Bounded Quantification for Gradual Typing

Harley Eades III and Michael Townsend

Computer and Information Sciences
Augusta University
Augusta, USA
heades@augusta.edu

Abstract. In an earlier paper we introduce a new categorical model based on retracts that combines static and dynamic typing. We then showed that our model gave rise to a new and simple type system which combines static and dynamic typing. In this paper, we extend this type system with bounded quantification and lists, and then develop a gradually typed surface language that uses our new type system as a core casting calculus. Finally, we prove the gradual guarantee as put forth by Siek et al.

1 Introduction

In a previous paper the authors [**?**] have shown that static and dynamic typing can be combined in a very simple and intuitive way by combining the work of Scott [13] and Lambek [9] on categorical models of the untyped and typed λ-calculus, respectively. First, add a new type ? read "the type of untyped programs" – also sometimes called the unknown type – and then add four new programs $\mathsf{split} : ? \to (? \to ?)$, $\mathsf{squash} : (? \to ?) \to ?$, $\mathsf{box}_C : C \to ?$, and $\mathsf{unbox}_C : ? \to C$, such that, $\mathsf{squash};\mathsf{split} = \mathsf{id}_{? \to ?}$ and $\mathsf{box}_C;\mathsf{unbox}_C = \mathsf{id}_C$. Categorically, split and squash, and box and unbox form two retracts. Then extending the simply typed λ-calculus with these two retracts results in a new core casting calculus, called Simply Typed Grady, for Siek and Taha's gradual functional type system [16]. Furthermore, the authors show that Siek and Taha's system can be given a categorical model in cartesian closed categories with the two retracts.

In this paper we extend Grady with bounded quantification and lists. We chose bounded quantification so that the bounds can be used to control which types are castable and which should not be. Currently, we will not allow polymorphic types to be cast to the unknown type, because we do not have a good model nor are we sure how this would affect gradual typing. We do this by adding a new bounds, $\mathbb{S}$, whose subtypes are all non-polymorphic types – referred to hence forth as simple types. Then we give box and unbox the following types:

$$\frac{\Gamma\ \mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{box} : \forall(X <: \mathbb{S}).(X \to ?)}\ \mathsf{box} \quad \text{and} \quad \frac{\Gamma\ \mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{unbox} : \forall(X <: \mathbb{S}).(? \to X)}\ \mathsf{unbox}$$

This differs from our previous work where we limited box and unbox to only atomic types, but then we showed that they could be extended to any type by combining box and unbox with split and squash. In this paper we take these extended versions as primitive.

Grady now consists of two languages: a surface language – called Surface Grady – and a core language – called Core Grady. The difference between the surface and the core is that the former is gradually typed while the latter is statically typed. Gradual typing is the combination of static and dynamic typing in such a way that one can program in dynamic style. That is, the programmer should not have to introduce explicit casts. The first functional gradually typed language is due to Siek and Taha [15] where they extend the typed λ-calculus with the unknown type ? and a new relation on types, called the type consistency relation, that indicated when types should be considered as being castable or not. For example, consider the function application rule of Surface Grady:

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathsf{SG}} t_1 : C \\ \Gamma \vdash_{\mathsf{SG}} t_2 : A_2 \quad \Gamma \vdash A_2 \sim A_1 \quad \mathsf{fun}(C) = A_1 \to B_1\end{array}}{\Gamma \vdash_{\mathsf{SG}} t_1\, t_2 : B_1} \to_e$$

This rule depends on the relation $\Gamma \vdash A_2 \sim A_1$ which is the type consistency relation. It is reflexive and symmetric, but not transitive, or one could prove that any type is consistent with any other type. Thus, type consistency indicates exactly where explicit casts need to be inserted. This rule also depends on the partial function:

$$\begin{array}{l}\mathsf{fun}(?) = ? \to ? \\ \mathsf{fun}(A_1 \to B_1) = A_1 \to B_1\end{array}$$

As we will see below $\Gamma \vdash ? \sim A$ for any type A. As an example, suppose $\Gamma \vdash_{\mathsf{SG}} t_1 : ?$ and $\Gamma \vdash_{\mathsf{SG}} t_2 : \mathsf{Nat}$. In addition, we know $\Gamma \vdash ? \sim \mathsf{Nat}$, and $\mathsf{fun}(?) = ? \to ?$. Then based on the rule above $\Gamma \vdash_{\mathsf{SG}} t_1\, t_2 : ?$ is typable. Notice there are no explicit casts. Using split and $\mathsf{box}_{\mathsf{Nat}}$ we can translate this application into Core Grady by adding the casts: $\Gamma \vdash_{\mathsf{CG}} (\mathsf{split}_{(? \to ?)}\, t_1)\, (\mathsf{box}_{\mathsf{Nat}}\, t_2) : ?$.

