2/x]g_2 = g_2$.

Case $g_1 = (x : g_{11}) \xrightarrow{\Xi_{11}} g_{12}$ and $g_2 = (x : g_{21}) \xrightarrow{\Xi_{21}} g_{22}$. By induction hypotheses and Proposition 45.

Case $g_1 = g_{11} \;{}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\; g_{12}$ and $g_2 = g_{21} \;{}^{\Xi_{21}}\!\oplus^{\Xi_{22}}\; g_{22}$. By induction hypotheses and Proposition 45.

□

Proposition 47. *If $G_1 \sqsubseteq G_2$ and $\Xi_1 \sqsubseteq \Xi_2$ then $[\Xi_1/x]G_1 \sqsubseteq [\Xi_2/x]G_2$.*

Proof. By Propositions 45 and 46. □

Proposition 48. *If $\varepsilon_1 \sqsubseteq \varepsilon_2$ and $\Xi_1 \sqsubseteq \Xi_2$ then $[\Xi_1/x]\varepsilon_1 \sqsubseteq [\Xi_2/x]\varepsilon_2$.*

Proof. Follows directly from using the Propositions 46 and 45 point-wisely. $\square$

Proposition 49. *If $t_1 \sqsubseteq t_2$ and $\Xi_1 \sqsubseteq \Xi_2$ then $[\Xi_1/x]t_1 \sqsubseteq [\Xi_2/x]t_2$.*

Proof. By induction on the definition of $[\Xi_1/x]t_1$. The variable case follows trivially by using the Propositions 46 and 45 point-wisely. All other cases follow directly from the induction hypotheses. $\square$

Proposition 50. *If $\varepsilon_1 \sqsubseteq \varepsilon_2$ then $idom(\varepsilon_1) \sqsubseteq idom(\varepsilon_2)$.*

Proof. Follows directly from the hypothesis and the precision relation between arrow types. $\square$

Proposition 51. *If $\varepsilon_1 \sqsubseteq \varepsilon_2$ then $icod(\varepsilon_1) \sqsubseteq icod(\varepsilon_2)$.*

Proof. Follows directly from the hypothesis and the precision relation between arrow types. $\square$

Proposition 52. *If $\varepsilon_1 \sqsubseteq \varepsilon_2$ then $ileft(\varepsilon_1) \sqsubseteq ileft(\varepsilon_2)$.*

Proof. Follows directly from the hypothesis and the precision relation between sum types. $\square$

Proposition 53. *If $\varepsilon_1 \sqsubseteq \varepsilon_2$ then $iright(\varepsilon_1) \sqsubseteq iright(\varepsilon_2)$.*

Proof. Follows directly from the hypothesis and the precision relation between sum types. $\square$

Proposition 54. *If $\varepsilon_1 \sqsubseteq \varepsilon_2$ and $\varepsilon_3 \sqsubseteq \varepsilon_4$ then $\varepsilon_1 +_\Xi \varepsilon_3 \sqsubseteq \varepsilon_2 +_\Xi \varepsilon_4$.*

Proof. Let $\varepsilon_i = \langle g_{i1}; \Xi_{i1}, g_{i2}; \Xi_{i2} \rangle$.

$\varepsilon_1 +_\Xi \varepsilon_3 = \langle g_{11}; \Xi_{11} + \Xi_{31}, \Xi_{12}; \Xi_{12} + \Xi_{32} \rangle$ and $\varepsilon_2 +_\Xi \varepsilon_4 = \langle g_{21}; \Xi_{21} + \Xi_{41}, \Xi_{22}; \Xi_{22} + \Xi_{42} \rangle$.

By Proposition 37, $\Xi_{11} + \Xi_{31} \sqsubseteq \Xi_{12} + \Xi_{32}$ and $\Xi_{21} + \Xi_{41} \sqsubseteq \Xi_{22} + \Xi_{42}$. Finally, $\varepsilon_1 +_\Xi \varepsilon_3 \sqsubseteq \varepsilon_2 +_\Xi \varepsilon_4$. $\square$

Proposition 55. *If $\varepsilon_1 \sqsubseteq \varepsilon_2$ and $\varepsilon_3 \sqsubseteq \varepsilon_4$ then $\varepsilon_1 \curlyvee_\Xi \varepsilon_3 \sqsubseteq \varepsilon_2 \curlyvee_\Xi \varepsilon_4$.*

Proof. Analogous to the proof for Proposition 54 but using Proposition 39. $\square$

Proposition 56. *If $i_1 \sqsubseteq i_2$ and $i_3 \sqsubseteq i_4$ then $\mathcal{I}_{<:}(i_1, i_3) \sqsubseteq \mathcal{I}_{<:}(i_2, i_4)$.*

Proof. Let $i_i = [s_{i1}, s_{i2}]$. We know that

1. $s_{21} \leq s_{11}$, $s_{12} \leq s_{22}$, $s_{41} \leq s_{31}$, $s_{32} \leq s_{42}$.

2. $s_{11} \leq \min(s_{12}, s_{32})$ and $\max(s_{11}, s_{31}) \leq s_{32}$.

3. $s_{21} \leq s_{11} \leq \min(s_{12}, s_{32}) \leq \min(s_{22}, s_{42})$.

4. $\max(s_{21}, s_{41}) \leq \max(s_{11}, s_{31}) \leq s_{32} \leq s_{42}$

5. $\mathcal{I}_{<:}(i_2, i_4) = \langle\, [s_{21}, \min(s_{22}, s_{42})], [\max(s_{21}, s_{41}), s_{42}] \,\rangle$ is defined and $\mathcal{I}_{<:}(i_1, i_3) \sqsubseteq \mathcal{I}_{<:}(i_2, i_4)$.

And the result holds. □

Proposition 57. *If* $\Xi_1 \sqsubseteq \Xi_2$ *and* $\Xi_3 \sqsubseteq \Xi_4$ *then* $\mathcal{I}_{<:}(\Xi_1, \Xi_3) \sqsubseteq \mathcal{I}_{<:}(\Xi_2, \Xi_4)$.

Proof. By induction on the definition of $\mathcal{I}_{<:}(\Xi_1, \Xi_3)$ and Proposition 56. □

Proposition 58. *If* $g_1 \sqsubseteq g_2$ *and* $g_3 \sqsubseteq g_4$ *then* $\mathcal{I}_{<:}(g_1, g_3) \sqsubseteq \mathcal{I}_{<:}(g_2, g_4)$.

Proof. By induction on the definition of $\mathcal{I}_{<:}(g_1, g_3)$.

Case $g \in \{\, \mathbb{R}, \mathsf{unit} \,\}$. Trivial.

Case $g_i = (x : g_{i1}) \xrightarrow{\Xi_{i2}} g_{i2}$. We know that

$$\frac{g_{11} \sqsubseteq g_{31} \qquad \Xi_{12} \sqsubseteq \Xi_{32} \qquad g_{12} \sqsubseteq g_{32}}{(x : g_{11}) \xrightarrow{\Xi_{12}} g_{12} \sqsubseteq (x : g_{31}) \xrightarrow{\Xi_{32}} g_{32}} \qquad \frac{g_{21} \sqsubseteq g_{41} \qquad \Xi_{22} \sqsubseteq \Xi_{42} \qquad g_{22} \sqsubseteq g_{42}}{(x : g_{21}) \xrightarrow{\Xi_{22}} g_{22} \sqsubseteq (x : g_{41}) \xrightarrow{\Xi_{42}} g_{42}}$$

$$\frac{\begin{array}{c}\mathcal{I}_{<:}(g_{11}, g_{31}) = \langle\, g'_{11}, g'_{31} \,\rangle \\ \mathcal{I}_{<:}(\Xi_{12}, \Xi_{32}) = \langle\, \Xi'_{12}, \Xi'_{32} \,\rangle \qquad \mathcal{I}_{<:}(g_{12}, g_{32}) = \langle\, g'_{12}, g'_{32} \,\rangle\end{array}}{\mathcal{I}_{<:}((x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}, (x : g_{31}) \xrightarrow{\Xi_{32}} g_{32}) = \langle\, (x : g'_{11}) \xrightarrow{\Xi'_{12}} g'_{12}, (x : g'_{31}) \xrightarrow{\Xi'_{32}} g'_{32} \,\rangle}$$

$$\frac{\begin{array}{c}\mathcal{I}_{<:}(g_{21}, g_{41}) = \langle\, g'_{21}, g'_{41} \,\rangle \\ \mathcal{I}_{<:}(\Xi_{22}, \Xi_{42}) = \langle\, \Xi'_{22}, \Xi'_{42} \,\rangle \qquad \mathcal{I}_{<:}(g_{22}, g_{42}) = \langle\, g'_{22}, g'_{42} \,\rangle\end{array}}{\mathcal{I}_{<:}((x : g_{21}) \xrightarrow{\Xi_{22}} g_{22}, (x : g_{41}) \xrightarrow{\Xi_{42}} g_{42}) = \langle\, (x : g'_{21}) \xrightarrow{\Xi'_{22}} g'_{22}, (x : g'_{41}) \xrightarrow{\Xi'_{42}} g'_{42} \,\rangle}$$

- By induction hypotheses, $\langle\, g'_{2j}, g'_{4j} \,\rangle$ are defined and $\langle\, g'_{1j}, g'_{3j} \,\rangle \sqsubseteq \langle\, g'_{2j}, g'_{4j} \,\rangle$.
- By Proposition 57, $\langle\, \Xi'_{22}, \Xi'_{42} \,\rangle$ are defined and $\langle\, \Xi'_{12}, \Xi'_{32} \,\rangle \sqsubseteq \langle\, \Xi'_{22}, \Xi'_{42} \,\rangle$.

Finally, $\mathcal{I}_{<:}(g_2, g_4)$ is defined and $\mathcal{I}_{<:}(g_1, g_3) \sqsubseteq \mathcal{I}_{<:}(g_2, g_4)$.

Case $g_i = g_{i1} \;{}^{\Xi_{i1}}{\oplus}^{\Xi_{i2}}\; g_{i2}$. Analogous to arrow types.

□

Proposition 59. *If* $G_1 \sqsubseteq G_2$ *and* $G_3 \sqsubseteq G_4$ *then* $\mathcal{I}_{<:}(G_1, G_3) \sqsubseteq \mathcal{I}_{<:}(G_2, G_4)$.

Proof. By Propositions 57 and 58. □

Proposition 60. *If* $\langle i_{11}, i_{12} \rangle \sqsubseteq \langle i_{21}, i_{22} \rangle$ *and* $\langle i_{31}, i_{32} \rangle \sqsubseteq \langle i_{41}, i_{42} \rangle$ *then* $\langle i_{11}, i_{12} \rangle \circ^{<:} \langle i_{31}, i_{32} \rangle \sqsubseteq \langle i_{21}, i_{22} \rangle \circ^{<:} \langle i_{41}, i_{42} \rangle$.

Proof. Let $i_{ij} = [s_{ij1}, s_{ij2}]$. Then, we know that

- $s_{1j1} \geq s_{2j1}$ and $s_{1j2} \leq s_{2j2}$.
- $s_{3j1} \geq s_{4j1}$ and $s_{3j2} \leq s_{4j2}$.
- $s_{112} \curlywedge s_{122} \curlywedge s_{312} \leq s_{212} \curlywedge s_{222} \curlywedge s_{412}$.
- $s_{121} \curlyvee s_{311} \curlyvee s_{321} \geq s_{221} \curlyvee s_{411} \curlyvee s_{421}$.
- $[s_{111}, s_{112} \curlywedge s_{122} \curlywedge s_{312}] \sqsubseteq [s_{211}, s_{212} \curlywedge s_{222} \curlywedge s_{412}]$.
- $[s_{121} \curlyvee s_{311} \curlyvee s_{321}, s_{322}] \sqsubseteq [s_{221} \curlyvee s_{411} \curlyvee s_{421}, s_{422}]$.
- $\langle i_{11}, i_{12} \rangle \circ^{<:} \langle i_{31}, i_{32} \rangle = \langle [s_{111}, s_{112} \curlywedge s_{122} \curlywedge s_{312}], [s_{121} \curlyvee s_{311} \curlyvee s_{321}, s_{322}] \rangle$.
- $\langle i_{21}, i_{22} \rangle \circ^{<:} \langle i_{41}, i_{42} \rangle = \langle [s_{211}, s_{212} \curlywedge s_{222} \curlywedge s_{412}], [s_{221} \curlyvee s_{411} \curlyvee s_{421}, s_{422}] \rangle$.

Finally, $\langle i_{11}, i_{12} \rangle \circ^{<:} \langle i_{31}, i_{32} \rangle \sqsubseteq \langle i_{21}, i_{22} \rangle \circ^{<:} \langle i_{41}, i_{42} \rangle$. □

Proposition 61. *If* $\langle \Xi_{11}, \Xi_{12} \rangle \sqsubseteq \langle \Xi_{21}, \Xi_{22} \rangle$ *and* $\langle \Xi_{31}, \Xi_{32} \rangle \sqsubseteq \langle \Xi_{41}, \Xi_{42} \rangle$ *then* $\langle \Xi_{11}, \Xi_{12} \rangle \circ^{<:} \langle \Xi_{31}, \Xi_{32} \rangle \sqsubseteq \langle \Xi_{21}, \Xi_{22} \rangle \circ^{<:} \langle \Xi_{41}, \Xi_{42} \rangle$.

Proof. By induction hypothesis on $\langle \Xi_{11}, \Xi_{12} \rangle \circ^{<:} \langle \Xi_{31}, \Xi_{32} \rangle$. All cases follow trivially (with no hypotheses) or by induction hypothesis with Proposition 60. □

Proposition 62. *If* $\langle g_{11}, g_{12} \rangle \sqsubseteq \langle g_{21}, g_{22} \rangle$ *and* $\langle g_{31}, g_{32} \rangle \sqsubseteq \langle g_{41}, g_{42} \rangle$ *then* $\langle g_{11}, g_{12} \rangle \circ^{<:} \langle g_{31}, g_{32} \rangle \sqsubseteq \langle g_{21}, g_{22} \rangle \circ^{<:} \langle g_{41}, g_{42} \rangle$.

Proof. By induction on $\langle g_{11}, g_{12} \rangle \circ^{<:} \langle g_{31}, g_{32} \rangle$. All cases follow trivially (with no hypotheses) or by induction hypotheses with Proposition 61. □

Proposition 63 (Consistent transitivity monotonicity). *If* $\varepsilon_1 \sqsubseteq \varepsilon_2$ *and* $\varepsilon_3 \sqsubseteq \varepsilon_4$ *then* $\varepsilon_1 \circ^{<:} \varepsilon_3 \sqsubseteq \varepsilon_2 \circ^{<:} \varepsilon_4$.

Proof. By Propositions 62 and 61. □

Lemma 64. *If* $t^{G_1} \sqsubseteq t^{G_2}$, *then* $G_1 \sqsubseteq G_2$.

Proof. By induction in the syntax of t^{G_1}.

Case $\varepsilon r :: G$, $\varepsilon \lambda x.t :: G$, $\varepsilon \langle \lambda x.t, \gamma \rangle :: G$, $\varepsilon \mathsf{ctx}(\gamma, t) :: G$, $\varepsilon \mathsf{tt} :: G$, $\varepsilon(\mathsf{inl}^g t) :: G$, $\varepsilon(\mathsf{inr}^g t) :: G$, $\varepsilon t :: G$. By inspection on $(\mathrm{I}G{\sqsubseteq_{::}})$.

Case $t^{G_1} = x^{G_1}$. By inspection on $(\mathrm{I}G{\sqsubseteq_x})$.

Case $t^{G_1} = t_{11}^{\mathbb{R};\Xi_{11}} + t_{12}^{\mathbb{R};\Xi_{12}}$, where $G_1 = \mathbb{R};\Xi_{11} + \Xi_{12}$. By inspection on $(\mathrm{I}G\sqsubseteq_{+})$, $t^{G_2} = t_{21}^{\mathbb{R};\Xi_{21}} + t_{22}^{\mathbb{R};\Xi_{22}}$, where $G_2 = \mathbb{R};\Xi_{21} + \Xi_{22}$, $t_{11}^{\mathbb{R};\Xi_{11}} \sqsubseteq t_{21}^{\mathbb{R};\Xi_{21}}$, $t_{12}^{\mathbb{R};\Xi_{12}} \sqsubseteq t_{22}^{\mathbb{R};\Xi_{22}}$. By induction hypothesis, $\mathbb{R};\Xi_{11} \sqsubseteq \mathbb{R};\Xi_{21}$ and $\mathbb{R};\Xi_{12} \sqsubseteq \mathbb{R};\Xi_{22}$. Finally, by Proposition 37, $G_1 \sqsubseteq G_2$.

Case $t^{G_1} = t_{11} \leq t_{12}$. Analogous.

Case $t^{G_1} = t_1^{(x:g_{11})\xrightarrow{\Xi_{12}} g_{12};\Xi_1}\ t_{11}^{g_{11};\Xi_{11}}$, where $G_1 = [\Xi_{11}/x]g_{12};\Xi_1 + [\Xi_{11}/x]\Xi_{12}$. By inspection on $(\mathrm{I}G\sqsubseteq_{@})$, $t^{G_2} = t_2^{(x:g_{21})\xrightarrow{\Xi_{22}} g_{22};\Xi_2}\ t_{21}^{g_{21};\Xi_{21}}$, where $G_2 = [\Xi_{21}/x]g_{22};\Xi_2 + [\Xi_{21}/x]\Xi_{22}$, $t_1^{(x:g_{11})\xrightarrow{\Xi_{12}} g_{12};\Xi_1} \sqsubseteq t_2^{(x:g_{21})\xrightarrow{\Xi_{22}} g_{22};\Xi_2}$, $t_{11}^{g_{11};\Xi_{11}} \sqsubseteq t_{21}^{g_{21};\Xi_{21}}$. By induction hypothesis, $(x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1 \sqsubseteq (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2$ and $g_{11};\Xi_{11} \sqsubseteq g_{21};\Xi_{21}$. Finally, by Propositions 46, 45 and 37, $G_1 \sqsubseteq G_2$.

Case $t^{G_1} = \mathsf{case}\ t_{11}^{g_{111}{}^{\Xi_{111}}\oplus^{\Xi_{112}} g_{112};\Xi_{11}}\ \mathsf{of}\,\{\, x^{g_{11};x} \Rightarrow t_{12}^{g_{12};\Xi_{12}} \,\}\ \{\, y^{g_{11};y} \Rightarrow t_{13}^{g_{13};\Xi_{13}} \,\}$, where $G_1 = [\Xi_{11}+\Xi_{111}/x]g_{12} \curlyvee [\Xi_{11}+\Xi_{112}/y]g_{13};\Xi_{11} \curlyvee [\Xi_{11}+\Xi_{111}/x]\Xi_{12} \curlyvee [\Xi_{11}+\Xi_{112}/y]\Xi_{13}$. By inspection on $(\mathrm{I}G\sqsubseteq_{\mathsf{case}})$, $t^{G_2} = \mathsf{case}\ t_{21}^{g_{211}{}^{\Xi_{211}}\oplus^{\Xi_{212}} g_{212};\Xi_{21}}\ \mathsf{of}\,\{\, x^{g_{21};x} \Rightarrow t_{22}^{g_{22};\Xi_{22}} \,\}\ \{\, y^{g_{21};y} \Rightarrow t_{23}^{g_{23};\Xi_{23}} \,\}$, where $G_2 = [\Xi_{21}+\Xi_{211}/x]g_{22} \curlyvee [\Xi_{21}+\Xi_{212}/y]g_{23};\Xi_{21} \curlyvee [\Xi_{21}+\Xi_{211}/x]\Xi_{22} \curlyvee [\Xi_{21}+\Xi_{212}/y]\Xi_{23}$, $t_{11}^{g_{111}{}^{\Xi_{111}}\oplus^{\Xi_{112}} g_{112};\Xi_{11}} \sqsubseteq t_{21}^{g_{211}{}^{\Xi_{211}}\oplus^{\Xi_{212}} g_{212};\Xi_{21}}$, $g_{11};x \sqsubseteq g_{21};x$, $t_{12}^{g_{12};\Xi_{12}} \sqsubseteq t_{22}^{g_{22};\Xi_{22}}$, $g_{12};y \sqsubseteq g_{22};y$, $t_{13}^{g_{13};\Xi_{13}} \sqsubseteq t_{23}^{g_{23};\Xi_{23}}$. By induction hypothesis, $g_{111}{}^{\Xi_{111}}\oplus^{\Xi_{112}} g_{112};\Xi_{11} \sqsubseteq g_{211}{}^{\Xi_{211}}\oplus^{\Xi_{212}} g_{212};\Xi_{21}$, $g_{12};\Xi_{12} \sqsubseteq g_{22};\Xi_{22}$ and $g_{13};\Xi_{13} \sqsubseteq g_{23};\Xi_{23}$. Finally, by direct application of propositions 37, 46, 45, 40 and 39, $G_1 \sqsubseteq G_2$.