Subtyping in Core Grady is standard subtyping for bounded system F extended with the new bounds for simple types. One important point is that in Core Grady the unknown type is not a top type, and in fact, is only related to itself and $\mathbb{S}$. However, subtyping in the surface language is substantially different.

In Surface Grady subtyping is the combination of subtyping and type consistency called consistent subtyping due to Siek and Taha [14]. We denote consistent subtyping by $\Gamma \vdash A \lesssim B$. Unlike Core Grady we have $\Gamma \vdash ? \lesssim A$ and $\Gamma \vdash A \lesssim ?$ for any type A. This gives us some flexibility when instantiating polymorphic functions. For example, suppose $\Gamma \vdash_{\mathsf{SG}} t : \forall(X <: \mathsf{Nat}).(X \to X)$. Then, $\Gamma \vdash_{\mathsf{SG}} [?]t : ? \to ?$ is typable, as well as, $\Gamma \vdash_{\mathsf{SG}} [\mathsf{Nat}]t : \mathsf{Nat} \to \mathsf{Nat}$ by subsumption. Similarly, if $\Gamma \vdash_{\mathsf{SG}} t : \forall(X <: ?).(X \to X)$, then we can instantiate t with any type at all. This seems very flexible, but it turns out that it does nothing more than what Core Grady allows when adding explicit casts.

Contributions. This paper offers the following contributions:

- The first gradual type system with bounded quantification. The core casting calculus is based on Simply Typed Grady [**?**] and Bounded System F [11].

We show that the bounds on types can be used to restrict which types can be cast to the unknown type and vice versa.
- We prove the gradual guarantee for our gradual type system.
- We show that explicit casting in the surface language of gradual type systems are derivable, and making use of these explicit casts increases the expressiveness of the language.

2 Related Work

We now give a brief summary of related work. Each of the articles discussed below can be consulted for further references.

- Abadi et al. [1] combine dynamic and static typing by adding a new type called Dynamic along with a new case construct for pattern matching on types. We do not add such a case construct, and as a result, show that we can obtain a surprising amount of expressivity without it. They also provide denotational models.

- Henglein [5] defines the dynamic λ-calculus by adding a new type Dyn to the simply typed λ-calculus and then adding primitive casting operations called tagging and check-and-untag. These new operations tag type constructors with their types. Then untagging checks to make sure the target tag matches the source tag, and if not, returns a dynamic type error. These operations can be used to build casting coercions which are very similar to our casting morphisms. We can also define split, squash, box, and unbox in terms of Henglein's casting coercions. We consider our previous paper [?] as a clarification of Henglein's system. His core casting calculus can be interpreted into our setting where we require retracts instead of full isomorphisms. This paper can be seen as a further extension of this type of work to include bounded quantification.

- There is a long history of polymorphism in both static and dynamic typing, and in systems that combine static and dynamic typing. Henglein and Rehof [6,12] show how to extend Henglein's previous work on combining static and dynamic typing discussed above. This work improves on their work by considering bounded quantification and adding a gradually typed surface language.

  Matthews and Ahmed [10] extend Girard/Reynolds' System F with static and dynamic typing where the unknown type is allowed to be cast to a type variable and vice versa. They call this type of cast "consealing". This was extended by Ahmed et al. [2] into a casting calculus with blame that included the ability to cast a polymorphic type to the unknown type and vice versa. Grady also includes the ability to box and unbox a type variable, but we have chosen not to allow one to cast a polymorphic type to the unknown type or vice versa. Currently, we are unsure how this will affect gradual typing, and we do not have a denotational model that allows this. We hope to include this feature in future work.

Ahmed et al. [2] has some similarities to this paper. Their "compatibility" relation is similar to type consistency, and they also discuss subtyping. However, they do not give a gradually typed surface language. In addition, our system can be seen as a further clarification of the underlying structure of the casting fragment of their system. We do not have full explicit casts of the form $t : A \Rightarrow B$, but instead only have box, unbox, split, and squash.

– As we mentioned in the introduction Siek and Taha [15] were the first to define gradual typing especially for functional languages, but only for simple types. Since their original paper introducing gradual types lots of languages have adopted it, but the term "gradual typing" started to become a catch all phrase for any language combining dynamic and static typing. As a result of this Siek et al. [16] later refine what it means for a language to support gradual typing by specifying the necessary metatheoretic properties a gradual type system must satisfy called the gradual guarantee. We prove the gradual guarantee for Grady in Section 4.

  Subtyping for gradual type systems was introduced by Siek and Taha [14], and then further extended by Garcia [4]. However, neither consider polymorphism. The subtyping system for Grady is based on Garcia's work. He does indeed prove the gradual guarantee, but again, his work does not consider polymorphism.

3 Grady: A Categorically Inspired Gradual Type System

We begin by introducing the surface and core languages making up Grady. Throughout this section we give a number of interesting examples. All example programs will be given in the concrete syntax of Grady[1]. Both the surface and the core languages are based on Bounded System F; for an introduction please see Pierce [11].

3.1 Surface Grady: A Gradual Type System

The aim of the gradual typing is to allow the programmer to program either statically and catch as many errors at compile time as possible, or to program in dynamic style and leave some error checking to run time, but the programmer should not be burdened by having to explicitly insert casts. Surface Grady is gradually typed and in this section we give the details of the language.

Surface Grady's syntax is defined in Fig. 1. The types $\top$ and $\mathbb{S}$ will be used

[1] The implementation and documentation of Grady can be found at `http://www.ct-gradual-typing.github.io/Grady`.

Syntax:

$$\begin{array}{lrl}
\text{(types)} & A, B, C ::= & X \mid \top \mid \mathbb{S} \mid \mathsf{Unit} \mid \mathsf{Nat} \mid ? \mid \mathsf{List}\,A \mid A \times B \mid A \to B \\
 & \mid & \forall (X <: A).B \\[1em]
\text{(skeletons)} & S, K, U ::= & ? \mid \mathsf{List}\,S \mid S_1 \times S_2 \mid S_1 \to S_2 \\[1em]
\text{(terms)} & t ::= & x \mid \mathsf{triv} \mid 0 \mid \mathsf{succ}\,t \mid \mathsf{case}\,t\,\mathsf{of}\,0 \to t_1, (\mathsf{succ}\,x) \to t_2 \\
 & \mid & (t_1, t_2) \mid \mathsf{fst}\,t \mid \mathsf{snd}\,t \mid [] \mid t_1 :: t_2 \\
 & \mid & \mathsf{case}\,t\,\mathsf{of}\,[] \to t_1, (x :: y) \to t_2 \mid \lambda(x : A).t \\
 & \mid & t_1\,t_2 \mid \Lambda(X <: A).t \mid [A]t \\[1em]
\text{(contexts)} & \Gamma ::= & \cdot \mid x : A \mid \Gamma_1, \Gamma_2
\end{array}$$

Metafunctions:

$$\begin{array}{ll}
\mathsf{nat}(?) = \mathsf{Nat} & \mathsf{list}(?) = \mathsf{List}\,? \\
\mathsf{nat}(\mathsf{Nat}) = \mathsf{Nat} & \mathsf{list}(\mathsf{List}\,A) = \mathsf{List}\,A \\[1em]
\mathsf{prod}(?) = ? \times ? & \mathsf{fun}(?) = ? \to ? \\
\mathsf{prod}(A \times B) = A \times B & \mathsf{fun}(A \to B) = A \to B
\end{array}$$

Fig. 1. Syntax and Metafunctions for Surface Grady

strictly as upper bounds with respect to quantification, and so, they will not have any introduction typing rules. The type of polymorphic functions is $\forall(X <: A).B$ where A is the called the bound on the type variable X. This bound will restrict which types are allowed to replace X during type application – also known as instantiation. The restriction is that the type one wishes to replace X with must be a subtype of the bounds A. As we mentioned in the introduction the type ? is the unknown type and should be thought of as the universe of untyped terms. Syntax for terms and typing contexts are fairly standard. One thing to note is that we do allow explicit boxing and unboxing, but not splitting and squashing, the latter are a purely core language feature. This is because any use of splitting or squashing can be algorithmically inserted, and so the programmer never has to worry about inserting those explicitly. Keep in mind that even though we allow explicitly boxing and unboxing in the surface language most places where boxing and unboxing would be used can be algorithmically inserted, and so the programmer should only use explicit boxing and unboxing in places where the algorithm fails.