□

Proposition 65. *If $g_1 \mathrel{\widetilde{<:}} g_2$, $g_1 \sqsubseteq g'_1$ and $g_2 \sqsubseteq g'_2$, then $g'_1 \mathrel{\widetilde{<:}} g'_2$.*

Proof. By definition of $\cdot \mathrel{\widetilde{<:}} \cdot$, there exists $\langle \tau_1, \tau_2 \rangle \in C_g^2(g_1, g_2)$ such that $\tau_1 <: \tau_2$. From $g_1 \sqsubseteq g'_1$ and $g_2 \sqsubseteq g'_2$ it follows that $C_g(g_1) \subseteq C_g(g'_1)$ and $C_g(g_2) \subseteq C_g(g'_2)$. Therefore, $\langle \tau_1, \tau_2 \rangle \in C_g^2(g'_1, g'_2)$ and the result holds. □

Proposition 66. *If $G_1 \mathrel{\widetilde{<:}} G_2$, $G_1 \sqsubseteq G'_1$ and $G_2 \sqsubseteq G'_2$, then $G'_1 \mathrel{\widetilde{<:}} G'_2$.*

Proof. By definition of $\cdot \mathrel{\widetilde{<:}} \cdot$, there exists $\langle \mathrm{T}_1, \mathrm{T}_2 \rangle \in C^2(G_1, G_2)$ such that $\mathrm{T}_1 <: \mathrm{T}_2$. From $G_1 \sqsubseteq G'_1$ and $G_2 \sqsubseteq G'_2$ it follows that $C(G_1) \subseteq C(G'_1)$ and $C(G_2) \subseteq C(G'_2)$. Therefore, $\langle \mathrm{T}_1, \mathrm{T}_2 \rangle \in C^2(G'_1, G'_2)$ and the result holds. □

Proposition 67. *If $i_1 \mathrel{\widetilde{<:}} i_3$ and $i_2 \mathrel{\widetilde{<:}} i_4$, then $i_1 + i_2 \mathrel{\widetilde{<:}} i_3 + i_4$.*

Proof. Let $i_i = [s_{i1}, s_{i2}]$. We know that $s_{11} \leq s_{32}$ and $s_{21} \leq s_{42}$. Then, $s_{11} + s_{21} \leq s_{32} + s_{42}$. Finally, $i_1 + i_2 \mathrel{\widetilde{<:}} i_3 + i_4$. □

Proposition 68. *If $\Xi_1 \mathrel{\widetilde{<:}} \Xi_3$ and $\Xi_2 \mathrel{\widetilde{<:}} \Xi_4$, then $\Xi_1 + \Xi_2 \mathrel{\widetilde{<:}} \Xi_3 + \Xi_4$.*

Proof. By induction on $\Xi_1 \mathrel{\widetilde{<:}} \Xi_3$. All cases follow with no premises or by induction hypothesis and Proposition 67. □

Proposition 69. *If $\Xi_1 \sqsubseteq \Xi_2$ then $\Delta \cdot \Xi_1 \sqsubseteq \Delta \cdot \Xi_2$.*

Proof.

$$\begin{aligned}
\Xi_1 \sqsubseteq \Xi_2 &\iff \forall x \in dom(\Xi_1) \cup dom(\Xi_2).\Xi_1(x) \sqsubseteq \Xi_2(x) \\
&\implies \forall x \in dom(\Xi_1) \cup dom(\Xi_2).\Delta(x) * \Xi_1(x) \sqsubseteq \Delta(x) * \Xi_2(x) \quad \text{By Proposition 34} \\
&\implies \sum_{x \in dom(\Xi_1) \cup dom(\Xi_2)} \Delta(x) * \Xi_1(x) \sqsubseteq \sum_{x \in dom(\Xi_1) \cup dom(\Xi_2)} \Delta(x) * \Xi_2(x) \\
&\implies \sum_{x \in dom(\Xi_1)} \Delta(x) * \Xi_1(x) \sqsubseteq \sum_{x \in dom(\Xi_2)} \Delta(x) * \Xi_2(x) \\
&\iff \Delta \cdot \Xi_1 \sqsubseteq \Delta \cdot \Xi_2
\end{aligned}$$

□

Proposition 70. *If* $\varepsilon_1 \triangleright \mathbb{R}; \Xi_{11} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{12}$ *and* $\varepsilon_2 \triangleright \mathbb{R}; \Xi_{21} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{22}$, *then* $\varepsilon_1 +_{\Xi} \varepsilon_2 \triangleright \mathbb{R}; \Xi_{11} + \Xi_{21} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{12} + \Xi_{22}$.

Proof. Let $\varepsilon_i = \langle \mathbb{R}; \Xi'_{i1}, \mathbb{R}; \Xi'_{i2} \rangle$. By definition, we know that $\Xi'_{i1} \sqsubseteq \Xi_{i1}$ and $\Xi'_{i2} \sqsubseteq \Xi_{i2}$. By Proposition 37, $\Xi'_{11} + \Xi'_{21} \sqsubseteq \Xi_{11} + \Xi_{21}$ and $\Xi'_{12} + \Xi'_{22} \sqsubseteq \Xi_{12} + \Xi_{22}$. Finally, $\varepsilon_1 +_{\Xi} \varepsilon_2 \triangleright \mathbb{R}; \Xi_{11} + \Xi_{21} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{12} + \Xi_{22}$. □

Proposition 71. *If* $\varepsilon_1 \triangleright \mathbb{R}; \Xi_{11} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{12}$ *and* $\varepsilon_2 \triangleright \mathbb{R}; \Xi_{21} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{22}$, *then* $\varepsilon_1 \curlyvee_{\Xi} \varepsilon_2 \triangleright \mathbb{R}; \Xi_{11} \curlyvee \Xi_{21} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{12} \curlyvee_{\Xi} \Xi_{22}$.

Proof. Let $\varepsilon_i = \langle \mathbb{R}; \Xi'_{i1}, \mathbb{R}; \Xi'_{i2} \rangle$. By definition, we know that $\Xi'_{i1} \sqsubseteq \Xi_{i1}$ and $\Xi'_{i2} \sqsubseteq \Xi_{i2}$. By Proposition 39, $\Xi'_{11} \curlyvee \Xi'_{21} \sqsubseteq \Xi_{11} \curlyvee \Xi_{21}$ and $\Xi'_{12} \curlyvee \Xi'_{22} \sqsubseteq \Xi_{12} \curlyvee \Xi_{22}$. Finally, $\varepsilon_1 \curlyvee_{\Xi} \varepsilon_2 \triangleright \mathbb{R}; \Xi_{11} \curlyvee \Xi_{21} \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_{12} \curlyvee \Xi_{22}$. □

Proposition 72. *If* $\varepsilon \triangleright G_1 \mathrel{\widetilde{<:}} G_2$, *then* $[\Xi/x]\varepsilon \triangleright [\Xi/x]G_1 \mathrel{\widetilde{<:}} [\Xi/x]G_2$

Proof. Let $\varepsilon = \langle G'_1, G'_2 \rangle$. By definition, we know that $G'_1 \sqsubseteq G_1$ and $G'_2 \sqsubseteq G_2$. By Proposition 47, $[\Xi/x]G'_1 \sqsubseteq [\Xi/x]G_1$ and $[\Xi/x]G'_2 \sqsubseteq [\Xi/x]G_2$. Finally, $[\Xi/x]\varepsilon \triangleright [\Xi/x]G_1 \mathrel{\widetilde{<:}} [\Xi/x]G_2$. □

Lemma 73 (Substitutions)**.** *If* $t \in \mathbb{T}[g; \Xi]$ *and* t *is not a runtime term, then* $\forall x, \Xi'$ *such that* $x \in FV(t)$, $[\Xi'/x]t \in \mathbb{T}[[\Xi'/x]g; [\Xi'/x]\Xi]$.

Proof. By induction on the structure of t.

- Case $r, b, \mathtt{tt}$: Trivial.
- Case $t_1 + t_2$: We know that,

$$(\text{IGplus})\ \frac{t_1 \in \mathbb{T}[\mathbb{R}; \Xi_1] \qquad t_2 \in \mathbb{T}[\mathbb{R}; \Xi_2]}{t_1 + t_2 \in \mathbb{T}[\mathbb{R}; \Xi_1 + \Xi_2]}$$

Therefore, by induction hypothesis,

$$(\text{IG}\textsc{plus})\ \frac{[\Xi'/x]t_1 \in \mathbb{T}[\mathbb{R};\Xi_1] \qquad [\Xi'/x]t_2 \in \mathbb{T}[\mathbb{R};\Xi_2]}{[\Xi'/x](t_1+t_2) \in \mathbb{T}[\mathbb{R};[\Xi'/x](\Xi_1+\Xi_2)]}$$

And the result holds.

- Case $\lambda x.t, t_1\ t_2, t :: g; \Xi, \mathsf{inl}^g t, \mathsf{inr}^g t$: Analogous.
- Case $x^{g;\Xi}$: We know that $x^{g;\Xi} \in \mathbb{T}[g;\Xi]$. Then, by (IG$\textsc{var}$), $[\Xi'/x]x^{g;\Xi} = x^{[\Xi'/x]g;[\Xi'/x]\Xi} \in \mathbb{T}[[\Xi'/x]g;[\Xi'/x]\Xi]$. And the result holds.

□

Proposition 74. $\langle i_{11}, i_{12} \rangle \circ^{<:} (\langle i_{21}, i_{22} \rangle \circ^{<:} \langle i_{31}, i_{32} \rangle) = (\langle i_{11}, i_{12} \rangle \circ^{<:} \langle i_{21}, i_{22} \rangle) \circ^{<:} \langle i_{31}, i_{32} \rangle$

Proof.

$$\begin{aligned}
&(\langle [s_{11}, s_{12}], [s_{13}, s_{14}] \rangle \circ^{<:} \langle [s_{21}, s_{22}], [s_{23}, s_{24}] \rangle) \circ^{<:} \langle [s_{31}, s_{32}], [s_{33}, s_{34}] \rangle \\
&= \langle [s_{11}, \min(s_{12}, s_{14}, s_{22})], [\max(s_{13}, s_{21}, s_{23}), s_{24}] \rangle \circ^{<:} \langle [s_{31}, s_{32}], [s_{33}, s_{34}] \rangle \\
&= \langle [s_{11}, \min(\min(s_{12}, s_{14}, s_{22}), s_{24}, s_{32})], [\max(\max(s_{13}, s_{21}, s_{23}), s_{31}, s_{33}), s_{34}] \rangle \\
&= \langle [s_{11}, \min(s_{12}, s_{14}, s_{22}, s_{24}, s_{32})], [\max(s_{13}, s_{21}, s_{23}, s_{31}, s_{33}), s_{34}] \rangle \\
&= \langle [s_{11}, \min(s_{12}, s_{14}, \min(s_{22}, s_{24}, s_{32}))], [\max(s_{13}, s_{21}, \max(s_{23}, s_{31}, s_{33})), s_{34}] \rangle \\
&= \langle [s_{11}, s_{12}], [s_{13}, s_{14}] \rangle \circ^{<:} \langle [s_{21}, \min(s_{22}, s_{24}, s_{32})], [\max(s_{23}, s_{31}, s_{33}), s_{34}] \rangle \\
&= \langle [s_{11}, s_{12}], [s_{13}, s_{14}] \rangle \circ^{<:} (\langle [s_{21}, s_{22}], [s_{23}, s_{24}] \rangle \circ^{<:} \langle [s_{31}, s_{32}], [s_{33}, s_{34}] \rangle)
\end{aligned}$$

□

Proposition 75. $\langle \Xi_1, \Xi_2 \rangle \circ^{<:} (\langle \Xi_3, \Xi_4 \rangle \circ^{<:} \langle \Xi_5, \Xi_6 \rangle) = (\langle \Xi_1, \Xi_2 \rangle \circ^{<:} \langle \Xi_3, \Xi_4 \rangle) \circ^{<:} \langle \Xi_5, \Xi_6 \rangle$

Proof. We proceed by induction on the structure of Ξ_j.

Case $\Xi_j = \varnothing$. Trivial.

Case $\Xi_j = \Xi_{j0} + i_j x$.

1. Let $\langle \Xi'_{11}, \Xi'_{41} \rangle = \langle \Xi_{10}, \Xi_{20} \rangle \circ^{<:} \langle \Xi_{30}, \Xi_{40} \rangle$.
2. Let $\langle i'_{11}, i'_{41} \rangle = \langle i_1, i_2 \rangle \circ^{<:} \langle i_3, i_4 \rangle$.
3. Let $\langle \Xi''_{11}, \Xi'_{61} \rangle = \langle \Xi'_{11}, \Xi'_{41} \rangle \circ^{<:} \langle \Xi_{50}, \Xi_{60} \rangle = (\langle \Xi_{10}, \Xi_{20} \rangle \circ^{<:} \langle \Xi_{30}, \Xi_{40} \rangle) \circ^{<:} \langle \Xi_{50}, \Xi_{60} \rangle$.
4. Let $\langle i''_{11}, i'_{61} \rangle = \langle i'_{11}, i'_{41} \rangle \circ^{<:} \langle i_5, i_6 \rangle = (\langle i_1, i_2 \rangle \circ^{<:} \langle i_3, i_4 \rangle) \circ^{<:} \langle i_5, i_6 \rangle$.
5. Let $\langle \Xi'_{32}, \Xi'_{62} \rangle = \langle \Xi_{30}, \Xi_{40} \rangle \circ^{<:} \langle \Xi_{50}, \Xi_{60} \rangle$.
6. Let $\langle i'_{32}, i'_{62} \rangle = \langle i_3, i_4 \rangle \circ^{<:} \langle i_5, i_6 \rangle$.
7. Let $\langle \Xi'_{12}, \Xi''_{62} \rangle = \langle \Xi_{10}, \Xi_{20} \rangle \circ^{<:} \langle \Xi'_{32}, \Xi'_{62} \rangle = \langle \Xi_{10}, \Xi_{20} \rangle \circ^{<:} (\langle \Xi_{30}, \Xi_{40} \rangle \circ^{<:} \langle \Xi_{50}, \Xi_{60} \rangle)$.
8. Let $\langle i'_{12}, i''_{62} \rangle = \langle i_1, i_2 \rangle \circ^{<:} \langle i'_{32}, i'_{62} \rangle = \langle i_1, i_2 \rangle \circ^{<:} (\langle i_3, i_4 \rangle \circ^{<:} \langle i_5, i_6 \rangle)$.

9. By Proposition 74, $\langle\, i''_{11}, i'_{61} \,\rangle = \langle\, i'_{12}, i''_{62} \,\rangle$.
10. By induction hypothesis, $\langle\, \Xi''_{11}, \Xi'_{61} \,\rangle = \langle\, \Xi'_{12}, \Xi''_{62} \,\rangle$.

Notice that

– $(\langle\, \Xi_1, \Xi_2 \,\rangle \circ^{<:} \langle\, \Xi_3, \Xi_4 \,\rangle) \circ^{<:} \langle\, \Xi_5, \Xi_6 \,\rangle = \langle\, \Xi''_{11} + i''_{11}x, \Xi'_{61} + i'_{61}x \,\rangle$, and
– $\langle\, \Xi_1, \Xi_2 \,\rangle \circ^{<:} (\langle\, \Xi_3, \Xi_4 \,\rangle \circ^{<:} \langle\, \Xi_5, \Xi_6 \,\rangle) = \langle\, \Xi'_{12} + i'_{12}x, \Xi''_{62} + i''_{62}x \,\rangle$.

Finally, by (9) and (10), $\langle\, \Xi_1, \Xi_2 \,\rangle \circ^{<:} (\langle\, \Xi_3, \Xi_4 \,\rangle \circ^{<:} \langle\, \Xi_5, \Xi_6 \,\rangle) = (\langle\, \Xi_1, \Xi_2 \,\rangle \circ^{<:} \langle\, \Xi_3, \Xi_4 \,\rangle) \circ^{<:} \langle\, \Xi_5, \Xi_6 \,\rangle$.

□

Proposition 76. $\langle\, g_1, g_2 \,\rangle \circ^{<:} (\langle\, g_3, g_4 \,\rangle \circ^{<:} \langle\, g_5, g_6 \,\rangle) = (\langle\, g_1, g_2 \,\rangle \circ^{<:} \langle\, g_3, g_4 \,\rangle) \circ^{<:} \langle\, g_5, g_6 \,\rangle$

Proof. Analogous to 75. By induction on the structure of g_j and Proposition 75. □

Proposition 77 (Consistent transitivity associativity)**.** $\varepsilon_1 \circ^{<:} (\varepsilon_2 \circ^{<:} \varepsilon_3) = (\varepsilon_1 \circ^{<:} \varepsilon_2) \circ^{<:} \varepsilon_3$

Proof. Follows directly from Propositions 76 and 75. □

B.2. Galois connections

Proposition 7 (Galois connection for sensitivities)**.** *$\langle C_i, A_i \rangle$ is a Galois connection, i.e.:*

1. *(Soundness) for any non-empty set of static sensitivities $S = \{\, \overline{s} \,\}$, we have $S \subseteq C_i(A_i(S))$*
2. *(Optimality) for any gradual sensitivity i, we have $i \sqsubseteq A_i(C_i(i))$.*

Proof. We first prove Soundness.

- Let $S = \{\, \overline{s} \,\}$.
- Then $A_i(S) = [\min(S), \max(S)]$.
- But $C_i([\min(S), \max(S)]) = \{\, s \mid s \geq \min(S) \wedge s \leq \max(S) \,\}$.
- Finally, for any $s \in S$, $s \geq \min(S) \wedge s \leq \max(S)$, so $S \subseteq C_i(A_i(S))$.

We then prove Optimality.

- Let $i = [s_1, s_2]$.

- Then, let $S = C_i(i) = \{ s \mid s \geq s_1 \wedge s \leq s_2 \}$.
- But $A_i(S) = [\min(S), \max(S)] = [s_1, s_2]$.
- Finally, the result trivially holds.

□

Proposition 78 (Galois connection for sensitivity environments)**.** *$\langle C_\Xi, A_\Xi \rangle$ is a Galois connection, i.e.:*

1. *(Soundness) for any non-empty set of static sensitivity environments $S = \{ \overline{\Sigma} \}$, we have $S \subseteq C_\Xi(A_\Xi(S))$*
2. *(Optimality) for any gradual sensitivity environment Ξ, we have $\Xi \sqsubseteq A_\Xi(C_\Xi(\Xi))$.*

Proof. We first prove Soundness, by induction on the structure of the non-empty set S.

Case $S = \{ \varnothing \}$. Then $A_\Xi(S) = \varnothing$, but $C_\Xi(\varnothing) = \varnothing$. Finally, the result trivially holds.

Case $S = \{ \overline{\Sigma + sx} \}$. Then $A_\Xi(S) = A_\Xi(\{ \overline{\Sigma} \}) + A_i(\{ \overline{s} \})x$. But $C_\Xi(A_\Xi(\{ \overline{\Sigma} \}) + A_i(\{ \overline{s} \})x) = C_\Xi(A_\Xi(\{ \overline{\Sigma} \})) + C_i(A_i(\{ \overline{s} \}))x$. By induction hypothesis $\{ \overline{\Sigma} \} \subseteq C_\Xi(A_\Xi(\{ \overline{\Sigma} \}))$ and, by Proposition 7, $\{ \overline{s} \} \subseteq C_i(A_i(\{ \overline{s} \}))$. Finally, $S \subseteq C_\Xi(A_\Xi(S))$.

We then prove Optimality, by induction on the structure of Ξ.

Case $\Xi = \varnothing$. Then, $C_\Xi(\varnothing) = \{ \varnothing \}$. But $A_\Xi(\{ \varnothing \}) = \varnothing$ and the result trivially holds.

Case $\Xi = \Xi' + ix$. Then, $C_\Xi(\Xi' + ix) = C_\Xi(\Xi') + C_i(i)x$. But $A_\Xi(C_\Xi(\Xi') + C_i(i)x) = A_\Xi(C_\Xi(\Xi')) + A_i(C_i(i))x$. By induction hypothesis, $\Xi' \sqsubseteq A_\Xi(C_\Xi(\Xi'))$ and, by Proposition 7, $i \sqsubseteq A_i(C_i(i))$. Finally, the result holds.

□

Proposition 79 (Galois connection for types)**.** *$\langle C_g, A_g \rangle$ is a Galois connection, i.e.:*

1. *(Soundness) for any non-empty set of static types $S = \{ \overline{\tau} \}$, we have $S \subseteq C_g(A_g(S))$*
2. *(Optimality) for any gradual type g, we have $g \sqsubseteq A_g(C_g(g))$.*

Proof. We first prove Soundness, by induction on the structure of the non-empty set S.

Case $S = \{ \mathbb{R} \}$. Then, $A_g(\{ \mathbb{R} \}) = \mathbb{R}$. But $C_g(\mathbb{R}) = \mathbb{R}$. Finally, the result trivially holds.

Case $S = \{ \mathbb{B} \}$ and $S = \{ \mathsf{unit} \}$. Analogous to case $S = \{ \mathbb{R} \}$.

Case $S = \{\, \overline{(x : \tau_1) \xrightarrow{\Sigma} \tau_1} \,\}$.

- Then, $A_g(S) = (x : A_g(\{\, \overline{\tau_1} \,\})) \xrightarrow{A_\Xi(\{\, \overline{\Sigma} \,\})} A_g(\{\, \overline{\tau_2} \,\})$.
- But $C_g(A_g(S)) = (x : C_g(A_g(\{\, \overline{\tau_1} \,\}))) \xrightarrow{C_\Xi(A_\Xi(\{\, \overline{\Sigma} \,\}))} C_g(A_g(\{\, \overline{\tau_2} \,\}))$.
- By induction hypotheses, $\{\, \overline{\tau_1} \,\} \subseteq C_g(A_g(\{\, \overline{\tau_1} \,\}))$ and $\{\, \overline{\tau_2} \,\} \subseteq C_g(A_g(\{\, \overline{\tau_2} \,\}))$.
- Additionally, by Proposition 78, $\{\, \overline{\Sigma} \,\} \subseteq C_\Xi(A_\Xi(\{\, \overline{\Sigma} \,\}))$.
- Finally, $S \subseteq C_g(A_g(S))$.

Case $S = \{\, \overline{\tau_1 \,{}^{\Sigma_1}\!\oplus^{\Sigma_2}\, \tau_2} \,\}$.

- Then, $A_g(S) = A_g(\{\, \overline{\tau_1} \,\}) \,{}^{A_g(\{\, \overline{\Sigma_1} \,\})}\!\oplus^{A_g(\{\, \overline{\Sigma_2} \,\})}\, A_g(\{\, \overline{\tau_2} \,\})$.
- But $C_g(A_g(S)) = C_g(A_g(\{\, \overline{\tau_1} \,\})) \,{}^{C_\Xi(A_g(\{\, \overline{\Sigma_1} \,\}))}\!\oplus^{C_\Xi(A_g(\{\, \overline{\Sigma_2} \,\}))}\, C_g(A_g(\{\, \overline{\tau_2} \,\}))$.
- By induction hypotheses, $\{\, \overline{\tau_1} \,\} \subseteq C_g(A_g(\{\, \overline{\tau_1} \,\}))$ and $\{\, \overline{\tau_2} \,\} \subseteq C_g(A_g(\{\, \overline{\tau_2} \,\}))$.
- Additionally, by Proposition 78, $\{\, \overline{\Sigma_1} \,\} \subseteq C_\Xi(A_\Xi(\{\, \overline{\Sigma_1} \,\}))$ and $\{\, \overline{\Sigma_2} \,\} \subseteq C_\Xi(A_\Xi(\{\, \overline{\Sigma_2} \,\}))$.
- Finally, $S \subseteq C_g(A_g(S))$.

We then prove Optimality by induction of the structure of g.

Case $g = \mathbb{R}$. Then, $C_g(\mathbb{R}) = \{\, \mathbb{R} \,\}$. But $A_g(\{\, \mathbb{R} \,\}) = \mathbb{R}$ and the result trivially holds.

Case $g = \mathbb{B}$ and $g = \mathsf{unit}$. Analogous to case $g = \mathbb{R}$.

Case $g = (x : g_1) \xrightarrow{\Xi} g_2$.

- Then, $C_g(g) = (x : C_g(g_1)) \xrightarrow{C_\Xi(\Xi)} C_g(g_2)$.
- But $A_g(C_g(g)) = (x : A_g(C_g(g_1))) \xrightarrow{A_\Xi(C_\Xi(\Xi))} A_g(C_g(g_2))$.
- By induction hypotheses, $g_1 \sqsubseteq A_g(C_g(g_1))$ and $g_2 \sqsubseteq A_g(C_g(g_2))$.
- Additionally, by Proposition 78, $\Xi \sqsubseteq A_\Xi(C_\Xi(\Xi))$.
- Finally, the result holds.