The rules defining type consistency and consistent subtyping can be found in Fig. 2. In a gradual type system we use a reflexive and symmetric, but non-transitive, relation on types to determine when two types may be cast between each other [15]. This relation is called type consistency, and is denoted by $\Gamma \vdash A \sim B$. Non-transitivity prevents the system from being able to cast between arbitrary types. The type $\mathbb{S}$ stands for "simple types" and is a super type whose

Consistent Subtyping:

$$\frac{\Gamma \vdash A : \star}{\Gamma \vdash A \lesssim A}\,\text{refl} \qquad \frac{\Gamma \vdash A : \star}{\Gamma \vdash A \lesssim \top}\,\text{top} \qquad \frac{X <: A' \in \Gamma \quad \Gamma \vdash A' \sim A}{\Gamma \vdash X \lesssim A}\,\text{var}$$

$$\frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash A \lesssim ?}\,\mathsf{box} \qquad \frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash ? \lesssim A}\,\mathsf{unbox} \qquad \frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash A \lesssim ?}\,\mathsf{box}$$

$$\frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash ? \lesssim A}\,\mathsf{unbox} \qquad \frac{}{\Gamma \vdash ? \lesssim S}\,\mathsf{split} \qquad \frac{}{\Gamma \vdash S \lesssim ?}\,\mathsf{squash}$$

$$\frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash \mathsf{Nat} \lesssim \mathbb{S}}\,\mathsf{Nat}_{\mathcal{S}} \qquad \frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash \mathsf{Unit} \lesssim \mathbb{S}}\,\mathsf{Unit}_{\mathcal{S}} \qquad \frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash \mathsf{List}\,A \lesssim \mathbb{S}}\,\mathsf{List}_{\mathcal{S}}$$

$$\frac{\Gamma \vdash A \lesssim \mathbb{S} \quad \Gamma \vdash B \lesssim \mathbb{S}}{\Gamma \vdash A \times B \lesssim \mathbb{S}}\,\times_{\mathcal{S}} \qquad \frac{\Gamma \vdash A \lesssim \mathbb{S} \quad \Gamma \vdash B \lesssim \mathbb{S}}{\Gamma \vdash A \to B \lesssim \mathbb{S}}\,\to_{\mathcal{S}}$$

$$\frac{\Gamma \vdash A \lesssim B}{\Gamma \vdash (\mathsf{List}\,A) \lesssim (\mathsf{List}\,B)}\,\mathsf{List} \qquad \frac{\Gamma \vdash A_1 \lesssim A_2 \quad \Gamma \vdash B_1 \lesssim B_2}{\Gamma \vdash (A_1 \times B_1) \lesssim (A_2 \times B_2)}\,\times$$

$$\frac{\Gamma \vdash A_2 \lesssim A_1 \quad \Gamma \vdash B_1 \lesssim B_2}{\Gamma \vdash (A_1 \to B_1) \lesssim (A_2 \to B_2)}\,\to$$

$$\frac{\Gamma, X <: A \vdash B_1 \lesssim B_2}{\Gamma \vdash (\forall (X <: A).B_1) \lesssim (\forall (X <: A).B_2)}\,\forall$$

Type Consistency:

$$\frac{\Gamma \vdash A : \star}{\Gamma \vdash A \sim A}\,\text{refl} \qquad \frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash A \sim ?}\,\mathsf{box} \qquad \frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash ? \sim A}\,\mathsf{unbox}$$

$$\frac{}{\Gamma \vdash ? \sim S}\,\mathsf{split} \qquad \frac{}{\Gamma \vdash S \sim ?}\,\mathsf{squash} \qquad \frac{\Gamma \vdash A \sim B}{\Gamma \vdash (\mathsf{List}\,A) \sim (\mathsf{List}\,B)}\,\mathsf{List}$$

$$\frac{\Gamma \vdash A_2 \sim A_1 \quad \Gamma \vdash B_1 \sim B_2}{\Gamma \vdash (A_1 \to B_1) \sim (A_2 \to B_2)}\,\to \qquad \frac{\Gamma \vdash A_1 \sim A_2 \quad \Gamma \vdash B_1 \sim B_2}{\Gamma \vdash (A_1 \times B_1) \sim (A_2 \times B_2)}\,\times$$

$$\frac{\Gamma, X <: A \vdash B_1 \sim B_2}{\Gamma \vdash (\forall (X <: A).B_1) \sim (\forall (X <: A).B_2)}\,\forall$$

Fig. 2. Subtyping and Type Consistency for Surface Grady

subtypes are all non-polymorphic types that do not contain the unknown type. Thus, by the definition of type consistency the only types that can be boxed or unboxed are non-polymorphic types. Note that the type consistency rules box and unbox embody boxing and unboxing, and splitting and squashing. The remainder of the rules simply are congruence rules. We will use this intuition when translating Surface Grady to Core Grady. Note that the type consistency, subtyping, and typing judgments for both Surface Grady and Core Grady all depend on kinding, denoted $\Gamma \vdash A : \star$, and well-formed contexts, denoted $\Gamma\,\mathrm{Ok}$, but these are standard, and in the interest of saving space we do not define them here. They simply insure that all type variables in types in and out of contexts are accounted for.

Consistent subtyping, denoted $\Gamma \vdash A \lesssim B$, was proposed by Siek and Taha [14] in their work extending gradual type systems to object oriented programming. It embodies both standard subtyping, denoted $\Gamma \vdash A <: B$, and type consistency. Thus, consistent subtyping is also non-transitive. One major difference between this definition of consistent subtyping and others found in the literature, for example in [14] and [4], is the rule for type variables. Naturally, we must have a rule for type variables, because we are dealing with polymorphism, but the proof of the gradual guarantee – see Section 4 – required that this rule be relaxed and allow the bounds provided by the programmer to be consistent with the subtype in question.

Typing for Surface Grady is given in Fig. 3. It follows the formulation of the Gradual Simply Typed λ-calculus given by Siek et al. [16] pretty closely. The most interesting rules are the elimination rules, because this is where type consistency – and hence casting – comes into play. Consider the elimination for lists:

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathsf{SG}} t : C \quad \mathsf{list}(C) = \mathsf{List}\,A \\ \Gamma \vdash_{\mathsf{SG}} t_1 : B_1 \quad \Gamma, x : A, y : \mathsf{List}\,A \vdash_{\mathsf{SG}} t_2 : B_2 \quad \Gamma \vdash B_1 \sim B \quad \Gamma \vdash B_2 \sim B\end{array}}{\Gamma \vdash_{\mathsf{SG}} \mathsf{case}\,t\,\mathsf{of}\,[] \to t_1, (x :: y) \to t_2 : B}\,\mathsf{List}_e$$

The type C can be either $?$ or $\mathsf{List}\,A$. If it is the former, then C will be split into $\mathsf{List}\,?$. In addition, we allow the type of the branches to be cast to other types as well, just as long as, they are consistent with B. The idea is, the branches of a case expression, whether for lists or natural numbers, can have different types only when they can be cast to the same type. For example, if t_1 was a boolean and t_2 was a natural number, then a type checking will fail, because the types are not consistent, and hence, we cannot cast between them. The other rules are setup similarly.

One non-obvious feature of gradual typing is the ability to use explicit casts in the surface language without having the explicit casts as primitive features of the language. This realization actually increases the expressivity of the language. Consider the application rule for Surface Grady:

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathsf{SG}} t_1 : C \\ \Gamma \vdash_{\mathsf{SG}} t_2 : A_2 \quad \Gamma \vdash A_2 \sim A_1 \quad \mathsf{fun}(C) = A_1 \to B_1\end{array}}{\Gamma \vdash_{\mathsf{SG}} t_1\,t_2 : B_1}\,\to_e$$

$$\frac{x : A \in \Gamma \quad \Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{SG}} x : A}\,\mathrm{var} \qquad \frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{SG}} \mathsf{triv} : \mathsf{Unit}}\,\mathsf{Unit} \qquad \frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{SG}} 0 : \mathsf{Nat}}\,\mathsf{zero}$$

$$\frac{\Gamma \vdash_{\mathsf{SG}} t : A \quad \mathsf{nat}(A) = \mathsf{Nat}}{\Gamma \vdash_{\mathsf{SG}} \mathsf{succ}\, t : \mathsf{Nat}}\,\mathsf{succ}$$