Case $g = g_1 \,{}^{\Xi_1}\!\oplus^{\Xi_2}\, g_2$.

- Then, $C_g(g) = C_g(g_1) \,{}^{C_g(\Xi_1)}\!\oplus^{C_g(\Xi_2)}\, C_g(g_2)$.
- But $A_g(C_g(g)) = A_g(C_g(g_1)) \,{}^{A_\Xi(C_\Xi(\Xi_1))}\!\oplus^{A_\Xi(C_\Xi(\Xi_2))}\, A_g(C_g(g_2))$.
- By induction hypotheses, $g_1 \sqsubseteq A_g(C_g(g_1))$ and $g_2 \sqsubseteq A_g(C_g(g_2))$.
- Additionally, by Proposition 78, $\Xi_1 \sqsubseteq A_\Xi(C_\Xi(\Xi_1))$ and $\Xi_2 \sqsubseteq A_\Xi(C_\Xi(\Xi_2))$.
- Finally, the result holds.

$\square$

Proposition 8 (Galois connection for type-and-effects). *$\langle C, A \rangle$ is a Galois connection, i.e.:*

1. *(Soundness) for any non-empty set of static type-and-effects* $S = \{\overline{\mathrm{T}}\}$*, we have* $S \subseteq C(A(S))$

2. *(Optimality) for any gradual type-and-effect* G*, we have* $G \sqsubseteq A(C(G))$.

Proof. We first prove Soundness.

- Let $S = \{\overline{\tau; \Sigma}\}$.
- Then, $A(S) = A_g(\{\overline{\tau}\}); A_\Xi(\{\overline{\Sigma}\})$.
- But $C(A(S)) = C_g(A_g(\{\overline{\tau}\})); C_\Xi(A_\Xi(\{\overline{\Sigma}\}))$.
- By Propositions 79 and 78, $\{\overline{\tau}\} \subseteq C_g(A_g(\{\overline{\tau}\}))$ and $\{\overline{\Sigma}\} \subseteq C_\Xi(A_\Xi(\{\overline{\Sigma}\}))$, respectively.
- Finally, the result holds.

We then prove Optimality.

- Let $G = g; \Xi$.
- Then, $C(G) = C_g(g); C_\Xi(\Xi)$.
- But $A(C(G)) = A_g(C_g(g)); A_\Xi(C_\Xi(\Xi))$.
- By Propositions 79 and 78, $g \sqsubseteq A_g(C_g(g))$ and $\Xi \sqsubseteq A_\Xi(C_\Xi(\Xi))$, respectively.
- Finally, the result holds.

□

B.3. Type Safety

Lemma 80 (Canonical Forms)**.** *Consider a value* $v \in \mathbb{T}[g; \Xi]$*. Then,* $v = \varepsilon u :: g; \Xi$ *with* $u \in \mathbb{T}[g'; \Xi']$ *and* $\varepsilon \triangleright g'; \Xi' \widetilde{<:} g; \Xi$*. Furthermore:*

1. *If* $g = \mathbb{R}$ *then* $u = r$ *with* $r \in \mathbb{T}[\mathbb{R}; \varnothing]$.
2. *If* $g = \mathbb{B}$ *then* $u = b$ *with* $b \in \mathbb{T}[\mathbb{B}; \varnothing]$.
3. *If* $g = \mathsf{unit}$ *then* $u = \mathsf{tt}$ *with* $\mathsf{tt} \in \mathbb{T}[\mathsf{unit}; \varnothing]$.
4. *If* $g = (x : g_1) \xrightarrow{\Xi_2} g_2$ *then* $u = \langle \lambda x^{g_1'}.t, \gamma \rangle$ *with* $t \in \mathbb{T}[g_2'; \Xi_2']$.
5. *If* $g = g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2} \ g_2$ *then* u *is either:*

- $\mathsf{inl}^{g_2'} v_1$ *with* $v_1 \in \mathbb{T}[g_1'; \Xi_1']$.
- $\mathsf{inr}^{g_1'} v_2$ *with* $v_2 \in \mathbb{T}[g_2'; \Xi_2']$.

Proof. By direct inspection of the typechecking rules for intrinsic terms. □

Proposition 81 ($\longrightarrow$ is well-defined)**.** . *If* $t \vdash \gamma$*,* $t \in \mathbb{T}[G]$ *and* $t \xrightarrow{\gamma} t'$*, then* $t' \in \mathbb{T}[G] \cup \{\, \mathbf{error} \,\}$.

Proof. By induction on the structure of derivation of $t \xmapsto{\gamma} t'$.

Case (IGR-PLUS).

Then $t = \varepsilon_1 r_1 :: \mathbb{R}; \Xi_1 + \varepsilon_2 r_2 :: \mathbb{R}; \Xi_2$. Then, let $\varepsilon_1 = \langle\, \mathbb{R}; \varnothing, \mathbb{R}; \Xi_{12} \,\rangle$ and $\varepsilon_2 = \langle\, \mathbb{R}; \varnothing, \mathbb{R}; \Xi_{22} \,\rangle$,

$$\text{(IGPLUS)}\ \dfrac{\text{(IGASCR)}\ \dfrac{r_i \in \mathbb{T}[\mathbb{R}; \varnothing] \qquad \varepsilon_i \triangleright \mathbb{R}; \varnothing \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_i}{\varepsilon_i r_i :: \mathbb{R}; \Xi_i \in \mathbb{T}[\mathbb{R}; \Xi_i]}}{\varepsilon_1 r_1 :: \mathbb{R}; \Xi_1 + \varepsilon_2 r_2 :: \mathbb{R}; \Xi_2 \in \mathbb{T}[\mathbb{R}; \Xi_1 + \Xi_2]}$$

Then,

$$\varepsilon_1 r_1 :: \mathbb{R}; \Xi_1 + \varepsilon_2 r_2 :: \mathbb{R}; \Xi_2 \longrightarrow \varepsilon_3 r_3 :: \mathbb{R}; \Xi_1 + \Xi_2$$

where

$$r_3 = r_1 [\![+]\!] r_2$$

$$\varepsilon_3 = \varepsilon_1 +_{\Xi} \varepsilon_2$$

– By Proposition 70, $\varepsilon_3 \triangleright \mathbb{R}; \varnothing \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_1 + \Xi_2$.

Finally

$$\text{(IGASCR)}\ \dfrac{r_3 \in \mathbb{T}[\mathbb{R}; \varnothing] \qquad \varepsilon_3 \triangleright \mathbb{R}; \varnothing \mathrel{\widetilde{<:}} \mathbb{R}; \Xi_1 + \Xi_2}{\varepsilon_3 r_3 :: \mathbb{R}; \Xi_1 + \Xi_2 \in \mathbb{T}[\mathbb{R}; \Xi_1 + \Xi_2]}$$

and the result holds

Case (IGR-LEQ). Analogous to (IGR-PLUS).

Case (IGR-VAR).

Then $t = x^{g;\Xi}$. Let $\gamma(x) = v$. We know that,

$$(\text{IG}\textsc{var})\ \frac{}{x^{g;\Xi} \in \mathbb{T}[g;\Xi]}$$

Also, we know, by the definition of well-formedness of γ, that $v \in \mathbb{T}[g;\Xi]$.
Then,

$$x^{g;\Xi} \longrightarrow \gamma(x)$$

And the result holds immediately.

Case $(\text{IG}\textsc{r-lam})$.

Then $t = \varepsilon\lambda x.t :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi$. Then,

$$(\text{IG}\textsc{ascr})\ \frac{\lambda x.t \in \mathbb{T}[(x : g_1') \xrightarrow{\Xi_2'} g_2'; \Xi'] \qquad \varepsilon \triangleright (x : g_1') \xrightarrow{\Xi_2'} g_2'; \Xi' \mathrel{\widetilde{<:}} (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi}{\varepsilon\lambda x.t :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi \in \mathbb{T}[(x : g_1) \xrightarrow{\Xi_2} g_2; \Xi]}$$

Therefore,

$$\varepsilon\lambda x.t :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi \xrightarrow{\gamma} \varepsilon\langle\lambda x.t, \gamma\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi$$

– We know by hypothesis that $\varepsilon\lambda x.t :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi \vdash \gamma$, but since ascriptions don't introduce fresh variables nor change inner terms types we also know that $\lambda x.t \vdash \gamma$.

Finally,

$$(\text{IG}\textsc{ascr})\ \frac{D_1 \qquad \varepsilon \triangleright (x : g_1') \xrightarrow{\Xi_2'} g_2'; \Xi' \mathrel{\widetilde{<:}} (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi}{\varepsilon\langle\lambda x.t, \gamma\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2; \Xi \in \mathbb{T}[(x : g_1) \xrightarrow{\Xi_2} g_2; \Xi]}$$

where,

$$D_1 = \quad (\text{IG}\textsc{closure})\ \frac{\lambda x.t \in \mathbb{T}[(x : g_1') \xrightarrow{\Xi_2'} g_2'; \Xi'] \qquad \lambda x.t \vdash \gamma}{\langle\lambda x.t, \gamma\rangle \in \mathbb{T}[(x : g_1') \xrightarrow{\Xi_2'} g_2'; \Xi']}$$

and the result holds.

Case (IG<span>R-APP</span>).

Then $t = (\varepsilon_1\langle\lambda x^{g_1''}.t,\gamma'\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2;\Xi)\ (\varepsilon_2 u :: g_1;\Xi_1)$.

$$\text{(IGAPP)}\ \dfrac{D_1 \qquad \text{(IGASCR)}\ \dfrac{u \in \mathbb{T}[g_1''';\Xi_1'''] \qquad \varepsilon_2 \triangleright g_1''';\Xi_1''' \mathrel{\widetilde{<:}} g_1;\Xi_1}{\varepsilon_2 u :: g_1;\Xi_1 \in \mathbb{T}[g_1;\Xi_1]}}{\varepsilon_1\langle\lambda x^{g_1''}.t,\gamma'\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2;\Xi\ \varepsilon_2 u :: g_1;\Xi_1 \in \mathbb{T}[[\Xi_1/x]g_2;\Xi' + [\Xi_1/x]\Xi]}$$

where,

$$D_1 = \text{(IGASCR)}\ \dfrac{D_2 \qquad \varepsilon_1 \triangleright (x : g_1'') \xrightarrow{\Xi_2''} g_2'';\varnothing \mathrel{\widetilde{<:}} (x : g_1) \xrightarrow{\Xi_2} g_2;\Xi}{\varepsilon_1\langle\lambda x^{g_1''}.t,\gamma'\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2;\Xi \in \mathbb{T}[(x : g_1) \xrightarrow{\Xi_2} g_2;\Xi]}$$

$$D_2 = \text{(IGCLOSURE)}\ \dfrac{\text{(IGLAM)}\ \dfrac{t \in \mathbb{T}[g_2'';\Xi_2'']}{\lambda x^{g_1''}.t \in \mathbb{T}[(x : g_1'') \xrightarrow{\Xi_2''} g_2'';\varnothing]} \qquad \lambda x^{g_1''}.t \vdash \gamma'}{\langle\lambda x^{g_1''}.t,\gamma'\rangle \in \mathbb{T}[(x : g_1'') \xrightarrow{\Xi_2''} g_2'';\varnothing]}$$

If $\varepsilon_2' = \varepsilon_2 \circ^{<:} sdom(\Xi_1,\varepsilon_1)$ is not defined, then $t \longrightarrow$ **error** and the result holds immediately. Suppose that the consistent transitivity hold, then:

$$\varepsilon_1\langle\lambda x^{g_1''}.t,\gamma'\rangle :: (x : g_1) \xrightarrow{\Xi_2} g_2;\Xi\ \varepsilon_2 u :: g_1;\Xi_1$$

$$\xrightarrow{\gamma} \varepsilon_{11}\mathsf{ctx}(\gamma_{ext}',[\Xi_1/x]t) :: [\Xi_1/x]g_2;\Xi + [\Xi_1/x]\Xi_2$$

where $\gamma_{ext}' = \gamma'[x \mapsto \varepsilon_2' u :: g_1'';\Xi_1]$, $\varepsilon_{11} = [\Xi_1/x]icod(\varepsilon_1)$.

- By inversion lemmas, we know that $icod(\varepsilon_1) \triangleright g_2'';\Xi_2'' \mathrel{\widetilde{<:}} g_2;\Xi_2$. Then, by Proposition 72, $\varepsilon_{11} \triangleright [\Xi_1/x]g_2'';[\Xi_1/x]\Xi_2'' \mathrel{\widetilde{<:}} [\Xi_1/x]g_2;[\Xi_1/x]\Xi_2$. Finally, we have that $\varepsilon_{11} \triangleright [\Xi_1/x]g_2'';[\Xi_1/x]\Xi_2'' \mathrel{\widetilde{<:}} [\Xi_1/x]g_2;\Xi + [\Xi_1/x]\Xi_2$.
- As $\varepsilon_2 \triangleright g_1''';\Xi_1''' \mathrel{\widetilde{<:}} g_1;\Xi_1$ and by inversion lemmas $sdom(\Xi_1,\varepsilon_1) \triangleright g_1;\Xi_1 \mathrel{\widetilde{<:}} g_1'';\Xi_1$, then $\varepsilon_2' \triangleright g_1''';\Xi_1''' \mathrel{\widetilde{<:}} g_1'';\Xi_1$.
- We want to prove that $[\Xi_1/x]t \vdash \gamma_{ext}'$: First, we know that $\lambda x^{g_1''}.t \vdash \gamma'$, then (1) $FV(\lambda x^{g_1''}.t) \subseteq dom(\gamma')$ and (2) $\forall y^{g_y;\Xi_y} \in FV(\lambda x^{g_1''}.t), \gamma(y) \in \mathbb{T}[g_y;\Xi_y]$. Also, $FV([\Xi_1/x]t) = FV(\lambda x^{g_1''}.t) \cup \{x\}$, then $FV([\Xi_1/x]t) \subseteq dom(\gamma') \cup \{x\} = dom(\gamma_{ext}')$. Since $x^{g_1'';\Xi_1} \notin FV(\lambda x^{g_1''}.t)$, we only need to prove that the value provided for x in γ_{ext}' is well-typed:

$$\text{(IGASCR)}\ \dfrac{u \in \mathbb{T}[g_1''';\Xi_1'''] \qquad \varepsilon_2' \triangleright g_1''';\Xi_1''' \mathrel{\widetilde{<:}} g_1'';\Xi_1}{\varepsilon_2' u :: g_1'';\Xi_1 \in \mathbb{T}[g_1'';\Xi_1]}$$

– We know that $t \in \mathbb{T}[g_2''; \Xi_2'']$. Then, by Proposition 73, $[\Xi_1/x]t \in \mathbb{T}[[\Xi_1/x]g_2''; [\Xi_1/x]\Xi_2'']$.

Finally,

$$(\mathrm{IG}\textsc{ascr})\ \frac{D_4 \qquad \varepsilon_{11} \triangleright [\Xi_1/x]g_2''; [\Xi_1/x]\Xi_2'' \mathrel{\widetilde{<:}} [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2}{\varepsilon_{11}\mathsf{ctx}(\gamma_{ext}', [\Xi_1/x]t) :: [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2 \in \mathbb{T}[[\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2]}$$

where

$$D_4 = (\mathrm{IG}\textsc{ctx})\ \frac{[\Xi_1/x]t \in \mathbb{T}[[\Xi_1/x]g_2''; [\Xi_1/x]\Xi_2''] \qquad [\Xi_1/x]t \vdash \gamma_{ext}'}{\mathsf{ctx}(\gamma_{ext}', [\Xi_1/x]t) \in \mathbb{T}[[\Xi_1/x]g_2''; [\Xi_1/x]\Xi_2'']}$$

and the result holds.

Case $(\mathrm{IG}\textsc{r-case-1})$.

Then, $t = \mathsf{case}\ (\varepsilon \mathsf{inl}^{g_{12}'}(\varepsilon_1 u :: g_{11}'; \Xi_{11}') :: g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1)\ \mathsf{of}\ \{\, x^{g_{11}} \Rightarrow t_2^{g_2;\Xi_2} \,\}\ \{\, y^{g_{12}} \Rightarrow t_3^{g_3;\Xi_3} \,\}$ and:

$$(\mathrm{IG}\textsc{case})\ \frac{D_1 \qquad t_2 \in \mathbb{T}[g_2; \Xi_2] \qquad t_3 \in \mathbb{T}[g_3; \Xi_3]}{\begin{array}{c}\mathsf{case}\ (\varepsilon \mathsf{inl}^{g_{12}'}(\varepsilon_1 u :: g_{11}'; \Xi_{11}') :: g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1)\ \mathsf{of}\ \{\, x^{g_{11}} \Rightarrow t_2^{g_2;\Xi_2} \,\}\ \{\, y^{g_{12}} \Rightarrow t_3^{g_3;\Xi_3} \,\} \in \\ \mathbb{T}[[\Xi_1 + \Xi_{11}/x]g_2 \sqcup [\Xi_1 + \Xi_{12}/y]g_3; \Xi_1 \sqcup [\Xi_1 + \Xi_{11}/x]\Xi_2 \sqcup [\Xi_1 + \Xi_{12}/y]\Xi_3]\end{array}}$$

$$D_1 = (\mathrm{IG}\textsc{ascr})\ \frac{D_2 \qquad \varepsilon \triangleright g_{11}' \,{}^{\Xi_{11}'}\!\oplus^{\varnothing} g_{12}'; \varnothing \mathrel{\widetilde{<:}} g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1}{\varepsilon \mathsf{inl}^{g_{12}'}(\varepsilon_1 u :: g_{11}'; \Xi_{11}') :: g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1 \in \mathbb{T}[g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1]}$$

$$D_2 = (\mathrm{IG}\textsc{inl})\ \frac{(\mathrm{IG}\textsc{ascr})\ \dfrac{u \in \mathbb{T}[g_{11}''; \Xi_{11}''] \qquad \varepsilon_1 \triangleright g_{11}''; \Xi_{11}'' \mathrel{\widetilde{<:}} g_{11}'; \Xi_{11}'}{\varepsilon_1 u :: g_{11}'; \Xi_{11}' \in \mathbb{T}[g_{11}'; \Xi_{11}']}}{\mathsf{inl}^{g_{12}'}(\varepsilon_1 u :: g_{11}'; \Xi_{11}') \in \mathbb{T}[g_{11}' \,{}^{\Xi_{11}'}\!\oplus^{\varnothing} g_{12}'; \varnothing]}$$

If $\varepsilon_2' = \mathcal{I}_{<:}([\Xi_x/x]g_2; [\Xi_x/x]\Xi_2, [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3; \Xi') \curlyvee_{\Xi} \varepsilon$ or $\varepsilon_1' = \varepsilon_1 \circ^{<:} \mathit{ileft}(\varepsilon)$ are not defined then $t \longrightarrow \mathbf{error}$, and the result holds immediately. Suppose that the consistent transitivity and the interior hold and are defined, then:

$$\mathsf{case}\ (\varepsilon \mathsf{inl}^{g_{12}'}(\varepsilon_1 u :: g_{11}'; \Xi_{11}') :: g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}} g_{12}; \Xi_1)\ \mathsf{of}\ \{\, x^{g_{11}} \Rightarrow t_2^{g_2;\Xi_2} \,\}\ \{\, y^{g_{12}} \Rightarrow t_3^{g_3;\Xi_3} \,\} \xrightarrow{\gamma}$$

$$\varepsilon_2'\mathsf{ctx}(\gamma_{ext}, t_{body}) :: g'; \Xi' \curlyvee \Xi_1$$

where

$$\begin{aligned}
\gamma_{ext} &= \gamma[x \mapsto \varepsilon_1' u :: g_{11}; \Xi_x] \\
t_{body} &= [\Xi_x/x]t_2 \\
\Xi_x &= \Xi_1 + \Xi_{11} \\
\Xi_y &= \Xi_1 + \Xi_{12} \\
g' &= [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3 \\
\Xi' &= [\Xi_x/x]\Xi_2 \sqcup [\Xi_y/y]\Xi_3
\end{aligned}$$

- We know that $\mathcal{I}_{<:}([\Xi_x/x]g_2; [\Xi_x/x]\Xi_2, [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3; \Xi') \triangleright [\Xi_x/x]g_2; [\Xi_x/x]\Xi_2 \mathrel{\widetilde{<:}} g'; \Xi'$, and $\varepsilon \triangleright g_{11}' \ {}^{\Xi_{11}' \oplus \varnothing}\ g_{12}'; \varnothing \mathrel{\widetilde{<:}} g_{11} \ {}^{\Xi_{11} \oplus \Xi_{12}}\ g_{12}; \Xi_1$. Then, by Proposition 71, $\varepsilon_2' \triangleright [\Xi_x/x]g_2; [\Xi_x/x]\Xi_2 \mathrel{\widetilde{<:}} g'; \Xi' \curlyvee \Xi_1$
- We know that $t_2 \in \mathbb{T}[g_2; \Xi_2]$. Then, by Lemma 73, $[\Xi_x/x]t_2 \in \mathbb{T}[[\Xi_x/x]g_2; [\Xi_x/x]\Xi_2]$.
- We know that $\varepsilon_1 \triangleright g_{11}''; \Xi_{11}'' \mathrel{\widetilde{<:}} g_{11}'; \Xi_{11}'$. Also, by inversion lemmas, $\mathit{ileft}(\varepsilon_1) \triangleright g_{11}'; \varnothing + \Xi_{11}' \mathrel{\widetilde{<:}} g_{11}; \Xi_1 + \Xi_{11}$. Then $\varepsilon_1' \triangleright g_{11}''; \Xi_{11}'' \mathrel{\widetilde{<:}} g_{11}; \Xi_1 + \Xi_{11}$.
- We want to prove that $[\Xi_x/x]t_2 \vdash \gamma_{ext}$: We know that $t \vdash \gamma$, then (1) $FV(t) \subseteq dom(\gamma)$ and (2) $\forall z^{g_z;\Xi_z} \in FV(t).\gamma(z) \in \mathbb{T}[g_z; \Xi_z]$. We also know that $FV([\Xi_x/x]t_2) = FV(t) \cup \{x\}$, then $FV([\Xi_x/x]t_2) \subseteq dom(\gamma) \cup \{x\} = dom(\gamma_{ext})$. Since $x^{g_{11};\Xi_x}$, we only need to prove that the value provided for x in γ_{ext} is well-typed:

$$(\text{IG}\textsc{ascr})\ \frac{u \in \mathbb{T}[g_{11}''; \Xi_{11}''] \qquad \varepsilon_1' \triangleright g_{11}''; \Xi_{11}'' \mathrel{\widetilde{<:}} g_{11}; \Xi_x}{\varepsilon_1' u :: g_{11}; \Xi_x \in \mathbb{T}[g_{11}; \Xi_x]}$$

Finally,

$$(\text{IG}\textsc{ascr})\ \frac{D_3 \qquad \varepsilon_2' \triangleright [\Xi_x/x]g_2; [\Xi_x/x]\Xi_2 \mathrel{\widetilde{<:}} g'; \Xi' \curlyvee \Xi_1}{\varepsilon_2' \mathsf{ctx}(\gamma_{ext}, t_{body}) :: g'; \Xi' \curlyvee \Xi_1 \in \mathbb{T}[g'; \Xi' \curlyvee \Xi_1]}$$

$$D_3 = (\text{IG}\textsc{ctx})\ \frac{[\Xi_x/x]t_2 \in \mathbb{T}[g_2; \Xi_2] \qquad [\Xi_x/x]t_2 \vdash \gamma_{ext}}{\mathsf{ctx}(\gamma_{ext}, t_{body}) \in \mathbb{T}[[\Xi_x/x]g_2; [\Xi_x/x]\Xi_2]}$$

And the result holds.

Case (IGr-case-2). Analogous to (IGr-case-1).

Case (IGr-ctx).

Then, $t^{g;\Xi} = \varepsilon\mathsf{ctx}(\gamma', v) :: g; \Xi$. Then,

$$(\text{I}G\textsc{ascr})\ \dfrac{(\text{I}G\textsc{ctx})\ \dfrac{v \in \mathbb{T}[g'; \Xi'] \qquad v \vdash \gamma'}{\mathsf{ctx}(\gamma', v) \in \mathbb{T}[g'; \Xi']} \qquad \varepsilon \triangleright g'; \Xi' \mathrel{\widetilde{<:}} g; \Xi}{\varepsilon\mathsf{ctx}(\gamma', v) :: g; \Xi \in \mathbb{T}[g; \Xi]}$$

Therefore,

$$\varepsilon\mathsf{ctx}(\gamma', v) :: g; \Xi \longrightarrow \varepsilon v :: g; \Xi$$

But,

$$(\text{I}G\textsc{ascr})\ \dfrac{v \in \mathbb{T}[g'; \Xi'] \qquad \varepsilon \triangleright g'; \Xi' \mathrel{\widetilde{<:}} g; \Xi}{\varepsilon v :: g; \Xi \in \mathbb{T}[g; \Xi]}$$

And the result holds.

Case (IGR-ASCR).

Then, $t = (u :: g_1; \Xi_1) :: g_2; \Xi_2$. Then,

$$(\text{I}G\textsc{ascr})\ \dfrac{(\text{I}G\textsc{ascr})\ \dfrac{u \in \mathbb{T}[g_0; \Xi_0] \qquad \varepsilon_1 \triangleright g_0; \Xi_0 \mathrel{\widetilde{<:}} g_1; \Xi_1}{\varepsilon_1 u :: g_1; \Xi_1 \in \mathbb{T}[g_1; \Xi_1]} \qquad \varepsilon_2 \triangleright g_1; \Xi_1 \mathrel{\widetilde{<:}} g_2; \Xi_2}{\varepsilon_2(\varepsilon_1 u :: g_1; \Xi_1) :: g_2; \Xi_2 \in \mathbb{T}[g_2; \Xi_2]}$$

If $\varepsilon_1 \circ^{<:} \varepsilon_2$ is not defined, then $t \longrightarrow$ **error** and the result holds immediately. Suppose that the consistent transitivity holds, then:

$$\varepsilon_2(\varepsilon_1 u :: g_1; \Xi_1) :: g_2; \Xi_2 \longrightarrow \varepsilon_1 \circ^{<:} \varepsilon_2 u :: g_2; \Xi_2$$

But,

$$(\text{I}G\textsc{ascr})\ \dfrac{u \in \mathbb{T}[g_0; \Xi_0] \qquad \varepsilon_1 \circ^{<:} \varepsilon_2 \triangleright g_0; \Xi_0 \mathrel{\widetilde{<:}} g_2; \Xi_2}{\varepsilon_1 \circ^{<:} \varepsilon_2 u :: g_2; \Xi_2 \in \mathbb{T}[g_2; \Xi_2]}$$

And the result holds.

$\square$

Proposition 82 ($\longmapsto$ is well-defined)**.** *If* $t \vdash \gamma$, $t \in \mathbb{T}[G]$ *and* $t \xrightarrow{\gamma}\!\!\!\!\!\!\mapsto t'$, *then* $t' \in \mathbb{T}[G] \cup \{\, \mathbf{error} \,\}$.

Proof. By induction on the structure of derivation of $t \overset{\gamma}{\longmapsto} t'$.

Case (IGR$\longrightarrow$)
By Proposition 81, $r \in \mathbb{T}[g; \Xi] \cup \{\, \mathbf{error} \,\}$.

Case (IGRE)
$t = E[t_1]$, $E[t_1] \in \mathbb{T}[g; \Xi]$, $t_1 \longmapsto t_2$, $t_1 \in \mathbb{T}[g'; \Xi']$, and $E : g'; \Xi' \to g; \Xi$. By induction hypothesis, $t_2 \in \mathbb{T}[g; \Xi]$, so $E[t_2] \in \mathbb{T}[g; \Xi]$.

Case (IGRctx)

$t = \varepsilon\mathsf{ctx}(\gamma', t_1) :: g; \Xi$, $t_1 \overset{\gamma'}{\longmapsto} t_2$. We know,

$$\text{(IGAscr)}\ \dfrac{\text{(IGctx)}\ \dfrac{t_1 \in \mathbb{T}[g'; \Xi'] \qquad \gamma' \vdash t_1}{\mathsf{ctx}(\gamma', t_1) \in \mathbb{T}[g'; \Xi']} \qquad \varepsilon \,\triangleright\, g'; \Xi' \mathrel{\widetilde{<:}} g; \Xi}{\varepsilon\mathsf{ctx}(\gamma', t_1) :: g; \Xi \in \mathbb{T}[g; \Xi]}$$

By induction hypothesis $t_2 \in \mathbb{T}[g'; \Xi']$, then

$$\text{(IGAscr)}\ \dfrac{\text{(IGctx)}\ \dfrac{t_2 \in \mathbb{T}[g'; \Xi'] \qquad \gamma' \vdash t_2}{\mathsf{ctx}(\gamma', t_2) \in \mathbb{T}[g'; \Xi']} \qquad \varepsilon \,\triangleright\, g'; \Xi' \mathrel{\widetilde{<:}} g; \Xi}{\varepsilon\mathsf{ctx}(\gamma', t_2) :: g; \Xi \in \mathbb{T}[g; \Xi]}$$

Case (IGR$\longrightarrow$err), (IGREerr), (IGRctxerr). $r = \mathbf{error}$

□

Proposition 83. *If* $t \in \mathbb{T}[g; \Xi]$, *then one of the following is true:*

- t *is a value* v.
- *if* $\gamma \vdash t$ *then* $t \overset{\gamma}{\longmapsto} t'$ *for some* $t' \in \mathbb{T}[g; \Xi]$.
- $t \overset{\gamma}{\longmapsto} \mathbf{error}$

Proof. By induction on the structure of t.

Case $\varepsilon r :: g'; \Xi', \varepsilon\langle t', \gamma' \rangle :: g'; \Xi, \varepsilon\mathsf{tt} :: g'; \Xi'$: t is a value.

Case (IGinl), (IGinr). $t = \varepsilon \mathsf{inl}^{g_1} t' :: G'$: We only show the proof inl as both cases are analogous. We know that,

$$\text{(IGascr)}\ \dfrac{\text{(IGinl)}\ \dfrac{t' \in \mathbb{T}[g_1; \Xi]}{\mathsf{inl}^{g_2} t \in \mathbb{T}[g_1 \ {}^{\Xi}\!\oplus^{\varnothing}\ g_2; \varnothing]} \qquad \varepsilon \vartriangleright g_1 \ {}^{\Xi}\!\oplus^{\varnothing}\ g_2; \varnothing \mathrel{\widetilde{<:}} G'}{\varepsilon \mathsf{inl}^{g_1} t' :: G' \in \mathbb{T}[G']}$$

By induction hypothesis,

– t' is a value v, then t' is a value.

– $t' \longmapsto t''$ for some $t'' \in \mathbb{T}[g_1; \Xi] \cup \{\, \mathbf{error} \,\}$. By Proposition 82 and either (IGRE) or (IGREErr).

Case (IGascr). $t = \varepsilon t' :: G$

$$\text{(IGascr)}\ \dfrac{t' \in \mathbb{T}[G'] \qquad \varepsilon \vartriangleright G' \mathrel{\widetilde{<:}} G}{\varepsilon t' :: G \in \mathbb{T}[G]}$$

By induction hypothesis,

– t' is a value v, then the result holds by Proposition 82 and either (IGR$\rightarrow$) or (IGR$\rightarrow$err).

– $t' \longmapsto t''$ for some $t' \in \mathbb{T}[G'] \cup \{\, \mathbf{error} \,\}$. By Proposition 82 and either (IGRE) or (IGREErr).

Case (IGplus). $t = t_1 + t_2$, and

$$\text{(IGplus)}\ \dfrac{t_1 \in \mathbb{T}[\mathbb{R}; \Xi_1] \qquad t_2 \in \mathbb{T}[\mathbb{R}; \Xi_2]}{t_1 + t_2 \in \mathbb{T}[\mathbb{R}; \Xi_1 + \Xi_2]}$$

By induction hypothesis on t_1 and t_2:

– If t_1 and t_2 are values $\varepsilon_1 u_1 :: \mathbb{R}; \Xi_1$ and $\varepsilon_2 u_2 :: \mathbb{R}; \Xi_2$, respectively, then by Lemma 80 $u_1 = r_1$ and $u_2 = r_2$ and the result holds by Proposition 82 and (IGR$\rightarrow$).

– If $t_1 \longmapsto t_1'$, then the result holds by Proposition 82 and either (IGRE) or (IGREErr).

– If $t_2 \longmapsto t_2'$, then the result holds by Proposition 82 and either (IGRE) or (IGREErr).

Case (IGleq). Analogous to (IGplus).

Case (IGvar). $t = x$. We know that $x \vdash \gamma$, therefore $x \in dom(\gamma)$. Then, the result holds by Proposition 82 and (IGR$\rightarrow$).

Case (IGlam). $t = \varepsilon \lambda x.t :: (x : g_1) \xrightarrow{\Xi} g_2$. The result holds by Proposition 82 and (IGR$\rightarrow$).

Case (IG<span>APP</span>). $t = t_1\ t_2$, and

$$(\text{IGAPP})\ \frac{t_1 \in \mathbb{T}[(x : g_1) \xrightarrow{\Xi_2} g_2; \Xi] \qquad t_2 \in \mathbb{T}[g_1; \Xi_1]}{t_1\ t_2 \in \mathbb{T}[[\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi_2]}$$

By induction hypothesis on t_1 and t_2:

- If t_1 and t_2 are values $\varepsilon_1 u_1 :: \mathbb{R}; \Xi_1$ and $\varepsilon_2 u_2 :: \mathbb{R}; \Xi_2$, respectively, then by Lemma 80 $u = \langle \lambda x^{g_1'}.t, \gamma' \rangle$ and the result holds by Proposition 82 and either (IGR→) or (IGR→ERR).
- If $t_1 \longmapsto t_1'$, then the result holds by Proposition 82 and either (IGRE) or (IGREERR).
- If $t_2 \longmapsto t_2'$, then the result holds by Proposition 82 and either (IGRE) or (IGREERR).

Case (IGCASE). Analogous to (IGAPP).

□

Proposition 30 (Type safety). *If $t \in \mathbb{T}[g; \Xi]$, then one of the following is true:*

- *t is a value v.*
- *$t \xmapsto{\varnothing} t'$ for some $t' \in \mathbb{T}[g; \Xi]$.*
- *$t \xmapsto{\varnothing}$ **error***

Proof. It follows directly as a corollary of Proposition 83. □

B.4. Gradual Guarantee

Proposition 25 (Equivalence for fully-static expressions). *Let e be a fully-static expression and G a static type ($G = \mathrm{T}$). $\cdot \vdash_s e : \mathrm{T}$ if and only if $\cdot \vdash e : \mathrm{T}$.*

Proof. By induction on the typing derivations. The proof is trivial because static sensitivities are given one-point intervals meanings, i.e. $[s, s]$, via concretization. □

B.4.1. Static Gradual Guarantee

Proposition 84. *If $\Gamma \vdash e : G$ and $\Gamma \sqsubseteq \Gamma'$, then $\Gamma' \vdash e : G'$ where $G \sqsubseteq G'$.*

Proof. By simple induction on the typing derivation. □

Proposition 26 (Static gradual guarantee)**.** *Let e_1 and e_2 be two closed expressions such that $e_1 \sqsubseteq e_2$ and $\cdot \vdash e_1 : G_1$. Then, $\cdot \vdash e_2 : G_2$ and $G_1 \sqsubseteq G_2$.*

Proof. We prove the property on open expressions instead: If $\Gamma \vdash e_1 : G_1$ and $e_1 \sqsubseteq e_2$, then $\Gamma \vdash e_2 : G$ and $G_1 \sqsubseteq G_2$.

We proceed by induction on the typing derivation.

Case $(G\textsc{RLit})$, $(G\textsc{Unit})$. Trivial by Definitions $(G\sqsubseteq_{\mathbb{R}})$ and $(G\sqsubseteq_{\mathsf{unit}})$, respectively.

Case $(G\textsc{Plus})$.

We know that $e_1 = e_{11} + e_{12}$ and, by $(G\textsc{Plus})$,

$$(G\textsc{Plus})\ \frac{\Gamma \vdash e_{11} : \mathbb{R}; \Xi_{11} \qquad \Gamma \vdash e_{12} : \mathbb{R}; \Xi_{12}}{\Gamma \vdash e_1 : \mathbb{R}; \Xi_{11} + \Xi_{12}}$$

Then, there exists $e_2 = e_{21} + e_{22}$ such that:

$$(G\sqsubseteq_{+})\ \frac{e_{11} \sqsubseteq e_{21} \qquad e_{12} \sqsubseteq e_{22}}{e_1 \sqsubseteq e_2}$$

We know that:

– By induction hypothesis, $\Gamma \vdash e_{21} : \mathbb{R}; \Xi_{21}$ and $\mathbb{R}; \Xi_{11} \sqsubseteq \mathbb{R}; \Xi_{21}$.

– By induction hypothesis, $\Gamma \vdash e_{22} : \mathbb{R}; \Xi_{22}$ and $\mathbb{R}; \Xi_{12} \sqsubseteq \mathbb{R}; \Xi_{22}$.

– By $(G\textsc{Plus})$, $\Gamma \vdash e_2 : \mathbb{R}; \Xi_{21} + \Xi_{22}$.

– By Proposition 37, $\Xi_{11} + \Xi_{12} \sqsubseteq \Xi_{21} + \Xi_{22}$. Then, $\mathbb{R}; \Xi_{11} + \Xi_{12} \sqsubseteq \mathbb{R}; \Xi_{21} + \Xi_{22}$.

Finally, the result holds.

Case $(G\textsc{Leq})$. Analogous to $(G\textsc{Plus})$.

Case $(G\textsc{Var})$. We know that $e_1 = x$. By $(G\textsc{Var})$, $\Gamma(x) = G_1$. Then, by inspection on $(G\sqsubseteq_x)$, $e_2 = x$ and the result trivially holds by $(G\textsc{Var})$.

Case $(G\textsc{Lam})$.

We know that $e_1 = \lambda(x : g_{11}).e_{12}$ and, by $(G\textsc{Lam})$,

$$(G\textsc{Lam})\ \frac{\Gamma, x : g_{11}; x \vdash e_{12} : g_{12}; \Xi_{12}}{\Gamma \vdash \lambda(x : g_{11}).e_{12} : (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}; \varnothing}$$

Then, there exists $e_2 = \lambda(x : g_{21}).e_{22}$ such that:

$$(G{\sqsubseteq_\lambda})\ \frac{g_{11} \sqsubseteq g_{21} \qquad e_{12} \sqsubseteq e_{22}}{\lambda(x : g_{11}).e_{12} \sqsubseteq \lambda(x : g_{21}).e_{22}}$$

We know that:

- By induction hypothesis, $\Gamma, x : g_{11}; x \vdash e_{22} : g_{22}; \Xi_{22}$ and $g_{12}; \Xi_{12} \sqsubseteq g_{22}; \Xi_{22}$.
- By Proposition 84, $\Gamma, x : g_{21}; x \vdash e_{22} : g'_{22}; \Xi'_{22}$ and $g_{22}; \Xi_{22} \sqsubseteq g'_{22}; \Xi'_{22}$. Then, $g_{12}; \Xi_{12} \sqsubseteq g'_{22}; \Xi'_{22}$.
- By (GLAM), $\Gamma \vdash e_2 : (x : g_{21}) \xrightarrow{\Xi'_{22}} g'_{22}; \varnothing$.
- By definition of $\sqsubseteq$ for arrow types, $(x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}; \varnothing \sqsubseteq (x : g_{21}) \xrightarrow{\Xi'_{22}} g'_{22}; \varnothing$.

Finally, the result holds.

Case (GAPP).

We know that $e_1 = e_{11}\ e_{12}$ and, by (GAPP),

$$(\text{GAPP})\ \frac{\Gamma \vdash e_{11} : (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}; \Xi_1 \qquad \Gamma \vdash e_{12} : g'_{11}; \Xi_{11} \qquad g'_{11} \mathrel{\widetilde{<:}} g_{11}}{\Gamma \vdash e_{11}\ e_{12} : [\Xi_{11}/x]g_{12}; \Xi_1 + [\Xi_{11}/x]\Xi_{12}}$$

Then, there exists $e_2 = e_{21}\ e_{22}$ such that:

$$(G{\sqsubseteq_@})\ \frac{e_{11} \sqsubseteq e_{21} \qquad e_{12} \sqsubseteq e_{22}}{e_{11}\ e_{12} \sqsubseteq e_{21}\ e_{22}}$$

We know that:

- By induction hypothesis, $\Gamma \vdash e_{21} : (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22}; \Xi_2$ and $(x : g_{11}) \xrightarrow{\Xi_{12}} g_{12}; \Xi_1 \sqsubseteq (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22}; \Xi_2$. Then, by Definition of precision of arrow types, $g_{11} \sqsubseteq g_{21}$, $g_{12} \sqsubseteq g_{22}$ and $\Xi_{12} \sqsubseteq \Xi_{22}$.
- By induction hypothesis, $\Gamma \vdash e_{22} : g'_{21}; \Xi_{21}$ and $g'_{11}; \Xi_{11} \sqsubseteq g'_{21}; \Xi_{21}$.
- By Proposition 65, $g'_{21} \mathrel{\widetilde{<:}} g_{21}$.
- By (GAPP), $\Gamma \vdash e_2 : [\Xi_{21}/x]g_{22}; \Xi_2 + [\Xi_{21}/x]\Xi_{22}$.
- By Propositions 47, 45 and 37, $[\Xi_{11}/x]g_{12}; \Xi_1 + [\Xi_{11}/x]\Xi_{12} \sqsubseteq [\Xi_{21}/x]g_{22}; \Xi_2 + [\Xi_{21}/x]\Xi_{22}$.

Finally, the result holds.

Case (GINL).

We know that $e_1 = \mathsf{inl}^{g_{12}} e_{11}$ and, by (GINL),

$$(\text{GINL})\ \frac{\Gamma \vdash e_{11} : g_{11}; \Xi_{11}}{\Gamma \vdash \mathsf{inl}^{g_{12}} e_{11} : g_{11} \ {}^{\Xi_{11}}\!\oplus^{\varnothing}\ g_{12}; \varnothing}$$

Then, there exists $e_2 = \mathsf{inl}^{g_{22}} e_{21}$ such that:

$$(G\sqsubseteq_{\mathsf{inl}})\ \frac{e_{11} \sqsubseteq e_{21} \qquad g_{12} \sqsubseteq g_{22}}{\mathsf{inl}^{g_{12}} e_{11} \sqsubseteq \mathsf{inl}^{g_{22}} e_{21}}$$

We know that:

- By induction hypothesis, $\Gamma \vdash e_{21} : g_{21}; \Xi_{21}$ and $g_{11}; \Xi_{11} \sqsubseteq g_{21}; \Xi_{21}$.
- By definition of precision of sum types, $g_{11} \ {}^{\Xi_{11}}\!\oplus^{\varnothing}\ g_{12}; \varnothing \sqsubseteq g_{21} \ {}^{\Xi_{21}}\!\oplus^{\varnothing}\ g_{22}; \varnothing$.

Finally, the result holds.

Case $(G\textsc{inr})$. Analogous to $(G\textsc{inl})$.

Case $(G\textsc{case})$.