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathsf{SG}} t : C \quad \mathsf{nat}(C) = \mathsf{Nat} \quad \Gamma \vdash A_1 \sim A\\ \Gamma \vdash_{\mathsf{SG}} t_1 : A_1 \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathsf{SG}} t_2 : A_2 \quad \Gamma \vdash A_2 \sim A\end{array}}{\Gamma \vdash_{\mathsf{SG}} \mathsf{case}\, t\, \mathsf{of}\, 0 \to t_1, (\mathsf{succ}\, x) \to t_2 : A}\,\mathsf{Nat}_e$$

$$\frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{SG}} [] : \forall(X <: \top).\mathsf{List}\, X}\,\mathrm{empty}$$

$$\frac{\Gamma \vdash_{\mathsf{SG}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{SG}} t_2 : A_2 \quad \mathsf{list}(A_2) = \mathsf{List}\, A_3 \quad \Gamma \vdash A_1 \sim A_3}{\Gamma \vdash_{\mathsf{SG}} t_1 :: t_2 : \mathsf{List}\, A_3}\,\mathsf{List}_i$$

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathsf{SG}} t : C \quad \mathsf{list}(C) = \mathsf{List}\, A\\ \Gamma \vdash_{\mathsf{SG}} t_1 : B_1 \quad \Gamma, x : A, y : \mathsf{List}\, A \vdash_{\mathsf{SG}} t_2 : B_2 \quad \Gamma \vdash B_1 \sim B \quad \Gamma \vdash B_2 \sim B\end{array}}{\Gamma \vdash_{\mathsf{SG}} \mathsf{case}\, t\, \mathsf{of}\, [] \to t_1, (x :: y) \to t_2 : B}\,\mathsf{List}_e$$

$$\frac{\Gamma \vdash_{\mathsf{SG}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{SG}} t_2 : A_2}{\Gamma \vdash_{\mathsf{SG}} (t_1, t_2) : A_1 \times A_2}\,\times_i \qquad \frac{\Gamma \vdash_{\mathsf{SG}} t : B \quad \mathsf{prod}(B) = A_1 \times A_2}{\Gamma \vdash_{\mathsf{SG}} \mathsf{fst}\, t : A_1}\,\times_{e_1}$$

$$\frac{\Gamma \vdash_{\mathsf{SG}} t : B \quad \mathsf{prod}(B) = A_1 \times A_2}{\Gamma \vdash_{\mathsf{SG}} \mathsf{snd}\, t : A_2}\,\times_{e_2} \qquad \frac{\Gamma, x : A \vdash_{\mathsf{SG}} t : B}{\Gamma \vdash_{\mathsf{SG}} \lambda(x : A).t : A \to B}\,\to_i$$

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathsf{SG}} t_1 : C\\ \Gamma \vdash_{\mathsf{SG}} t_2 : A_2 \quad \Gamma \vdash A_2 \sim A_1 \quad \mathsf{fun}(C) = A_1 \to B_1\end{array}}{\Gamma \vdash_{\mathsf{SG}} t_1\, t_2 : B_1}\,\to_e$$

$$\frac{\Gamma, X <: A \vdash_{\mathsf{SG}} t : B}{\Gamma \vdash_{\mathsf{SG}} \Lambda(X <: A).t : \forall(X <: A).B}\,\forall_i$$

$$\frac{\Gamma \vdash_{\mathsf{SG}} t : \forall(X <: B).C \quad \Gamma \vdash A \lesssim B}{\Gamma \vdash_{\mathsf{SG}} [A]t : [A/X]C}\,\forall_e \qquad \frac{\Gamma \vdash_{\mathsf{SG}} t : A \quad \Gamma \vdash A \lesssim B}{\Gamma \vdash_{\mathsf{SG}} t : B}\,\mathrm{sub}$$

Fig. 3. Typing rules for Surface Grady

Notice that this rule does not apply any implicit casts to the result of the application. Thus, if one needs to cast the result, then on first look, it would seem they are out of luck. However, if we push the cast into the argument position then this rule will insert the appropriate cast. This leads us to the following definitions:

$$\begin{array}{ll} \mathsf{box}_A\, t \;\;= (\lambda(x:?).x)\, t & \mathsf{squash}_S\, t = (\lambda(x:?).x)\, t \\ \mathsf{unbox}_A\, t = (\lambda(x:A).x)\, t & \mathsf{split}_S\, t \;\;= (\lambda(x:S).x)\, t \end{array}$$

The reader should keep in mind the differences between box and squash, and unbox and split. When these are translated into Core Grady using the cast insertion algorithm – see Sect. 3.3 – the inserted cast will match the definition. We now have the following result.

Lemma 1 (Explicit Casts Typing). *The following rules are derivable in Surface Grady:*

$$\frac{\Gamma \vdash_{\mathsf{SG}} t : A}{\Gamma \vdash_{\mathsf{SG}} \mathsf{box}_A\, t : ?}\,\mathsf{box} \qquad \frac{\Gamma \vdash_{\mathsf{SG}} t : A}{\Gamma \vdash_{\mathsf{SG}} \mathsf{unbox}_A\, t : A}\,\mathsf{unbox} \qquad \frac{\Gamma \vdash_{\mathsf{SG}} t : ?}{\Gamma \vdash_{\mathsf{SG}} \mathsf{split}_S\, t : S}\,\mathsf{split}$$

$$\frac{\Gamma \vdash_{\mathsf{SG}} t : S}{\Gamma \vdash_{\mathsf{SG}} \mathsf{squash}_S\, t : ?}\,\mathsf{squash}$$

As an example consider the following Surface Grady program – here we use the concrete syntax from Grady's implementation, but it is very similar to Haskell and not far from the mathematical syntax:

```
omega : ? → ?
omega = \(x : ?) → (x x);

ycomb : (? → ?) → ?
ycomb = \(f : ? → ?) → omega (\(x:?) → f (x x));

fix : forall (X <: Simple).((X → X) → X)
fix = \(X <: Simple) → \(f:X → X) → unbox<X> (ycomb f);
```

The previous example defines the Y combinator and a polymorphic fix point operator. This example is gradually typed, for example, in the definition of `omega` we are applying `x` to itself without an explicit cast. However, there is one explicit cast in the definition of `fix` which applies `unbox` to the result of the Y combinator. If we did not have this explicit cast, then `fix` would not type check for the reasons outlines above. Thus, pointing out the explicit casts in the surface language increases the number of valid programs. We will use `fix` to develop quite a few interesting examples including a full library of operations on lists.

Being able to define the typed fix point operator makes Grady very expressive. Combing `fix` with the eliminators for natural numbers and lists results in typed terminating recursion. We now give several examples in Surface Grady that illustrate this[2]:

```
append : forall (A <: Simple).([A] → [A] → [A])
append = \(A <: Simple) →
           ([ [A] → [A] → [A] ] fix)
             (\(r : [A] → [A] → [A]) → \(l1 : [A]) → \(l2 : [A]) →
               case l1 of
                 [] → l2,
                 (a :: as) → a ::(r as l2));

length : forall (A <: Simple).([A] → Nat)
length = \(A <: Simple) → ([ [A] → Nat]fix)
             (\(r : [A] → Nat) → \(l : [A]) →
               case l of
                 [] → 0,
                 (a :: as) → succ (r as));

foldr : forall (A <: Simple).
        (forall (B <: Simple).((A → B → B) → B → ([A] → B)))
foldr = \(A <: Simple) → \(B <: Simple) → \(f : A → B → B) → \(b : B) →
         ([ [A] → B]fix) (\(r : [A] → B) → \(l : [A]) →
                   case l of
                     [] → b,
                     (a :: as) → (f a (r as)));

foldl : forall (A <: Simple).
        (forall (B <: Simple).((B → A → B) → (B → [A] → B)))
foldl = \(A <: Simple) → \(B <: Simple) → \(f : B → A → B) →
         ([ B → [A] → B]fix) (\(r : B → [A] → B) → \(b : B) → \(l : [A]) →
                   case l of
                     [] → b,
                     (a :: as) → r (f b a) as);

zipWith : forall (A <: Simple).(forall (B <: Simple).(forall (C <: Simple).
                ((A → B → C) → ([A] → [B] → [C]))))
zipWith = \(A <: Simple) →
             \(B <: Simple) → \(C <: Simple) → \(f : A → B → C) →
            ([ [A] → [B] → [C] ] fix)
              (\(r : [A] → [B] → [C]) → \(l1 : [A]) → \(l2 : [B]) →
                 case l1 of
                   [] → [C][],
                   (a :: as) → case l2 of
                                  [] → [C][],
                                  (b :: bs) → (f a b) :: (r as bs));
```