We know that $e_1 = \mathsf{case}\ e_{11}\ \mathsf{of}\ \{\, x \Rightarrow e_{12}\,\}\ \{\, y \Rightarrow e_{13}\,\}$ and, by $(G\textsc{case})$,

$$(G\textsc{case})\ \frac{\begin{array}{c}\Gamma \vdash e_{11} : g_{111} \ {}^{\Xi_{111}}\!\oplus^{\Xi_{112}}\ g_{112}; \Xi_{11} \\ \Gamma, x : g_{111}; x \vdash e_{12} : g_{12}; \Xi_{12} \qquad \Gamma, y : g_{112}; y \vdash e_{13} : g_{13}; \Xi_{13}\end{array}}{\begin{array}{c}\Gamma \vdash \mathsf{case}\ e_{11}\ \mathsf{of}\ \{\, x \Rightarrow e_{12}\,\}\ \{\, y \Rightarrow e_{13}\,\} : \\ [\Xi_{11} + \Xi_{111}/x]g_{12} \curlyvee [\Xi_{11} + \Xi_{112}/y]g_{13}; \Xi_{11} \curlyvee [\Xi_{11} + \Xi_{111}/x]\Xi_{12} \curlyvee [\Xi_{11} + \Xi_{112}/y]\Xi_{13}\end{array}}$$

Then, there exists $e_2 = \mathsf{case}\ e_{21}\ \mathsf{of}\ \{\, x \Rightarrow e_{22}\,\}\ \{\, y \Rightarrow e_{23}\,\}$ such that:

$$(G\sqsubseteq_{\mathsf{case}})\ \frac{e_{11} \sqsubseteq e_{21} \qquad e_{12} \sqsubseteq e_{22} \qquad e_{13} \sqsubseteq e_{23}}{\mathsf{case}\ e_{11}\ \mathsf{of}\ \{\, x \Rightarrow e_{12}\,\}\ \{\, y \Rightarrow e_{13}\,\} \sqsubseteq \mathsf{case}\ e_{21}\ \mathsf{of}\ \{\, x \Rightarrow e_{22}\,\}\ \{\, y \Rightarrow e_{23}\,\}}$$

We know that:

- By induction hypothesis, $\Gamma \vdash e_{21} : g_{211} {}^{\Xi_{211}}\!\oplus^{\Xi_{212}} g_{212}; \Xi_{21}$ and $g_{111} {}^{\Xi_{111}}\!\oplus^{\Xi_{112}} g_{112}; \Xi_{11} \sqsubseteq g_{211} \ {}^{\Xi_{211}}\!\oplus^{\Xi_{212}}\ g_{212}; \Xi_{21}$.
- By induction hypothesis, $\Gamma, x : g_{111}; x \vdash e_{22} : g_{22}; \Xi_{22}$ and $g_{12}; \Xi_{12} \sqsubseteq g_{22}; \Xi_{22}$.
- By induction hypothesis, $\Gamma, y : g_{112}; y \vdash e_{23} : g_{23}; \Xi_{23}$ and $g_{13}; \Xi_{13} \sqsubseteq g_{23}; \Xi_{23}$.
- By Proposition 84, $\Gamma, x : g_{211}; x \vdash e_{22} : g'_{22}; \Xi'_{22}$ and $g_{22}; \Xi_{22} \sqsubseteq g'_{22}; \Xi'_{22}$. Then, $g_{12}; \Xi_{12} \sqsubseteq g'_{22}; \Xi'_{22}$.
- By Proposition 84, $\Gamma, y : g_{212}; y \vdash e_{23} : g'_{23}; \Xi'_{23}$ and $g_{23}; \Xi_{23} \sqsubseteq g'_{23}; \Xi'_{23}$. Then, $g_{13}; \Xi_{13} \sqsubseteq g'_{23}; \Xi'_{23}$.
- By $(G\textsc{case})$, $\Gamma \vdash \mathsf{case}\ e_{21}\ \mathsf{of}\ \{\, x \Rightarrow e_{22}\,\}\ \{\, y \Rightarrow e_{23}\,\} : [\Xi_{21} + \Xi_{211}/x]g'_{22} \curlyvee [\Xi_{21} + \Xi_{212}/y]g'_{23}; \Xi_{21} \curlyvee [\Xi_{21} + \Xi_{211}/x]\Xi'_{22} \curlyvee [\Xi_{21} + \Xi_{212}/y]\Xi'_{23}$.
- By Propositions 37, 46 and 40, $[\Xi_{11} + \Xi_{111}/x]g_{12} \curlyvee [\Xi_{11} + \Xi_{112}/y]g_{13} \sqsubseteq [\Xi_{21} + \Xi_{211}/x]g'_{22} \curlyvee [\Xi_{21} + \Xi_{212}/y]g'_{23}$.
- By Propositions 37, 45 and 39, $\Xi_{11} \curlyvee [\Xi_{11} + \Xi_{111}/x]\Xi_{12} \curlyvee [\Xi_{11} + \Xi_{112}/y]\Xi_{13} \sqsubseteq \Xi_{21} \curlyvee [\Xi_{21} + \Xi_{211}/x]\Xi'_{22} \curlyvee [\Xi_{21} + \Xi_{212}/y]\Xi'_{23}$.

Finally, the result holds.

Case $(G\text{ASCR})$.

We know that $e_1 = e_{11} :: G_1$ and, by $(G\text{ASCR})$,

$$(G\text{ASCR})\ \frac{\Gamma \vdash e_{11} \ :\ G_{11} \qquad G_{11} \mathrel{\widetilde{<:}} G_1}{\Gamma \vdash e_{11} :: G_1 \ :\ G_1}$$

Then, there exists $e_2 = e_{21} :: G_2$ such that:

$$(G\sqsubseteq_{::})\ \frac{e_{11} \sqsubseteq e_{21} \qquad G_1 \sqsubseteq G_2}{e_{11} :: G_1 \sqsubseteq e_{21} :: G_2}$$

We know that:

– By induction hypothesis, $\Gamma \vdash e_{21} \ :\ G_{21}$ and $G_{11} \sqsubseteq G_{21}$.
– By Proposition 66, $G_{21} \mathrel{\widetilde{<:}} G_2$.
– By $(G\text{ASCR})$, $\Gamma \vdash e_{21} :: G_2 \ :\ G_2$.

Finally, the result holds.

Case $(G\text{CLOSURE})$.

We know that $e_1 = \langle e_{11}, \gamma_1' \rangle$ and, by $(G\text{CLOSURE})$,

$$(G\text{CLOSURE})\ \frac{\exists \Gamma_1' : \gamma_1' \vdash \Gamma_1' \qquad \Gamma_1' \vdash e_{11} \ :\ G_1}{\Gamma \vdash \langle e_{11}, \gamma_1' \rangle \ :\ G_1}$$

Then, there exists $e_2 = \langle e_{12}, \gamma_2' \rangle$ such that:

$$(G\sqsubseteq_{\langle \lambda, \gamma \rangle})\ \frac{e_{11} \sqsubseteq e_{12} \qquad \gamma_1' \sqsubseteq \gamma_2'}{\langle e_{11}, \gamma_1' \rangle \sqsubseteq \langle e_{12}, \gamma_2' \rangle}$$

We know that:

– By definition of type environment and substitution precision, $dom(\gamma_1') = dom(\gamma_2')$.
– By induction hypothesis, $\Gamma_1' \vdash e_{12} \ :\ G_2$ and $G_1 \sqsubseteq G_2$.
– From the definition of type environment well-formedness, for any $x_i \in dom(\gamma_1')$, $\cdot \vdash \gamma_1'(x_i) \ :\ \Gamma_1'(x_i)$. Then, by induction hypothesis, $\cdot \vdash \gamma_2'(x_i) \ :\ G_{2i}$ where $\Gamma_1'(x_i) \sqsubseteq G_{2i}$. Let us call Γ_2' the type environment where $\Gamma_2'(x_i) = G_{2i}$.
– By construction, $\gamma_2' \vdash \Gamma_2'$.
– By Proposition 84, $\Gamma_2' \vdash e_{12} \ :\ G_2'$ and $G_2 \sqsubseteq G_2'$. Then $G_1 \sqsubseteq G_2'$.
– By choosing Γ_2' and $(G\text{ASCR})$, $\Gamma \vdash \langle e_{12}, \gamma_2' \rangle \ :\ G_2'$.

Finally, the result holds.

Case (GCTX). Analogous to (GCLOSURE).

$\square$

B.4.2. Dynamic Gradual Guarantee

Proposition 85 (Dynamic gradual guarantee for $\longrightarrow$)**.** *Suppose that* $t_{11} \sqsubseteq t_{12}$, *and* $\gamma_1 \sqsubseteq \gamma_2$. *If* $t_{11} \xrightarrow{\gamma_1} t_{21}$ *then* $t_{12} \xrightarrow{\gamma_2} t_{22}$ *where* $t_{21} \sqsubseteq t_{22}$.

Proof. Let $t_{11} = t_1^{g_1;\Xi_1}, t_{12} = t_1^{g_2;\Xi_2}, t_{21} = t_2^{g_1;\Xi_1}, t_{22} = t_2^{g_2;\Xi_2}$. By induction on the definition of $t_1^{g_1;\Xi_1} \longrightarrow t_2^{g_1;\Xi_1}$.

Case (IGR-PLUS).

We know that $t_1^{g_1;\Xi_1} = \varepsilon_{11} r_1 :: \mathbb{R};\Xi_{11} + \varepsilon_{12} r_2 :: \mathbb{R};\Xi_{12}$, then there exist $\varepsilon_{21}, \varepsilon_{22}, \Xi_{21}, \Xi_{22}$ such that $t_1^{g_2;\Xi_2} = \varepsilon_{21} r_1 :: \mathbb{R};\Xi_{21} + \varepsilon_{22} r_2 :: \mathbb{R};\Xi_{22}$ and:

$$\sqsubseteq_{+}\dfrac{\sqsubseteq_{::}\dfrac{\varepsilon_{11} \sqsubseteq \varepsilon_{21} \qquad \ldots \qquad \mathbb{R};\Xi_{11} \sqsubseteq \mathbb{R};\Xi_{21}}{\varepsilon_{11} r_1 :: \mathbb{R};\Xi_{11} \sqsubseteq \varepsilon_{21} r_1 :: \mathbb{R};\Xi_{21}} \qquad \sqsubseteq_{::}\dfrac{\varepsilon_{12} \sqsubseteq \varepsilon_{22} \qquad \ldots \qquad \mathbb{R};\Xi_{12} \sqsubseteq \mathbb{R};\Xi_{22}}{\varepsilon_{12} r_2 :: \mathbb{R};\Xi_{12} \sqsubseteq \varepsilon_{22} r_2 :: \mathbb{R};\Xi_{22}}}{\varepsilon_{11} r_1 :: \mathbb{R};\Xi_{11} + \varepsilon_{12} r_2 :: \mathbb{R};\Xi_{12} \sqsubseteq \varepsilon_{21} r_1 :: \mathbb{R};\Xi_{21} + \varepsilon_{22} r_2 :: \mathbb{R};\Xi_{22}}$$

If $t_1^{g_1;\Xi_1} \longrightarrow \varepsilon_{11} +_{\Xi} \varepsilon_{12} r_3 :: \mathbb{R};\Xi_{11} + \Xi_{12}$ where $r_3 = r_1 [\![+]\!] r_2$, then:

1. By Proposition 54, $\varepsilon_{21} +_{\Xi} \varepsilon_{22}$ is defined and $\varepsilon_{11} +_{\Xi} \varepsilon_{12} \sqsubseteq \varepsilon_{21} +_{\Xi} \varepsilon_{22}$.
2. By Proposition 39, $\mathbb{R};\Xi_{11} + \Xi_{12} \sqsubseteq \mathbb{R};\Xi_{21} + \Xi_{22}$.
3. $t_1^{g_2;\Xi_2} \longrightarrow \varepsilon_{21} +_{\Xi} \varepsilon_{22} r_3 :: \mathbb{R};\Xi_{21} + \Xi_{22}$.

Finally,

$$\sqsubseteq_{::}\dfrac{\varepsilon_{11} +_{\Xi} \varepsilon_{12} \sqsubseteq \varepsilon_{21} +_{\Xi} \varepsilon_{22} \qquad r_3 \sqsubseteq r_3 \qquad \mathbb{R};\Xi_{11} + \Xi_{12} \sqsubseteq \mathbb{R};\Xi_{21} + \Xi_{22}}{\varepsilon_{11} +_{\Xi} \varepsilon_{12} r_3 :: \mathbb{R};\Xi_{11} + \Xi_{12} \sqsubseteq \varepsilon_{21} +_{\Xi} \varepsilon_{22} r_3 :: \mathbb{R};\Xi_{21} + \Xi_{22}}$$

and the result holds.

Case (IGR-LEQ). Analogous to (IGR-PLUS).

Case (IGR-VAR). We know that $t_1^{g_1;\Xi_1} = x^{g_1;\Xi_1}$, then by (IG$\sqsubseteq_x$) $t_1^{g_2;\Xi_2} = x^{g_2;\Xi_2}$ where $g_1;\Xi_1 \sqsubseteq g_2;\Xi_2$. If $t_1^{g_1;\Xi_1} \xrightarrow{\gamma_1} \gamma_1(x)$, then by hypothesis ($\gamma_1 \sqsubseteq \gamma_2$) we know that $t_1^{g_2;\Xi_2} \xrightarrow{\gamma_2} \gamma_2(x)$ where, in particular, $\gamma_1(x) \sqsubseteq \gamma_2(x)$. Therefore, the result holds immediately.

Case (IGR-LAM).

We know that $t_1^{g_1;\Xi_1} = \varepsilon_1 \lambda x^{g'_{11}}.t_{11} :: (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1$, then there exist ε_2, $(x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2$ such that $t_1^{g_2;\Xi_2} = \varepsilon_2 \lambda x^{g'_{21}}.t_{21} :: (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2$ and:

$$\sqsubseteq_{::}\ \frac{\varepsilon_1 \sqsubseteq \varepsilon_2 \qquad \lambda x^{g'_{11}}.t_{11} \sqsubseteq \lambda x^{g'_{21}}.t_{21} \qquad (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1 \sqsubseteq (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2}{\varepsilon_1 \lambda x^{g'_{11}}.t_{11} :: (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1 \sqsubseteq \varepsilon_2 \lambda x^{g'_{21}}.t_{21} :: (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2}$$

Trivially, $t_1^{g_1;\Xi_1} \xrightarrow{\gamma_1} \varepsilon_1 \langle \lambda x^{g'_{11}}.t_{11}, \gamma_1 \rangle :: (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1$ and $t_1^{g_2;\Xi_2} \xrightarrow{\gamma_2} \varepsilon_2 \langle \lambda x^{g'_{21}}.t_{21}, \gamma_2 \rangle :: (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2$. Finally,

$$\sqsubseteq_{::}\ \frac{\varepsilon_1 \sqsubseteq \varepsilon_2 \qquad \sqsubseteq_{\langle \lambda,\gamma \rangle}\ \dfrac{\gamma_1 \sqsubseteq \gamma_2 \qquad \lambda x^{g'_{11}}.t_{11} \sqsubseteq \lambda x^{g'_{21}}.t_{21}}{\langle \lambda x^{g'_{11}}.t_{11}, \gamma_1 \rangle \sqsubseteq \langle \lambda x^{g'_{21}}.t_{21}, \gamma_2 \rangle} \qquad (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1 \sqsubseteq (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2}{\varepsilon_1 \langle \lambda x^{g'_{11}}.t_{11}, \gamma_1 \rangle :: (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1 \sqsubseteq \varepsilon_2 \langle \lambda x^{g'_{21}}.t_{21}, \gamma_2 \rangle :: (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2}$$

And the result holds.

Case (IGR-APP).

We know that $t_1^{g_1;\Xi_1} = (\varepsilon_{11} \langle \lambda x^{g'_{11}}.t'_1, \gamma'_1 \rangle :: (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1)\ (\varepsilon_{12} u_1 :: g_{11};\Xi_{11})$, then there exist $t_1^{g_2;\Xi_2} = (\varepsilon_{21} \langle \lambda x^{g'_{21}}.t'_2, \gamma'_2 \rangle :: (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2)\ (\varepsilon_{22} u_2 :: g_{21};\Xi_{21})$ such that:

$$\sqsubseteq_{@}\ \frac{D_1 \qquad \sqsubseteq_{::}\ \dfrac{\varepsilon_{12} \sqsubseteq \varepsilon_{22} \qquad u_1 \sqsubseteq u_2 \qquad g_{11};\Xi_{11} \sqsubseteq g_{21};\Xi_{21}}{\varepsilon_{12} u_1 :: g_{11};\Xi_{11} \sqsubseteq \varepsilon_{22} u_2 :: g_{21};\Xi_{21}}}{\begin{array}{c}(\varepsilon_{11} \langle \lambda x^{g'_{11}}.t'_1, \gamma'_1 \rangle :: (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1)\ (\varepsilon_{12} u_1 :: g_{11};\Xi_{11}) \sqsubseteq \\ (\varepsilon_{21} \langle \lambda x^{g'_{21}}.t'_2, \gamma'_2 \rangle :: (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2)\ (\varepsilon_{22} u_2 :: g_{21};\Xi_{21})\end{array}}$$

where

$$D_1 = \sqsubseteq_{::}\ \frac{\varepsilon_{11} \sqsubseteq \varepsilon_{21} \qquad D_2 \qquad \dfrac{g_{11} \sqsubseteq g_{21} \qquad \Xi_{12} \sqsubseteq \Xi_{22} \qquad g_{12} \sqsubseteq g_{22} \qquad \Xi_1 \sqsubseteq \Xi_2}{(x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1 \sqsubseteq (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2}}{\varepsilon_{11} \langle \lambda x^{g'_{11}}.t'_1, \gamma'_1 \rangle :: (x : g_{11}) \xrightarrow{\Xi_{12}} g_{12};\Xi_1 \sqsubseteq \varepsilon_{21} \langle \lambda x^{g'_{21}}.t'_2, \gamma'_2 \rangle :: (x : g_{21}) \xrightarrow{\Xi_{22}} g_{22};\Xi_2}$$

$$D_2 = \sqsubseteq_{\langle \lambda,\gamma \rangle}\ \frac{\gamma'_1 \sqsubseteq \gamma'_2 \qquad \sqsubseteq_{\lambda}\ \dfrac{g'_{11} \sqsubseteq g'_{21} \qquad t'_1 \sqsubseteq t'_2}{\lambda x^{g'_{11}}.t'_1 \sqsubseteq \lambda x^{g'_{21}}.t'_2}}{\langle \lambda x^{g'_{11}}.t'_1, \gamma'_1 \rangle \sqsubseteq \langle \lambda x^{g'_{21}}.t'_2, \gamma'_2 \rangle}$$

If $t_1^{g_1;\Xi_1}$ takes one step:

$$t_1^{g_1;\Xi_1} \xrightarrow{\gamma_1} \varepsilon'_{11} \mathsf{ctx}(\gamma'_{1ext}, t_{1body}) :: [\Xi_{11}/x]g_{12};\Xi_1 + [\Xi_{11}/x]\Xi_{12}$$

where $\gamma'_{1ext} = \gamma'_1[x \mapsto \varepsilon_{12}u_1 :: g_{11}; \Xi_{11}]$, $t_{1body} = [\Xi_{11}/x]t'_1$, $\varepsilon'_{11} = [\Xi_{11}/x]icod(\varepsilon_{11})$, $\varepsilon'_{12} = \varepsilon_{12} \circ^{<:} [\Xi_{11}/x]idom(\varepsilon_{11})$, then:

1. Let $\varepsilon'_{21} = [\Xi_{21}/x]icod(\varepsilon_{21})$. By Propositions 51 and 48, $\varepsilon'_{11} \sqsubseteq \varepsilon'_{21}$.
2. Let $\varepsilon'_{22} = \varepsilon_{22} \circ^{<:} [\Xi_{21}/x]idom(\varepsilon_{21})$. By Propositions 63, 48 and 50, ε'_{22} is defined and $\varepsilon'_{12} \sqsubseteq \varepsilon'_{22}$.
3. By $(IG\sqsubseteq_{::})$, $\varepsilon'_{12}u_1 :: g'_{11}; \Xi_{11} \sqsubseteq \varepsilon'_{22}u_2 :: g'_{21}; \Xi_{21}$.
4. Let $\gamma'_{2ext} = \gamma'_2[x \mapsto \varepsilon_{22}u_2 :: g_{21}; \Xi_{21}]$. By $(IG\sqsubseteq_{\gamma})$, $\gamma'_{1ext} \sqsubseteq \gamma'_{2ext}$.
5. Let $t_{2body} = [\Xi_{21}/x]t'_2$. By Proposition 49, $t_{1body} \sqsubseteq t_{2body}$.
6. By Propositions 46, 45 and 37 $[\Xi_{11}/x]g_{12}; \Xi_1 + [\Xi_{11}/x]\Xi_{12} \sqsubseteq [\Xi_{21}/x]g_{22}; \Xi_2 + [\Xi_{21}/x]\Xi_{22}$.
7. $t_1^{g_2;\Xi_2} \xrightarrow{\gamma_2} \varepsilon'_{21}\mathsf{ctx}(\gamma'_{2ext}, t_{2body}) :: [\Xi_{21}/x]g_{22}; \Xi_2 + [\Xi_{21}/x]\Xi_{22}$

Finally,

$$\sqsubseteq_{::} \frac{\varepsilon'_{11} \sqsubseteq \varepsilon'_{21} \qquad D_3 \qquad [\Xi_{11}/x]g_{12}; \Xi_1 + [\Xi_{11}/x]\Xi_{12} \sqsubseteq [\Xi_{21}/x]g_{22}; \Xi_2 + [\Xi_{21}/x]\Xi_{22}}{\begin{array}{c}\varepsilon'_{11}\mathsf{ctx}(\gamma'_{1ext}, t_{1body}) :: [\Xi_{11}/x]g_{12}; \Xi_1 + [\Xi_{11}/x]\Xi_{12} \sqsubseteq \\ \varepsilon'_{21}\mathsf{ctx}(\gamma'_{2ext}, t_{2body}) :: [\Xi_{21}/x]g_{22}; \Xi_2 + [\Xi_{21}/x]\Xi_{22}\end{array}}$$

where

$$D_3 = \sqsubseteq_{\mathsf{ctx}} \frac{\gamma'_{1ext} \sqsubseteq \gamma'_{2ext} \qquad t_{1body} \sqsubseteq t_{2body}}{\mathsf{ctx}(\gamma'_{1ext}, t_{1body}) \sqsubseteq \mathsf{ctx}(\gamma'_{2ext}, t_{2body})}$$

and the result holds.

Case $(IG\text{R-CASE-1})$.

We know that $t_1^{g_1;\Xi_1} = \mathsf{case}\ v_1\ \mathsf{of}\ \{\, x \Rightarrow t_{12}^{g_{12};\Xi_{12}} \,\}\ \{\, y \Rightarrow t_{13}^{g_{13};\Xi_{13}} \,\}$, where $v_1 = \varepsilon_1\mathsf{inl}^{g'_{112}}(\varepsilon_{11}u_1 :: g'_{11}; \Xi'_{11}) :: g_{111}\ {}^{\Xi_{111}}\!\oplus^{\Xi_{112}}\ g_{112}; \Xi_{11}$.

Then, there exist $t_1^{g_2;\Xi_2} = \mathsf{case}\ v_2\ \mathsf{of}\ \{\, x \Rightarrow t_{22}^{g_{22};\Xi_{22}} \,\}\ \{\, y \Rightarrow t_{23}^{g_{23};\Xi_{23}} \,\}$, where $v_2 = \varepsilon_2\mathsf{inl}^{g'_{212}}(\varepsilon_{21}u_1 :: g'_{21}; \Xi'_{21}) :: g_{211}\ {}^{\Xi_{211}}\!\oplus^{\Xi_{212}}\ g_{212}; \Xi_{21}$, such that:

$$\sqsubseteq_{\mathsf{case}} \frac{D_1 \qquad g_{111} \sqsubseteq g_{211} \qquad t_{12}^{g_{12};\Xi_{12}} \sqsubseteq t_{22}^{g_{22};\Xi_{22}} \qquad g_{112} \sqsubseteq g_{212} \qquad t_{13}^{g_{13};\Xi_{13}} \sqsubseteq t_{23}^{g_{23};\Xi_{23}}}{\begin{array}{c}\mathsf{case}\ \varepsilon_1\mathsf{inl}^{g'_{112}}(\varepsilon_{11}u_1 :: g'_{11}; \Xi'_{11}) :: g_{111}\ {}^{\Xi_{111}}\!\oplus^{\Xi_{112}}\ g_{112}; \Xi_{11}\ \mathsf{of}\ \{\, x \Rightarrow t_{12}^{g_{12};\Xi_{12}} \,\}\ \{\, y \Rightarrow t_{13}^{g_{13};\Xi_{13}} \,\} \sqsubseteq \\ \mathsf{case}\ \varepsilon_2\mathsf{inl}^{g'_{212}}(\varepsilon_{21}u_1 :: g'_{21}; \Xi'_{21}) :: g_{211}\ {}^{\Xi_{211}}\!\oplus^{\Xi_{212}}\ g_{212}; \Xi_{21}\ \mathsf{of}\ \{\, x \Rightarrow t_{22}^{g_{22};\Xi_{22}} \,\}\ \{\, y \Rightarrow t_{23}^{g_{23};\Xi_{23}} \,\}\end{array}}$$

where

$$D_1 = \sqsubseteq_{::} \frac{\varepsilon_1 \sqsubseteq \varepsilon_2 \qquad D_2 \qquad g_{111}\ {}^{\Xi_{111}}\!\oplus^{\Xi_{112}}\ g_{112}; \Xi_{11} \sqsubseteq g_{211}\ {}^{\Xi_{211}}\!\oplus^{\Xi_{212}}\ g_{212}; \Xi_{21}}{\begin{array}{c}\varepsilon_1\mathsf{inl}^{g'_{112}}(\varepsilon_{11}u_1 :: g'_{11}; \Xi'_{11}) :: g_{111}\ {}^{\Xi_{111}}\!\oplus^{\Xi_{112}}\ g_{112}; \Xi_{11} \sqsubseteq \\ \varepsilon_2\mathsf{inl}^{g'_{212}}(\varepsilon_{21}u_1 :: g'_{21}; \Xi'_{21}) :: g_{211}\ {}^{\Xi_{211}}\!\oplus^{\Xi_{212}}\ g_{212}; \Xi_{21}\end{array}}$$

$$D_2 = \sqsubseteq_{\mathsf{inl}} \frac{\sqsubseteq_{::} \dfrac{\varepsilon_{11} \sqsubseteq \varepsilon_{21} \qquad u_1 \sqsubseteq u_1 \qquad g'_{11}; \Xi'_{11} \sqsubseteq g'_{21}; \Xi'_{21}}{\varepsilon_{11}u_1 :: g'_{11}; \Xi'_{11} \sqsubseteq \varepsilon_{21}u_1 :: g'_{21}; \Xi'_{21}}}{\mathsf{inl}^{g'_{112}}(\varepsilon_{11}u_1 :: g'_{11}; \Xi'_{11}) \sqsubseteq \mathsf{inl}^{g'_{212}}(\varepsilon_{21}u_1 :: g'_{21}; \Xi'_{21})}$$

If $t_1^{g_1;\Xi_1} \xrightarrow{\gamma_1} \varepsilon'_{12}\mathsf{ctx}(\gamma_{1ext}, t_{1body}) :: g'_1; \Xi'_1 \sqcup \Xi_{11}$, where $\Xi_{1x} = \Xi_{11} + \Xi_{111}$, $\Xi_{1y} = \Xi_{11} + \Xi_{112}$, $g'_1 = [\Xi_{1x}/x]g_{12} \sqcup [\Xi_{1y}/y]g_{13}$, $\Xi'_1 = [\Xi_{1x}/x]\Xi_{12} \sqcup [\Xi_{1y}/y]\Xi_{13}$, $\varepsilon'_{11} = \varepsilon_{11} \circ^{<:} \mathit{ileft}(\varepsilon_1)$, $\varepsilon'_{12} = \mathcal{I}_{<:}([\Xi_{1x}/x]g_{12}; [\Xi_{1x}/x]\Xi_{12}, g'_1; \Xi'_1) \curlyvee_{\Xi} \varepsilon_1$, $\gamma_{1ext} = \gamma_1[x \mapsto \varepsilon'_{11}u_1 :: g_{111}; \Xi_{1x}]$, $t_{1body} = [\Xi_{1x}/x]t_{12}$. Then:

1. Let $\Xi_{2x} = \Xi_{21} + \Xi_{211}$. By Proposition 37, $\Xi_{1x} \sqsubseteq \Xi_{2x}$.
2. Let $\Xi_{2y} = \Xi_{21} + \Xi_{212}$. By Proposition 37, $\Xi_{1y} \sqsubseteq \Xi_{2y}$.
3. Let $g'_2 = [\Xi_{2x}/x]g_{22} \sqcup [\Xi_{2y}/y]g_{23}$. By Propositions 46 and 40, $g'_1 \sqsubseteq g'_2$.
4. Let $\Xi'_2 = [\Xi_{2x}/x]\Xi_{22} \sqcup [\Xi_{2y}/y]\Xi_{23}$. By Propositions 45 and 39, $\Xi'_1 \sqsubseteq \Xi'_2$.
5. Let $\varepsilon'_{21} = \varepsilon_{21} \circ^{<:} \mathit{ileft}(\varepsilon_2)$. By Propositions 52 and 63, ε'_{21} is defined and $\varepsilon'_{11} \sqsubseteq \varepsilon'_{21}$.
6. By lemma 64, $g_{12}; \Xi_{12} \sqsubseteq g_{22}; \Xi_{22}$ and $g_{13}; \Xi_{13} \sqsubseteq g_{23}; \Xi_{23}$.
7. By Propositions 46 and 45, $[\Xi_{1x}/x]g_{12}; [\Xi_{1x}/x]\Xi_{12} \sqsubseteq [\Xi_{2x}/x]g_{22}; [\Xi_{2x}/x]\Xi_{22}$.
8. By Proposition 59, $\mathcal{I}_{<:}([\Xi_{1x}/x]g_{12}; [\Xi_{1x}/x]\Xi_{12}, g'_1; \Xi'_1) \sqsubseteq \mathcal{I}_{<:}([\Xi_{2x}/x]g_{22}; [\Xi_{2x}/x]\Xi_{22}, g'_2; \Xi'_2)$.
9. Let $\varepsilon'_{22} = \mathcal{I}_{<:}([\Xi_{2x}/x]g_{22}; [\Xi_{2x}/x]\Xi_{22}, g'_2; \Xi'_2) \curlyvee_{\Xi} \varepsilon_2$. By Proposition 55, $\varepsilon'_{12} \sqsubseteq \varepsilon'_{22}$.
10. By (IG$\sqsubseteq_{::}$), $\varepsilon'_{11}u_1 :: g_{111}; \Xi_{1x} \sqsubseteq \varepsilon'_{21}u_1 :: g_{211}; \Xi_{2x}$.
11. Let $\gamma_{2ext} = \gamma_2[x \mapsto \varepsilon'_{21}u_1 :: g_{211}; \Xi_{2x}]$ By (IG$\sqsubseteq_{\gamma}$), $\gamma_{1ext} \sqsubseteq \gamma_{2ext}$.
12. Let $t_{2body} = [\Xi_{2x}/x]t_{22}$. By Proposition 49, $t_{1body} \sqsubseteq t_{2body}$.
13. By Proposition 37, $\Xi'_1 \sqcup \Xi_{11} \sqsubseteq \Xi'_2 \sqcup \Xi_{21}$.

Finally,

$$\sqsubseteq_{::}\dfrac{\varepsilon'_{12} \sqsubseteq \varepsilon'_{22} \qquad \sqsubseteq_{\mathsf{ctx}}\dfrac{\gamma_{1ext} \sqsubseteq \gamma_{2ext} \qquad t_{1body} \sqsubseteq t_{2body}}{\mathsf{ctx}(\gamma_{1ext}, t_{1body}) \sqsubseteq \mathsf{ctx}(\gamma_{2ext}, t_{2body})} \qquad g'_1; \Xi'_1 \sqcup \Xi_{11} \sqsubseteq g'_2; \Xi'_2 \sqcup \Xi_{21}}{\varepsilon'_{12}\mathsf{ctx}(\gamma_{1ext}, t_{1body}) :: g'_1; \Xi'_1 \sqcup \Xi_{11} \sqsubseteq \varepsilon'_{22}\mathsf{ctx}(\gamma_{2ext}, t_{2body}) :: g'_2; \Xi'_2 \sqcup \Xi_{21}}$$

and the result holds.

Case (IGR-CASE-2). Analogous to (IGR-CASE-1).

Case (IGR-CTX)

We know that $t_1^{g_1;\Xi_1} = \varepsilon_1\mathsf{ctx}(\gamma'_1, v_1) :: g_1; \Xi_1$, then there exist $t_1^{g_2;\Xi_2} = \varepsilon_2\mathsf{ctx}(\gamma'_2, v_1) :: g_2; \Xi_2$ such that:

$$\sqsubseteq_{::}\dfrac{\varepsilon_1 \sqsubseteq \varepsilon_2 \qquad \sqsubseteq_{\mathsf{ctx}}\dfrac{\gamma'_1 \sqsubseteq \gamma'_2 \qquad v_1 \sqsubseteq v_1}{\mathsf{ctx}(\gamma'_1, v_1) \sqsubseteq \mathsf{ctx}(\gamma'_2, v_1)} \qquad g_1; \Xi_1 \sqsubseteq g_2; \Xi_2}{\varepsilon_1\mathsf{ctx}(\gamma'_1, v_1) :: g_1; \Xi_1 \sqsubseteq \varepsilon_2\mathsf{ctx}(\gamma'_2, v_1) :: g_2; \Xi_2}$$

If $t_1^{g_1;\Xi_1} \longmapsto \varepsilon_1 v_1 :: g_1; \Xi_1$, then $t_2^{g_1;\Xi_1} \longmapsto \varepsilon_2 v_1 :: g_2; \Xi_2$ and:

$$\sqsubseteq_{::}\dfrac{\varepsilon_1 \sqsubseteq \varepsilon_2 \qquad v_1 \sqsubseteq v_1 \qquad g_1; \Xi_1 \sqsubseteq g_2; \Xi_2}{\varepsilon_1 v_1 :: g_1; \Xi_1 \sqsubseteq \varepsilon_2 v_1 :: g_2; \Xi_2}$$

$\boxed{E \sqsubseteq E}$ **Precision of evaluation contexts**

$$\sqsubseteq_{E1}\ \frac{}{\square \sqsubseteq \square} \qquad \sqsubseteq_{E2}\ \frac{E_1 \sqsubseteq E_2 \qquad t_1 \sqsubseteq t_2}{E_1 + t_1 \sqsubseteq E_2 + t_2} \qquad \sqsubseteq_{E3}\ \frac{v_1 \sqsubseteq v_2 \qquad E_1 \sqsubseteq E_2}{v_1 + E_1 \sqsubseteq v_2 + E_2}$$

$$\sqsubseteq_{E4}\ \frac{E_1 \sqsubseteq E_2 \qquad t_1 \sqsubseteq t_2}{E_1 \leq t_1 \sqsubseteq E_2 \leq t_2} \qquad \sqsubseteq_{E5}\ \frac{v_1 \sqsubseteq v_2 \qquad E_1 \sqsubseteq E_2}{v_1 \leq E_1 \sqsubseteq v_2 \leq E_2} \qquad \sqsubseteq_{E6}\ \frac{E_1 \sqsubseteq E_2 \qquad t_1 \sqsubseteq t_2}{E_1\ t_1 \sqsubseteq E_2\ t_2}$$

$$\sqsubseteq_{E7}\ \frac{v_1 \sqsubseteq v_2 \qquad E_1 \sqsubseteq E_2}{v_1\ E_1 \sqsubseteq v_2\ E_2} \qquad \sqsubseteq_{E8}\ \frac{g_{12} \sqsubseteq g_{22} \qquad E_1 \sqsubseteq E_2}{\mathsf{inl}^{g_{12}} E_1 \sqsubseteq \mathsf{inl}^{g_{22}} E_2} \qquad \sqsubseteq_{E9}\ \frac{g_{11} \sqsubseteq g_{21} \qquad E_1 \sqsubseteq E_2}{\mathsf{inr}^{g_{11}} E_1 \sqsubseteq \mathsf{inr}^{g_{21}} E_2}$$

$$\sqsubseteq_{E10}\ \frac{G_{11} \sqsubseteq G_{21} \qquad G_{12} \sqsubseteq G_{22} \qquad t_{11} \sqsubseteq t_{21} \qquad t_{12} \sqsubseteq t_{22} \qquad E_1 \sqsubseteq E_2}{\mathsf{case}\ E_1\ \mathsf{of}\ \{ x^{G_{11}} \Rightarrow t_{11} \}\ \{ y^{G_{12}} \Rightarrow t_{12} \} \sqsubseteq \mathsf{case}\ E_2\ \mathsf{of}\ \{ x^{G_{21}} \Rightarrow t_{21} \}\ \{ y^{G_{22}} \Rightarrow t_{22} \}}$$

$$\sqsubseteq_{E11}\ \frac{\varepsilon_1 \sqsubseteq \varepsilon_2 \qquad E_1 \sqsubseteq E_2 \qquad G_1 \sqsubseteq G_2}{\varepsilon_1 E_1 :: G_1 \sqsubseteq \varepsilon_2 E_2 :: G_2}$$

Figure B.1: Precision of evaluation contexts

And the result immediately holds.

Case (IGR-ASCR).

We know that $t_1^{g_1;\Xi_1} = \varepsilon_1(\varepsilon_1' u_1 :: g_1'; \Xi_1') :: g_1; \Xi_1$, then there exists $t_1^{g_2;\Xi_2} = \varepsilon_2(\varepsilon_2' u_2 :: g_2'; \Xi_2') :: g_2; \Xi_2$ such that:

$$\sqsubseteq_{::}\ \frac{\varepsilon_1 \sqsubseteq \varepsilon_2 \qquad \sqsubseteq_{::}\ \dfrac{\varepsilon_1' \sqsubseteq \varepsilon_2' \qquad u_1 \sqsubseteq u_2 \qquad g_1'; \Xi_1' \sqsubseteq g_2'; \Xi_2'}{\varepsilon_1' u_1 :: g_1'; \Xi_1' \sqsubseteq \varepsilon_2' u_2 :: g_2'; \Xi_2'} \qquad g_1; \Xi_1 \sqsubseteq g_2; \Xi_2}{\varepsilon_1(\varepsilon_1' u_1 :: g_1'; \Xi_1') :: g_1; \Xi_1 \sqsubseteq \varepsilon_2(\varepsilon_2' u_2 :: g_2'; \Xi_2') :: g_2; \Xi_2}$$

If $t_1^{g_1;\Xi_1} \longmapsto \varepsilon_1' \circ^{<:} \varepsilon_1 u_1 :: g_1; \Xi_1$, then:

1. By Proposition 63, $\varepsilon_2' \circ^{<:} \varepsilon_2$ is defined and $\varepsilon_1' \circ^{<:} \varepsilon_1 \sqsubseteq \varepsilon_2' \circ^{<:} \varepsilon_2$.

Finally,

$$\sqsubseteq_{::}\ \frac{\varepsilon_1' \circ^{<:} \varepsilon_1 \sqsubseteq \varepsilon_2' \circ^{<:} \varepsilon_2 \qquad u_1 \sqsubseteq u_2 \qquad g_1; \Xi_1 \sqsubseteq g_2; \Xi_2}{\varepsilon_1' \circ^{<:} \varepsilon_1 u_1 :: g_1; \Xi_1 \sqsubseteq \varepsilon_2' \circ^{<:} \varepsilon_2 u_2 :: g_2; \Xi_2}$$

And the result holds.

□

Proposition 86. *Suppose $t_1 = E_1[t_1']$. If $t_1 \sqsubseteq t_2$ then there exist E_2 and t_2' such that $t_2 = E_2[t_2']$ where $E_1 \sqsubseteq E_2$ and $t_1' \sqsubseteq t_2'$.*

Proof. By induction on $t_1 = E_1[t_1']$.

Case $E_1 = E + t_{12}$. By inspection on $(\mathrm{IG}\sqsubseteq_{+})$, $t_2 = t_{21} + t_{22}$, where $t_1' \sqsubseteq t_{21}$ and $t_{12} \sqsubseteq t_{22}$. Then, $E_2 = E + t_{22}$ and $t_2' = t_{21}$.

Case $E_1 = v_{11} + E$. By inspection on $(\mathrm{IG}\sqsubseteq_{+})$, $t_2 = v_{21} + t_{22}$, where $v_{11} \sqsubseteq v_{21}$ and $t_1' \sqsubseteq t_{22}$. Then, $E_2 = v_{11} + E$ and $t_2' = t_{22}$.

Case $E_1 = E \leq t_{12}$. By inspection on $(\mathrm{IG}\sqsubseteq_{\leq})$, $t_2 = t_{21} \leq t_{22}$, where $t_1' \sqsubseteq t_{21}$ and $t_{12} \sqsubseteq t_{22}$. Then, $E_2 = E \leq t_{22}$ and $t_2' = t_{21}$.

Case $E_1 = v_{11} \leq E$. By inspection on $(\mathrm{IG}\sqsubseteq_{\leq})$, $t_2 = v_{21} \leq t_{22}$, where $v_{11} \sqsubseteq v_{21}$ and $t_1' \sqsubseteq t_{22}$. Then, $E_2 = v_{11} \leq E$ and $t_2' = t_{22}$.

Case $E_1 = E\ t_{12}$ and $E_1 = v_{11}\ E$. Analogous.

Case $E_1 = \mathsf{inl}^{g_{12}} E$. By inspection on $(\mathrm{IG}\sqsubseteq_{\mathsf{inl}})$, $t_2 = \mathsf{inl}^{g_{22}} t_{21}$, where $g_{12} \sqsubseteq g_{22}$ and $t_1' \sqsubseteq t_{21}$. Then, $E_2 = \mathsf{inl}^{g_{22}} E$ and $t_2' = t_{21}$.

Case $E_1 = \mathsf{inr}^{g_{11}} E$. Analogous.

Case $E_1 = \mathsf{case}\ E\ \mathsf{of}\,\{\, x^{G_{11}} \Rightarrow t_{12} \,\}\ \{\, y^{G_{12}} \Rightarrow t_{13} \,\}$.

- By inspection on $(\mathrm{IG}\sqsubseteq_{\mathsf{case}})$, $t_2 = \mathsf{case}\ t_{21}\ \mathsf{of}\,\{\, x^{G_{21}} \Rightarrow t_{22} \,\}\ \{\, y^{G_{22}} \Rightarrow t_{23} \,\}$, where $t_1' \sqsubseteq t_{21}$, $G_{11} \sqsubseteq G_{21}$, $t_{12} \sqsubseteq t_{22}$, $G_{12} \sqsubseteq G_{22}$, $t_{13} \sqsubseteq t_{23}$.
- Then, $E_2 = \mathsf{case}\ E\ \mathsf{of}\,\{\, x^{G_{21}} \Rightarrow t_{22} \,\}\ \{\, y^{G_{22}} \Rightarrow t_{23} \,\}$ and $t_2' = t_{21}$.

Case $E_1 = \varepsilon_1 E :: G_1$. By inspection on $(\mathrm{IG}\sqsubseteq_{::})$, $t_2 = \varepsilon_2 t_{21} :: G_2$, where $\varepsilon_1 \sqsubseteq \varepsilon_2$, $t_1' \sqsubseteq t_{21}$ and $G_1 \sqsubseteq G_2$. Then, $E_2 = \varepsilon_2 E :: G_2$ and $t_2' = t_{21}$.

Finally, the result holds. $\square$

Proposition 31 (Dynamic gradual guarantee). *Suppose* $t_{11} \sqsubseteq t_{12}$ *and* $\gamma_1 \sqsubseteq \gamma_2$. *If* $t_{11} \xmapsto{\gamma_1} t_{21}$ *then* $t_{12} \xmapsto{\gamma_2} t_{22}$ *where* $t_{21} \sqsubseteq t_{22}$.

Proof. Let $t_{11} = t_1^{G_1}$, $t_{12} = t_1^{G_2}$, $t_{21} = t_2^{G_1}$ and $t_{22} = t_2^{G_2}$. By induction on the definition of $t_1^{G_1} \longmapsto t_2^{G_1}$.

Case $(\mathrm{IGR}{\longrightarrow})$. By Proposition 85

Case (IGRE). We know that $t_1^{G_1} = E_1[t_1'^{G_1'}]$, $t_2^{G_1} = E_1[t_2'^{G_1'}]$, $t_1'^{G_1'} \longmapsto t_2'^{G_1'}$, $E_1 : G_1' \to G_1$. Then,

1. By Proposition 86, there exist $E_2 : G_2' \to G_2$ and $t_1'^{G_2'}$ such that $t_1^{G_2} = E_2[t_1'^{G_2'}]$ where $t_1'^{G_1'} \sqsubseteq t_1'^{G_2'}$ and $E_1 \sqsubseteq E_2$.
2. By induction hypothesis, $t_1'^{G_2'} \longmapsto t_2'^{G_2'}$ where $t_2'^{G_1'} \sqsubseteq t_2'^{G_2'}$.
3. Let $t_2^{G_2} = E_2[t_2'^{G_2'}]$. By (IGRE), $t_1^{G_2} \longmapsto t_2^{G_2}$.
4. By $(\mathrm{IG}\sqsubseteq_{E10})$, $t_2^{G_1} \sqsubseteq t_2^{G_2}$.

And the result holds.

Case (IGRctx). We know that $t_1^{G_1} = \mathsf{ctx}(\gamma_1', t'^{G_1'}_1)$, $t_2^{G_1} = \mathsf{ctx}(\gamma_1', t'^{G_1'}_2)$, $t'^{G_1'}_1 \xmapsto{\gamma_1'} t'^{G_1'}_2$. Then,

1. By (IG$\sqsubseteq_{\mathsf{ctx}}$), $t_1^{G_2} = \mathsf{ctx}(\gamma_2', t'^{G_2'}_1)$ where $\gamma_1' \sqsubseteq \gamma_2'$ and $t'^{G_1'}_1 \sqsubseteq t'^{G_2'}_1$.
2. By induction hypothesis, $t'^{G_2'}_1 \xmapsto{\gamma_2'} t'^{G_2'}_2$ where $t'^{G_1'}_2 \sqsubseteq t'^{G_2'}_2$.
3. Let $t_2^{G_2} = \mathsf{ctx}(\gamma_2', t'^{G_2'}_2)$. By (IGRctx), $t_1^{G_2} \longmapsto t_2^{G_2}$.
4. By (IG$\sqsubseteq_{\mathsf{ctx}}$), $t_1^{G_2} \sqsubseteq t_2^{G_2}$.

And the result holds.

□

B.5. Soundness

Proof artifacts. Before proving metric preservation, we introduce several artifacts for easing the proofs:

- **Sensitivity substitutions**: We introduce a structure ψ, called sensitivity substitution, that maps variables to sensitivity environments. Its role is to close the sensitivities of an open term. For example, consider $t = x^{\mathbb{R};x} + 2y^{\mathbb{R};y} + z^{\mathbb{R};z}$ and $\psi = \{\, y \mapsto 2x, z \mapsto 5x \,\}$. Then, $\psi(t) = x^{\mathbb{R};x} + 2y^{\mathbb{R};2x} + z^{\mathbb{R};5x}$. Essentially, a sensitivity substitution states which of the free variables are directly and indirectly sensitive. Formally, the syntax and application (on terms, type-and-effects, type environments and evidences) of sensitivity substitutions is defined as following:

$$\psi ::= \varnothing \mid \psi[x \mapsto \Xi]$$

$$\begin{aligned} \varnothing(t) &= t \\ \psi[x \mapsto \Xi](t) &= \psi([\Xi/x]t) \end{aligned}$$

$$\begin{aligned} \varnothing(G) &= G \\ \psi[x \mapsto \Xi](G) &= \psi([\Xi/x]G) \end{aligned}$$

$$\begin{aligned} \psi(\Gamma) &= \Gamma \\ \psi(\Gamma, y : G) &= \psi(\Gamma), y : \psi(G) \end{aligned}$$

$$\psi(\langle\, G_1, G_2 \,\rangle) = \langle\, \psi(G_1), \psi(G_2) \,\rangle$$

- **Evidence projections**: We use the notation $\varepsilon.\Xi_l$ and $\varepsilon.\Xi_r$ for projecting the sensitivity effects from the left-hand side and right-hand side, respectively. Formally, they are defined as:

$$\langle g_1; \Xi_1, g_2; \Xi_2 \rangle.\Xi_l = \Xi_1$$
$$\langle g_1; \Xi_1, g_2; \Xi_2 \rangle.\Xi_r = \Xi_2$$

- *sdom*: We introduce the *sdom* function as a shortcut for *idom* and an additional substitution. Formally:

$$sdom(\Xi, \varepsilon) = [\Xi/x]idom(\varepsilon) \quad \text{where } \varepsilon = \langle (x : g'_{11}) \xrightarrow{\Xi'_{12}} g'_{12}; \Xi'_1, (x : g'_{21}) \xrightarrow{\Xi'_{22}} g'_{22}; \Xi'_2 \rangle$$

sdom naturally inherits the properties of sensitivity substitution and *idom*.

Lemma 87. *If* $\pi_1(\Delta \cdot \Xi_1) \leq \pi_1(\Delta \cdot \Xi_2)$*, and* $\pi_1(\Delta \cdot \Xi_3) \leq \pi_1(\Delta \cdot \Xi_4)$*, then* $\pi_1(\Delta \cdot (\Xi_1 \sqcup \Xi_3)) \leq \pi_1(\Delta \cdot (\Xi_2 \sqcup \Xi_4))$

Proof.

$$\begin{aligned}\pi_1(\Delta \cdot (\Xi_1 \sqcup \Xi_3)) &= \pi_1((\Delta \cdot \Xi_1) \sqcup (\Delta \cdot \Xi_3)) \\ &= \max(\pi_1(\Delta \cdot \Xi_1), \pi_1((\Delta \cdot \Xi_3))) && \text{By prop 19} \\ &\leq \max(\pi_1(\Delta \cdot \Xi_2), \pi_1((\Delta \cdot \Xi_4))) \\ &= \pi_1((\Delta \cdot \Xi_2) \sqcup (\Delta \cdot \Xi_4)) \\ &= \pi_1(\Delta \cdot (\Xi_2 \sqcup \Xi_4))\end{aligned}$$

□

Proposition 88. *If*

- $\varepsilon_i u_i :: g; \Xi \in \mathbb{T}[g; \Xi]$
- $\pi_1(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r)) \geq \infty$

then $(\varepsilon_1 u_1 :: g; \Xi, \varepsilon_1 u_2 :: g; \Xi) \in \mathcal{V}_\Delta[\![g; \Xi]\!]$.

Proof. For proving the goal we have to prove that,

1. $(\varepsilon_1 u_1 :: g; \Xi, \varepsilon_1 u_2 :: g; \Xi) \in \mathsf{Atom}[\![g; \Xi]\!]$: Trivial as by hypothesis both values typecheck.
2. $\neg(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r) \mathrel{\tilde{<}} |u_1 - u_2|)$:

$$\begin{aligned}&\neg(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r) \mathrel{\tilde{<}} |u_1 - u_2|) \\ &\iff |u_1 - u_2| \leq \pi_1(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r)) \\ &\iff |u_1 - u_2| \leq \infty \leq \pi_1(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r)) \\ &\iff \top\end{aligned}$$

Finally, $(\varepsilon_1 u_1 :: g; \Xi, \varepsilon_1 u_2 :: g; \Xi) \in \mathcal{V}_\Delta[\![g; \Xi]\!]$. □

Proposition 89. *Suppose $t \in \mathbb{T}[G]$, $t \vdash \Gamma$, $t' = \psi(t)$ and $\Gamma' = \psi(\Gamma)$. Then, $\forall \Delta, \gamma_1, \gamma_2$ such that $\Gamma' \vdash \Delta$ and $(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma']\!]$, it follows that $(t' \mid \gamma_1, t' \mid \gamma_2) \in \mathcal{T}_\Delta[\![\psi(g); \psi(\Xi)]\!]$.*

Proof. We proceed by induction on $t \in \mathbb{T}[g; \Xi]$

Case $t = \varepsilon n :: \mathbb{R}; \Xi \in \mathbb{T}[\mathbb{R}; \Xi]$, and $t' = \psi(\varepsilon) n :: \mathbb{R}; \psi(\Xi) \in \mathbb{T}[\mathbb{R}; \psi(\Xi)]$. Since t' is already a value, in order to prove the goal, we have to prove that $\underbrace{(t', t')} \in \mathcal{V}_\Delta[\![\mathbb{R}; \psi(\Xi)]\!]$. As $\psi(\varepsilon).\Xi_r \curlyvee \psi(\varepsilon).\Xi_r = \psi(\varepsilon).\Xi_r$, we have to prove that $\neg|n - n| > \Delta \cdot \psi(\varepsilon).\Xi_r$, which is true as $n - n = 0$ and $\Delta \cdot \psi(\varepsilon).\Xi_r \geq 0$, and the result holds.

Case $t = t_{01} + t_{02} \in \mathbb{T}[\mathbb{R}; \Xi_0]$. Then $t' = t_1 + t_2 \in \mathbb{T}[\mathbb{R}; \Xi]$, for $t_1 = \psi(t_1)$, $t_2 = \psi(t_2)$ and $\Xi = \psi(\Xi_0)$. We have to prove that

$$(t' \mid \gamma_1, t' \mid \gamma_2) \in \mathcal{T}_\Delta[\![\mathbb{R}; \Xi]\!]$$

i.e. if $t' \xmapsto{\gamma_i}{}^* v_i$, then

$$(v_1, v_2) \in \mathcal{V}_\Delta[\![\mathbb{R}; \Xi]\!]$$

where $v_i = \varepsilon_i r_i \mathbb{R}; \Xi$.

If $t' \xmapsto{\gamma_i}{}^*$ **error** the result holds immediately. Suppose the term reduces. By induction hypothesis on $t_{0j} \in \mathbb{T}[\mathbb{R}; \Xi_{0j}]$, then $t_j \xmapsto{\gamma_i}{}^* v_{ji}$ and $(v_{j1}, v_{j2}) \in \mathcal{V}_\Delta[\![\mathbb{R}; \Xi_j]\!]$, where $\Xi = \Xi_1 + \Xi_2$ and $v_i = v_{1i} + v_{2i}$. In particular,

- Let $v_{ji} = \varepsilon_{ji} r_{ji} :: \mathbb{R}; \Xi_j$.
- $\varepsilon_i = \varepsilon_{1i} +_\Xi \varepsilon_{2i}$.
- $r_i = r_{1i} + r_{2i}$.
- $|r_1 - r_2| = |r_{11} + r_{21} - r_{12} - r_{22}| \leq |r_{11} - r_{12}| + |r_{21} - r_{22}|$.
-

$$\begin{aligned}\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r) &= \Delta \cdot ((\varepsilon_{11} +_\Xi \varepsilon_{21}).\Xi_r \curlyvee (\varepsilon_{12} +_\Xi \varepsilon_{22}).\Xi_r) \\ &= \Delta \cdot ((\varepsilon_{11}.\Xi_r + \varepsilon_{21}.\Xi_r) \curlyvee (\varepsilon_{12}.\Xi_r + \varepsilon_{22}.\Xi_r))\end{aligned}$$

- $\neg(\Delta \cdot (\varepsilon_{j1}.\Xi_r \curlyvee \varepsilon_{j2}.\Xi_r) \mathrel{\widetilde{<}} |r_{j1} - r_{j2}|)$ is true. Furthermore,

$$\begin{aligned}&\neg(\Delta \cdot (\varepsilon_{j1}.\Xi_r \curlyvee \varepsilon_{j2}.\Xi_r) \mathrel{\widetilde{<}} |r_{j1} - r_{j2}|) \\ &\implies \neg(\Delta \cdot (\varepsilon_{11}.\Xi_r \curlyvee \varepsilon_{12}.\Xi_r) + \Delta \cdot (\varepsilon_{21}.\Xi_r \curlyvee \varepsilon_{22}.\Xi_r) \mathrel{\widetilde{<}} |r_{11} - r_{12}| + |r_{21} - r_{22}|) \\ &\implies \neg(\Delta \cdot ((\varepsilon_{11}.\Xi_r \curlyvee \varepsilon_{12}.\Xi_r) + (\varepsilon_{21}.\Xi_r \curlyvee \varepsilon_{22}.\Xi_r)) \mathrel{\widetilde{<}} |r_{11} - r_{12}| + |r_{21} - r_{22}|) \\ &\implies \neg(\Delta \cdot ((\varepsilon_{11}.\Xi_r + \varepsilon_{21}.\Xi_r) \curlyvee (\varepsilon_{12}.\Xi_r + \varepsilon_{22}.\Xi_r)) \mathrel{\widetilde{<}} |r_{11} - r_{12}| + |r_{21} - r_{22}|) \\ &\implies \neg(\Delta \cdot ((\varepsilon_{11}.\Xi_r + \varepsilon_{21}.\Xi_r) \curlyvee (\varepsilon_{12}.\Xi_r + \varepsilon_{22}.\Xi_r)) \mathrel{\widetilde{<}} |r_{11} + r_{21} - r_{12} - r_{22}|) \\ &\implies \neg(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r) \mathrel{\widetilde{<}} |r_1 - r_2|)\end{aligned}$$

Finally, the result holds.

Case $t = x^{g;\Xi} \in \mathbb{T}[g;\Xi]$, and $t' = x^{\psi(g);\psi(\Xi)}$. We know that $x^{\psi(g);\psi(\Xi)} \xmapsto{\gamma_i} \gamma_i(x)$, but by the definition of related environments, $(\gamma_1(x), \gamma_2(x)) \in \mathcal{V}_\Delta[\![\psi(g); \psi(\Xi)]\!]$ and the result holds.

Case $t = \varepsilon_0(\mathsf{inl}^{g'_{02}} t_0) :: g_{01} \ {}^{\Xi_{01}}\!\oplus^{\Xi_{02}} \ g_{02}; \Xi_0 \in \mathbb{T}[g_{01} \ {}^{\Xi_{01}}\!\oplus^{\Xi_{02}} \ g_{02}; \Xi_0]$.

Then $t' = \varepsilon(\mathsf{inl}^{g'_2} t^{g'_1;\Xi'_1}) :: g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2} g_2$, for $\varepsilon = \psi(\varepsilon_0)$, $g_i = \psi(g_{0i})$, $\Xi_i = \psi(\Xi_{oi})$, $g'_i = \psi(g'_{0i})$, and $t^{g'_1} = \psi(t_0)$.

We have to prove that

$$(\varepsilon(\mathsf{inl}^{g'_2} t^{g'_1}) :: g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2} \ g_2; \Xi \mid \gamma_1, \varepsilon(\mathsf{inl}^{g'_2} t^{g'_1}) :: g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2} \ g_2; \Xi \mid \gamma_2) \in \mathcal{T}_\Delta[\![g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2} \ g_2; \Xi]\!]$$

i.e. if $t' \xmapsto{\gamma_i}{}^* v_i$, which means, by (IGRE), that $t^{g'_1;\Xi'_1} \xmapsto{\gamma_i}{}^* v'_i$, then

$$(v_1, v_2) \in \mathcal{V}_\Delta[\![g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2} \ g_2; \Xi]\!]$$

where $v_i = \varepsilon(\mathsf{inl}^{g'_2} v'_i) :: g_1 \ {}^{\Xi_1}\!\oplus^{\Xi_2} \ g_2; \Xi$.

If $t^{g'_1;\Xi'_1} \xmapsto{\gamma_i}{}^* \textbf{error}$, the result holds immediately. Suppose the term reduces. Knowing that $\varepsilon.\Xi_r \curlyvee \varepsilon.\Xi_r = \varepsilon.\Xi_r$, we have to prove that if $\widetilde{\Delta \cdot \varepsilon.\Xi_r < \infty}$ then, for all $\Gamma'', \gamma''_1, \gamma''_2$, such that $(\gamma''_1, \gamma''_2) \in \mathcal{G}_\Delta[\![\Gamma'']\!]$, either $(useL(v_1) \mid \gamma''_1, useL(v_2) \mid \gamma''_2) \in \mathcal{T}_\Delta[\![g_1; \Xi_1]\!]$, or $(useR(v_1) \mid \gamma''_1, useR(v_2) \mid \gamma''_2) \in \mathcal{T}_\Delta[\![g_2; \Xi + \Xi_2]\!]$, which is equivalent to show that either $(useL(v_1), useL(v_2)) \in \mathcal{V}_\Delta[\![g_1; \Xi + \Xi_1]\!]$ or $(useR(v_1), useR(v_2)) \in \mathcal{V}_\Delta[\![g_2; \Xi + \Xi_2]\!]$ (because $useR(v_i)$ and $useL(v_i)$ are already values or undefined).

Note that $\varepsilon = \langle g_{11} \ {}^{\Xi_{11}}\!\oplus^{\varnothing} \ g_{12}; \varnothing, g_{21} \ {}^{\Xi_{21}}\!\oplus^{\Xi_{22}} \ g_{22}; \Xi' \rangle$, for some g_{ij}, Ξ_{ij}, Ξ'. But

$$useL(\varepsilon(\mathsf{inl}^{g'_2} v_i) :: g_1 \ {}^{\Xi_1}\!\oplus^{\varnothing} \ g_2) = ileft(\varepsilon) v_i :: g_1; \Xi + \Xi_1$$

for $ileft(\varepsilon) = \langle g_{11}; \Xi_{11}, g_{21}; \Xi' + \Xi_{21} \rangle$. By induction hypothesis on $t_0 \in \mathbb{T}[g'_{01}; \Xi'_{01}]$, we know that $(v'_1, v'_2) \in \mathcal{V}_\Delta[\![g'_1; \Xi'_1]\!]$. But $ileft(\varepsilon) \vdash g'_1; \Xi'_1 \ \widetilde{<:}\ g_1; \Xi + \Xi_1$, and the result holds by Proposition 32.

Case $t = \varepsilon_0(\lambda x^{g'_{01};x}.t_0^{g'_{02};\Xi''_0}) :: (x : g_{01}) \xrightarrow{\Xi'_0} g_{02}; \in \mathbb{T}[(x : g_{01}) \xrightarrow{\Xi'_0} g_{02}; \Xi_0]$.

Then $t' = \varepsilon(\lambda x^{g'_1;x}.t^{g'_2;\Xi''}) :: (x : g_1) \xrightarrow{\Xi'} g_2; \in \mathbb{T}[(x : g_1) \xrightarrow{\Xi'} g_2; \Xi]$ (after ψ substitution). We know that

$$\varepsilon(\lambda x^{g'_1;x}.t^{g'_2;\Xi''}) :: (x : g_1) \xrightarrow{\Xi'} g_2; \Xi \xmapsto{\gamma_i} v_i$$

For, $v_i = \varepsilon\langle \lambda x^{g'_1;x}.t^{g'_2;\Xi''}, \gamma_i \rangle :: (x : g_1) \xrightarrow{\Xi'} g_2; \Xi$.

We have to prove that,

$$\begin{aligned}(\varepsilon\langle \lambda x^{g'_1;x}.t^{g'_2;\Xi''}, \gamma_1 \rangle :: (x : g_1) \xrightarrow{\Xi'} g_2; \Xi,\\ \varepsilon\langle \lambda x^{g'_1;x}.t^{g'_2;\Xi''}, \gamma_2 \rangle :: (x : g_1) \xrightarrow{\Xi'} g_2; \Xi)\\ \in \mathcal{V}_\Delta[\![(x : g_1) \xrightarrow{\Xi'} g_2; \Xi]\!]\end{aligned}$$

i.e. $\forall v_1', v_2', \Xi_1, \Gamma'', \gamma_1', \gamma_2'$, such that $\forall(\gamma_1', \gamma_2') \in \mathcal{G}_\Delta[\![\Gamma'']\!]$ and $(v_1', v_2') \in \mathcal{V}_\Delta[\![g_1; \Xi_1]\!]$, it follows that $(v_1\ v_1' \mid \gamma_1', v_2\ v_2' \mid \gamma_2') \in \mathcal{T}_\Delta[\![[\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi']\!]$. Consider $v_i' = \varepsilon_{2i} u_{2i} :: g_1; \Xi_1$ Then

$$v_i\ v_i' \xmapsto{\gamma_i'} \varepsilon_{11}\mathsf{ctx}(\gamma_i[x \mapsto v_i''], [\Xi_1/x]t^{g_2';\Xi''}) :: [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi'$$

where $v_i'' = (\varepsilon_{2i} \circ^{<:} sdom(\Xi_1, \varepsilon))u_{2i} :: g_1; x$ (notice that if consistent transitivity does not hold the result holds immediately).

We know that $sdom(\Xi_1, \varepsilon) \ \triangleright\ g_1'; \Xi_1 \ \widetilde{<:}\ g_1; \Xi_1$. By Prop 32, we know that $(v_1'', v_2'') \in \mathcal{V}_\Delta[\![g_1; \Xi_1]\!]$.

Let $\gamma_i'' = \gamma_i'[x \mapsto v_i'']$, then by Definition of related substitutions $(\gamma_1'', \gamma_2'') \in \mathcal{G}_\Delta[\![(\Gamma'', x : g_1'; \Xi_1)]\!]$.

We know that $t^{g_2';\Xi''} \in \mathbb{T}[g_2'; \Xi'']$

By Prop 73, $[\Xi_1/x]t^{g_2';\Xi''} \in \mathbb{T}[[\Xi_1/x]g_2'; [\Xi_1/x]\Xi'']$, then we use induction hypothesis on $t_0^{g_{02}';\Xi_0''}$, using $(\gamma_1'', \gamma_2'') \in \mathcal{G}_\Delta[\![(\Gamma, x : g_1'; \Xi_1)]\!]$, and $\psi' = \psi[x \mapsto \Xi_1]$. Let us call $t'' = [\Xi_1/x]t^{g_2';\Xi''} = \psi'(t_0^{g_{02}';\Xi_0''})$. Note that $\psi'(g_{02}') = [\Xi_1/x]g_2'$, and $\psi'(\Xi_0'') = [\Xi_1/x]\Xi''$. We know that if $t'' \xmapsto{\gamma_i''} v_{ci}$ then $(v_{c1}, v_{c2}) \in \mathcal{V}_\Delta[\![[\Xi_1/x]g_2'; [\Xi_1/x]\Xi'']\!]$.

Therefore

$$\frac{t'' \xmapsto{\gamma_i''}{}^{*}\ v_{ci}}{\varepsilon_{11}\mathsf{ctx}(\gamma_i'', t'') :: [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi' \xmapsto{\gamma_i'}{}^{*}\ \varepsilon_{11}\mathsf{ctx}(\gamma_i'', v_{ci}) :: [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi'}$$

and

$$\varepsilon_{11}\mathsf{ctx}(\gamma_i'', v_{ci}) :: [\Xi_1/x]g_2; [\Xi_1/x]\Xi' \xmapsto{\gamma_i'} \varepsilon_{11}(v_{ci}) :: [\Xi_1/x]g_2; \Xi + [\Xi_1/x]\Xi'$$

And the result follows by Proposition 32.

Case $t = t_{01}\ t_{02} \in \mathbb{T}[g; \Xi]$.

Then $t' = t_1\ t_2 \in \mathbb{T}[g'; \Xi']$, where $t_i = \psi(t_{0i}), g' = \psi(g)$, and $\Xi' = \psi(\Xi)$.

Suppose $t_{01} \in \mathbb{T}[(x : g_{01}) \xrightarrow{\Xi_{02}} g_{02}; \Xi_{01}]$, and thus $t_{02} \in \mathbb{T}[g_{01}; \Xi_{03}]$. Also $\psi(g_{0i}) = g_i$, and $\psi(\Xi_{0i}) = \Xi_i$. By induction hypotheses, we know that $t_1 \xmapsto{\gamma_i'}{}^{*}\ v_{1i}$, and $t_2 \xmapsto{\gamma_i'}{}^{*}\ v_{2i}$ (otherwise the result holds immediately), and that $(v_{11}, v_{12}) \in \mathcal{V}_\Delta[\![(x : g_1) \xrightarrow{\Xi_2} g_2; \Xi_1]\!]$, and $(v_{21}, v_{22}) \in \mathcal{V}_\Delta[\![g_1; \Xi_3]\!]$. Following the definition of related functions instantiated with $(v_{21}, v_{22}) \in \mathcal{V}_\Delta[\![g_1; \Xi_3]\!]$ we know that

$$(v_{11}\ v_{21}, v_{12}\ v_{22}) \in \mathcal{T}_\Delta[\![[\Xi_3/x]g_2; \Xi_2 + [\Xi_3/x]\Xi_1]\!]$$

but $g' = [\Xi_3/x]g_2$ and $\Xi' = \Xi_2 + [\Xi_3/x]\Xi_1$ and the result holds.

Case $t = \mathsf{case}\ t_1^{g_{01};\Xi_{01}}\ \mathsf{of}\,\{\, x \Rightarrow t_2^{g_{02};\Xi_{02}} \,\}\ \{\, y \Rightarrow t_3^{g_{03};\Xi_{03}} \,\} \in \mathbb{T}[g_0; \Xi_0]$.

Then $t' = \mathsf{case}\ t^{g_1;\Xi_1}\ \mathsf{of}\,\{\, x \Rightarrow t_2^{g_2;\Xi_2} \,\}\ \{\, y \Rightarrow t_3^{g_3;\Xi_3} \,\}$, for $g_i = \psi(g_{0i})$, $\Xi_i = \psi(\Xi_{0i})$.

Let us suppose consistent transitivity hold, otherwise the result holds immediately. By induction hypothesis on $t^{g_{01};\Xi_{01}} \in \mathbb{T}[g_{01};\Xi_{01}]$, if $t^{g_1;\Xi_1} \xrightarrow{\gamma_i}{}^{*} v_{i1}$, then $(v_{11}, v_{21}) \in \mathcal{V}_\Delta[\![g_1;\Xi_1]\!]$, where $g_1 = g_{11}\ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\ g_{12}$, for some g_{1i}, Ξ_{1i}.

Suppose $\Xi_m = \varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r$, and $\neg(\Delta \cdot \Xi_m \ \widetilde{<:}\ \infty)$, and without loosing generality $v_{11} = (\varepsilon_1 \mathsf{inl}^{g_{lr}}(\varepsilon_l u_l :: g_{ll};\Xi_l) :: g_{11}\ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\ g_{12};\Xi_1)$, and $v_{21} = (\varepsilon_2 \mathsf{inr}^{g_{rl}}(\varepsilon_r u_r :: g_{rr};\Xi_r) :: g_{11}\ {}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\ g_{12};\Xi_1)$.

Then

$$\begin{array}{l}\mathsf{case}\ v_{11}\ \mathsf{of}\ \{\, x \Rightarrow t_2^{g_2;\Xi_2}\,\}\ \{\, y \Rightarrow t_3^{g_3;\Xi_3}\,\} \xrightarrow{\gamma_1} \\ \quad \varepsilon'_{12}\mathsf{ctx}(\gamma_1[x \mapsto \varepsilon'_{11} u_l :: g_{11};\Xi_x], [\Xi_x/x]t_2) :: [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3;\Xi'\end{array}$$

$$\begin{array}{l}\mathsf{case}\ v_{12}\ \mathsf{of}\ \{\, x \Rightarrow t_2^{g_2;\Xi_2}\,\}\ \{\, y \Rightarrow t_3^{g_3;\Xi_3}\,\} \xrightarrow{\gamma_2} \\ \quad \varepsilon'_{22}\mathsf{ctx}(\gamma_2[y \mapsto \varepsilon'_{21} u_r :: g_{12};\Xi_y], [\Xi_y/y]t_3) :: [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3;\Xi'\end{array}$$

for $\Xi_x = \Xi_1 + \Xi_{11}$, $\Xi_y = \Xi_1 + \Xi_{12}$, $\Xi' = \Xi_1 \curlyvee [\Xi_x/x]\Xi_2 \curlyvee [\Xi_y/y]\Xi_3$, and $\varepsilon'_{11} = \varepsilon_l \circ^{<:} \mathit{ileft}(\varepsilon_1)$, $\varepsilon'_{21} = \varepsilon_r \circ^{<:} \mathit{ileft}(\varepsilon_2)$, $\varepsilon'_{12} = \mathcal{I}_{<:}([\Xi_x/x]g_2;[\Xi_x/x]\Xi_2, \Xi' \curlyvee \varepsilon_1.\Xi_r)$, $\varepsilon'_{22} = \mathcal{I}_{<:}([\Xi_y/y]g_3;[\Xi_y/y]\Xi_3, \Xi' \curlyvee \varepsilon_2.\Xi_r)$

As $\neg(\Delta \cdot \Xi_n \lesssim \infty)$, it means that $\pi_1(\Delta \cdot \Xi_m) \geq \infty$.

Suppose

$$\begin{array}{l}\quad \varepsilon'_{12}\mathsf{ctx}(\gamma_1[x \mapsto \varepsilon'_{11} u_l :: g_{11};\Xi_x], [\Xi_x/x]t_2) :: [\Xi_x/x]g_2 \sqcup [\Xi_x/x]g_2;\Xi' \\ \xrightarrow{\gamma_1}{}^{*} \varepsilon'_{12} v_2 :: [\Xi_x/x]g_2 \sqcup [\Xi_x/x]g_2;\Xi' \\ \xrightarrow{\gamma_1} v'_2\end{array}$$

and

$$\begin{array}{l}\quad \varepsilon'_{22}\mathsf{ctx}(\gamma_2[y \mapsto \varepsilon'_{21} u_r :: g_{12};\Xi_y], [\Xi_y/y]t_3) :: [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3;\Xi' \\ \xrightarrow{\gamma_1}{}^{*} \varepsilon'_{22} v_3 :: [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3;\Xi' \\ \xrightarrow{\gamma_1} v'_3\end{array}$$

where $v'_i = (\varepsilon_{ui} \circ^{<:} \varepsilon'_{i2})u'_i :: [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3;\Xi'$, for some v'_i and u'_i.

Notice that

1. By definition of $\mathcal{I}_{<:}$, $\varepsilon'_{i2}.\Xi_r \sqsubseteq \varepsilon_i.\Xi_r$, so $\pi_1(\Delta \cdot \varepsilon_i.\Xi_r) \leq \pi_1(\Delta \cdot \varepsilon'_{i2}.\Xi_r)$.
2. Then by Lemma 87, $\pi_1(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r) \leq \pi_1(\Delta \cdot (\varepsilon'_{i2}.\Xi_r \curlyvee \varepsilon'_{i2}.\Xi_r))$
3. As $\pi_1(\Delta \cdot \Xi_m) = \pi_1(\Delta \cdot (\varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r)) \geq \infty$, then $\pi_1(\Delta \cdot (\varepsilon'_{i2}.\Xi_r \curlyvee \varepsilon'_{i2}.\Xi_r)) \geq \infty$.
4. By monotonicity of consistent transitivity (Lemma 63), $(\varepsilon_{ui} \circ^{<:} \varepsilon'_{i2}).\Xi_r \sqsubseteq \varepsilon'_{i2}.\Xi_r$, and thus $\forall x, \pi_1(\varepsilon'_{i2}.\Xi_r(x)) \leq \pi_1((\varepsilon_{ui} \circ^{<:} \varepsilon'_{i2}).\Xi_r(x))$.
5. By proposition 39, $(\varepsilon_{u1} \circ^{<:} \varepsilon'_{12}).\Xi_r \curlyvee (\varepsilon_{u2} \circ^{<:} \varepsilon'_{22}).\Xi_r \sqsubseteq \varepsilon'_{12}.\Xi_r \curlyvee \varepsilon'_{22}.\Xi_r$
6. By proposition 69, $\Delta \cdot (\varepsilon_{u1} \circ^{<:} \varepsilon'_{12}).\Xi_r \curlyvee (\varepsilon_{u2} \circ^{<:} \varepsilon'_{22}).\Xi_r \sqsubseteq \Delta \cdot \varepsilon'_{12}.\Xi_r \curlyvee \varepsilon'_{22}.\Xi_r$
7. Which means that $\pi_1(\Delta \cdot (\varepsilon_{u1} \circ^{<:} \varepsilon'_{12}).\Xi_r \curlyvee (\varepsilon_{u2} \circ^{<:} \varepsilon'_{22}).\Xi_r) \geq \pi_1(\Delta \cdot \varepsilon'_{12}.\Xi_r \curlyvee \varepsilon'_{22}.\Xi_r)$

8. But by (3) then $\pi_1(\Delta \cdot (\varepsilon_{u1} \circ^{<:} \varepsilon'_{12}).\Xi_r \curlyvee (\varepsilon_{u2} \circ^{<:} \varepsilon'_{22}).\Xi_r) \geq \infty$, and the result holds by Proposition 88.

Now suppose $\Xi_m = \varepsilon_1.\Xi_r \curlyvee \varepsilon_2.\Xi_r$, and $(\Delta \cdot \Xi_m \,\widetilde{<:}\, \infty)$, and $v_{11} = (\varepsilon_1 \mathsf{inl}^{g'_{12}}(\varepsilon'_1 u_1 :: g'_{11}; \Xi'_{11}) :: g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\, g_{12}; \Xi_1)$, and $v_{21} = (\varepsilon_2 \mathsf{inl}^{g'_{22}}(\varepsilon'_2 u_2 :: g'_{21}; \Xi'_{21}) :: g_{11} \,{}^{\Xi_{11}}\!\oplus^{\Xi_{12}}\, g_{12}; \Xi_1)$ (the case where both terms are inr is analogous).

Then we know that

$$(\mathit{ileft}(\varepsilon_1)(\varepsilon'_1 u_1 :: g'_{11}; \Xi'_{11}) :: g_{11}; \Xi_x \mid \gamma_1, \mathit{ileft}(\varepsilon_2)(\varepsilon'_2 u_2 :: g'_{21}; \Xi'_{21}) :: g_{11}; \Xi_x \mid \gamma_2) \in \mathcal{T}_\Delta[\![g_{11}; \Xi_x]\!]$$

for $\Xi_x = \Xi_x$, i.e. if consistent transitivity holds, then

$$(\varepsilon'_{11} u_1 :: g_{11}; \Xi_x, \varepsilon'_{12} u_2 :: g_{11}; \Xi_x) \in \mathcal{V}_\Delta[\![g_{11}; \Xi_x]\!]$$

for $\varepsilon'_{1i} = \varepsilon'_i \circ^{<:} \mathit{ileft}(\varepsilon_i)$.

Then

$$\mathsf{case}\ v_{i1}\ \mathsf{of}\,\{\, x \Rightarrow t_2^{g_2;\Xi_2} \,\}\ \{\, y \Rightarrow t_3^{g_3;\Xi_3} \,\} \xmapsto{\gamma_i} \varepsilon'_{i2}\mathsf{ctx}(\gamma'_i, [\Xi_x/x]t_2) :: [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3; \Xi'$$

for $\gamma'_i = \gamma_i[x \mapsto \varepsilon'_{21} u_2 :: g_{11}; \Xi_x]$, $\Xi_y = \Xi_1 + \Xi_{12}$, $\Xi' = \Xi_1 \curlyvee [\Xi_x/x]\Xi_2 \curlyvee [\Xi_y/y]\Xi_3$, and $\varepsilon'_{i2} = \mathcal{I}_{<:}([\Xi_x/x]g_2; [\Xi_x/x]\Xi_2, \Xi' \curlyvee \varepsilon_i.\Xi_r)$. Notice that by Definition of related environments

$$(\gamma'_1, \gamma'_2) \in \mathcal{G}_\Delta[\![\Gamma, x : g_{11}; \Xi_x]\!]$$

By induction hypothesis on $t_{02} \in \mathbb{T}[g_{02}; \Xi_{02}]$, using $\psi' = \psi[x \mapsto \Xi_x]$ (and $\psi'(t_{02}) = [\Xi_x/x]\psi(t_{02}) = [\Xi_x/x]t_2$), if $[\Xi_x/x]t_2 \xmapsto{\gamma'_i} v_i$, then $(v_1, v_2) \in \mathcal{V}_\Delta[\![g_2; \Xi_2]\!]$.

Then if

$$\begin{aligned} &\varepsilon'_{i2}\mathsf{ctx}(\gamma'_i, [\Xi_x/x]t_2) :: [\Xi_x/x]g_2 \sqcup [\Xi_x/x]g_2; \Xi' \\ \xmapsto{\gamma_1}{}^{*}\ &\varepsilon'_{12} v_2 :: [\Xi_x/x]g_2 \sqcup [\Xi_x/x]g_2; \Xi' \\ \xmapsto{\gamma_1}\ &v'_2 \end{aligned}$$

where $v'_i = (\varepsilon_{ui} \circ^{<:} \varepsilon'_{i2})u'_i :: [\Xi_x/x]g_2 \sqcup [\Xi_y/y]g_3; \Xi'$, for some v'_i and u'_i. The result holds by Proposition 32.

Case $t = \varepsilon t_0 :: g; \Xi \in \mathbb{T}[g; \Xi]$. Suppose $t_0 \in \mathbb{T}[g_0; \Xi_0]$, then $\varepsilon \vdash g_0; \Xi_0 \,\widetilde{<:}\, g; \Xi$.

By induction hypothesis on $t_0 \in \mathbb{T}[g_0; \Xi_0]$, we know that $\psi(t_0) \xmapsto{\gamma'_i}{}^{*} v_i$ and $(v_1, v_2) \in \mathcal{V}_\Delta[\![\psi(g_0); \psi(\Xi_0)]\!]$.

By lemma 73, $\psi(\varepsilon) \vdash \psi(g_0); \psi(\Xi_0) \,\widetilde{<:}\, \psi(g); \psi(\Xi)$, and the result follows by Proposition 32.

□

Lemma 32. *If $(v_1, v_2) \in \mathcal{V}_\Delta[\![G]\!]$ and $\varepsilon_i \triangleright G \,\widetilde{<:}\, G'$, then $\forall \Gamma, \gamma_1, \gamma_2 : (\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$ it follows that $(\varepsilon_1 v_1 :: G' \mid \gamma_1, \varepsilon_2 v_2 :: G' \mid \gamma_2) \in \mathcal{T}_\Delta[\![G']\!]$.*

Proof. Let us suppose both combination of evidence succeed (otherwise the result holds immediately). Let us assume that $v_i = \varepsilon_{ui} u :: g; \Xi$, and that $\varepsilon_{ui} \circ \varepsilon_i = \varepsilon'_i$. Therefore we have to prove that $(\varepsilon'_1 u :: g'; \Xi', \varepsilon'_2 u :: g'; \Xi') \in \mathcal{V}_{\Delta'}[\![g'; \Xi']\!]$.

Case $g = \mathbb{R}$. Then we know that $(\varepsilon_{u1} n_1 :: \mathbb{R}; \Xi, \varepsilon_{u2} n_2 :: \mathbb{R}; \Xi) \in \mathcal{V}_\Delta[\![\mathbb{R}; \Xi]\!]$, for some n_i.

1. Suppose $\Xi_m = \varepsilon_{u1}.\Xi_r \curlyvee \varepsilon_{u2}.\Xi_r$ and $\Xi'_m = \varepsilon'_1.\Xi_r \curlyvee \varepsilon'_2.\Xi_r$.
2. By proposition 63, $\varepsilon'_i.\Xi_r \sqsubseteq \varepsilon_{ui}.\Xi_r$, and by Proposition 39, $\varepsilon'_1.\Xi_r \curlyvee \varepsilon'_2.\Xi_r \sqsubseteq \varepsilon_{u1}.\Xi_r \curlyvee \varepsilon_{u2}.\Xi_r$, i.e. $\forall x, \pi_1(\Xi_m(x)) = \pi_1(\varepsilon_{u1}.\Xi_r \curlyvee \varepsilon_{u2}.\Xi_r)(x) \leq \pi_1((\varepsilon'_1.\Xi_r \curlyvee \varepsilon'_2.\Xi_r)(x)) = \pi_1(\Xi'_m(x))$
3. We know $\neg(\Delta \cdot \Xi_m \mathrel{\widetilde{<:}} |n_1 - n_2|)$, which is equivalent to $|n_1 - n_2| \leq \pi_1(\Delta \cdot \Xi_m)$
4. We have to prove that $\neg(\Delta \cdot \Xi'_m \mathrel{\widetilde{<:}} |n_1 - n_2|)$, which is equivalent to $|n_1 - n_2| \leq \pi_1(\Delta \cdot \Xi'_m)$
5. But we know that $\forall x \in dom(\Xi'_m), \pi_1(\Xi_m(x)) \leq \pi_1(\Xi'_m(x))$, therefore $\pi_1(\Delta \cdot \Xi_m) \leq \pi_1(\Delta \cdot \Xi'_m)$, and the result holds.

Case $g = (y : g_1) \xrightarrow{\Xi_l} g_2$. Therefore $g' = (y : g'_1) \xrightarrow{\Xi'_l} g'_2$

Then we know that $\forall v_{a1}, v_{a2}, \Xi_a, \gamma_{a1}, \gamma_{a2}, \forall(\gamma_{a1}, \gamma_{a2}) \in \mathcal{G}_\Delta[\![\Gamma; \psi]\!]$, and $(v_{a1}, v_{a2}) \in \mathcal{V}_\Delta[\![g_1; \Xi_a]\!]$, then $(v_1\ v_{a1} \mid \gamma_{a1}, v_2\ v_{a2} \mid \gamma_{a2}) \in \mathcal{T}_\Delta[\![[\Xi_a/y]g_2; \Xi + [\Xi_a/y]\Xi_l]\!]$.

Suppose that $v_1 = \varepsilon_{u1}\langle \lambda y^{g_{11};y}.t_1, \gamma'_1\rangle :: g; \Xi$, $v_2 = \varepsilon_{u2}\langle \lambda y^{g_{21};y}.t_2, \gamma'_2\rangle :: g; \Xi$, and that $\varepsilon_{u1} \circ \varepsilon_1 = \varepsilon'_1$ and $\varepsilon_{u2} \circ \varepsilon_2 = \varepsilon'_2$.

Then $v'_1 = \varepsilon'_1\langle \lambda y^{g_{11};y}.t_1, \gamma'_1\rangle :: (y : g'_1) \xrightarrow{\Xi'_l} g'_2; \Xi'$, and $v'_2 = \varepsilon'_2\langle \lambda y^{g_{21};y}.t_2, \gamma'_2\rangle :: (y : g'_1) \xrightarrow{\Xi'_l} g'_2; \Xi'$.

We have to prove that $\forall v_{b1}, v_{b2}, \Xi_b, \gamma_{b1}, \gamma_{b2}, \forall(\gamma_{b1}, \gamma_{b2}) \in \mathcal{G}_\Delta[\![\Gamma']\!]$, and $(v_{b1}, v_{b2}) \in \mathcal{V}_\Delta[\![g'_1; \Xi_b]\!]$, then $(v'_1\ v_{b1} \mid \gamma_{b1}, v'_2\ v_{b2} \mid \gamma_{b2}) \in \mathcal{T}_\Delta[\![[\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l]\!]$. Note that $sdom(\Xi_b, \varepsilon'_i) = sdom(\Xi_b, \varepsilon_{ui} \circ \varepsilon_i) = sdom(\Xi_b, \varepsilon_i) \circ sdom(\Xi_b, \varepsilon_{ui})$ (Proposition 77).

Suppose $v_{bi} = \varepsilon_{bi} u_{bi} :: g'_1; \Xi_b$. Then $\varepsilon'_{bi} = \varepsilon_{bi} \circ sdom(\Xi_b, \varepsilon'_i) = \varepsilon_{bi} \circ (sdom(\Xi_b, \varepsilon) \circ sdom(\Xi_b, \varepsilon_i))$, and by associativity (Proposition 77), $\varepsilon'_{bi} = (\varepsilon_{bi} \circ sdom(\Xi_b, \varepsilon_i)) \circ sdom(\Xi_b, \varepsilon_{ui})$. Also $\varepsilon'_{i1} = [\Xi_b/y]icod(\varepsilon'_i) = [\Xi_b/y]icod(\varepsilon_{ui} \circ \varepsilon_i) = [\Xi_b/y]icod(\varepsilon_{ui}) \circ [\Xi_b/y]icod(\varepsilon_i)$, $\gamma''_i = \gamma'_i[y \mapsto \varepsilon'_{bi} u_{bi} :: g_{i1}; \Xi_b]$ we have to prove that:

$$\begin{aligned}&(\varepsilon'_{11}\mathsf{ctx}(\gamma''_1, [\Xi_b/y]t_1) :: [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l,\\ &\ \varepsilon'_{21}\mathsf{ctx}(\gamma''_2, [\Xi_b/y]t_2) :: [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l) \in \mathcal{T}_\Delta[\![[\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l]\!]\end{aligned}$$

Then by induction hypothesis on $(v_{b1}, v_{b2}) \in \mathcal{V}_\Delta[\![g'_1; \Xi_b]\!]$, using $\varepsilon''_i = sdom(\Xi_b, \varepsilon_i) \vdash g'_1; \Xi_b \mathrel{\widetilde{<:}} g_1; \Xi_b$ we know that

$$((\varepsilon_{b1} \circ \varepsilon''_i) u_{b1} :: g_1; \Xi_b, (\varepsilon_{b2} \circ \varepsilon''_i) u_{b2} :: g_2; \Xi_b) \in \mathcal{V}_\Delta[\![g_1; \Xi_b]\!]$$

Then by choosing $\Xi_a = \Xi_b$, $v_{ai} = (\varepsilon_{bi} \circ \varepsilon''_i) u_{bi} :: g_i; \Xi_b$, we know that $(v_1\ v_{a1} \mid \gamma_{a1}, v_2\ v_{a2} \mid \gamma_{a2}) \in \mathcal{T}_\Delta[\![[\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l]\!]$, i.e.

$$\begin{aligned}&(\varepsilon''_{11}\mathsf{ctx}(\gamma''_1, t_1) :: [\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l,\\ &\ \varepsilon''_{21}\mathsf{ctx}(\gamma''_2, t_2) :: [\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l) \in \mathcal{T}_{\Delta'}[\![[\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l]\!]\end{aligned}$$

for $\varepsilon''_{i1} = [\Xi_b/y]icod(\varepsilon_{ui})$. Then we know that

$$\begin{aligned}&\varepsilon''_{i1}\mathsf{ctx}(\gamma''_i, t_i) :: [\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l\\ \xrightarrow{\gamma_1}{}^{*}\ &\varepsilon''_{i1}v_{fi} :: [\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l\\ \xrightarrow{\gamma_1}\ &v'_{fi}\end{aligned}$$

where $(v'_{f1}, v'_{f2}) \in \mathcal{V}_\Delta[\![[\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l]\!]$.

Note that $y \notin FV(\Xi)$, then $[\Xi_b/y]\Xi = \Xi$. By induction hypothesis, using $\varepsilon'''_i = [\Xi_b/y]icod(\varepsilon_i) \vdash [\Xi_b/y]g_2; \Xi + [\Xi_b/y]\Xi_l \mathrel{\widetilde{<:}} [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l$. Then

$$(\varepsilon'''_i v'_{f1} :: [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l, \varepsilon'''_i v'_{f2} :: [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l) \in \mathcal{T}_\Delta[\![[\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l]\!]$$

i.e. if $\varepsilon'''_i v'_{fi} :: [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l \xrightarrow{\gamma_i}{}^{*} v''_{fi}$, then $(v''_{f1}, v''_{f2}) \in \mathcal{V}_\Delta[\![[\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l]\!]$.

But $\varepsilon'_{i1} = [\Xi_b/y]icod(\varepsilon_i) \circ [\Xi_b/y]icod(\varepsilon) = \varepsilon''_{i1} \circ \varepsilon'''_i$

Therefore by associativity of consistent transitivity, we know that

$$\begin{aligned}&\varepsilon'_{i1}\mathsf{ctx}(\gamma''_i, t_i) :: [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l\\ =&(\varepsilon''_{i1} \circ \varepsilon'''_i)\mathsf{ctx}(\gamma''_i, t_i) :: [\Xi_b/y]g'_2; \Xi' + [\Xi_b/y]\Xi'_l\\ \xrightarrow{\gamma_1}{}^{*}\ &v''_{fi}\end{aligned}$$

and the result holds.

□

Theorem 33 ((Gradual) metric preservation) If $t \in \mathbb{T}[G]$ and $t \vdash \Gamma$, then $\forall \Delta, \gamma_1, \gamma_2$ such that $\Gamma \vdash \Delta$ and $(\gamma_1, \gamma_2) \in \mathcal{G}_\Delta[\![\Gamma]\!]$, it follows that $(t \mid \gamma_1, t \mid \gamma_2) \in \mathcal{T}_\Delta[\![G]\!]$.

Proof. Direct as a corollary of Proposition 89. □