All of the previous examples are staticly typed, but eventually the static types are boxed and moved to the dynamic fragment when running `fix`.

[2] Please see the following example file for the complete list library: `https://github.com/ct-gradual-typing/Grady/blob/master/Examples/Gradual/List.gry`

3.2 Core Grady: The Casting Calculus

Core Grady is a non-gradual type system that combines both static and dynamic typing. It is an extension of the authors previous work [**?**], adding bounded quantification and lists, on combing the work of Scott [13] and Lambek [8]. Furthermore, Core Grady is a simple extension of Bounded System F. The syntax for Core Grady can be found in Fig. 4. The syntax of types and typing context

$$
\begin{array}{rl}
\text{(skeletons)} & S, K, U ::= ? \mid \mathsf{List}\, S \mid S_1 \times S_2 \mid S_1 \to S_2 \\[1em]
\text{(terms)} & t ::= \cdots \mid \mathsf{squash}_S \mid \mathsf{split}_S \mid \mathsf{error}_A
\end{array}
$$

Fig. 4. Syntax for Core Grady

for Core Grady are exactly the same as Surface Grady, and so we do not repeat them here. The syntax of terms is an extension of the syntax for Surface Grady, and so we only show the additions. The term error_A will be used to trigger a type error during run time.

Subtyping for Core Grady is as one might expect for Bounded System F. The rules for subtyping are given in Fig. 5. Note that Core Grady does not

$$
\frac{\Gamma \vdash A : \star}{\Gamma \vdash A <: A}\ \text{refl}
\qquad
\frac{\Gamma \vdash A : \star}{\Gamma \vdash A <: \top}\ \text{top}
\qquad
\frac{X <: A \in \Gamma \quad \Gamma\ \text{Ok}}{\Gamma \vdash X <: A}\ \text{var}
$$

$$
\frac{\Gamma\ \text{Ok}}{\Gamma \vdash \mathsf{Nat} <: \mathbb{S}}\ \mathsf{Nat}_{\mathcal{S}}
\qquad
\frac{\Gamma\ \text{Ok}}{\Gamma \vdash \mathsf{Unit} <: \mathbb{S}}\ \mathsf{Unit}_{\mathcal{S}}
\qquad
\frac{\Gamma \vdash A <: \mathbb{S}}{\Gamma \vdash \mathsf{List}\, A <: \mathbb{S}}\ \mathsf{List}_{\mathcal{S}}
$$

$$
\frac{\Gamma \vdash A <: \mathbb{S} \quad \Gamma \vdash B <: \mathbb{S}}{\Gamma \vdash A \to B <: \mathbb{S}}\ \to_{\mathcal{S}}
\qquad
\frac{\Gamma \vdash A <: \mathbb{S} \quad \Gamma \vdash B <: \mathbb{S}}{\Gamma \vdash A \times B <: \mathbb{S}}\ \times_{\mathcal{S}}
$$

$$
\frac{\Gamma \vdash A <: B}{\Gamma \vdash \mathsf{List}\, A <: \mathsf{List}\, B}\ \mathsf{List}
\qquad
\frac{\Gamma \vdash A_1 <: A_2 \quad \Gamma \vdash B_1 <: B_2}{\Gamma \vdash A_1 \times B_1 <: A_2 \times B_2}\ \times
$$

$$
\frac{\Gamma \vdash A_2 <: A_1 \quad \Gamma \vdash B_1 <: B_2}{\Gamma \vdash A_1 \to B_1 <: A_2 \to B_2}\ \to
\qquad
\frac{\Gamma, X <: A \vdash B_1 <: B_2}{\Gamma \vdash \forall(X <: A).B_1 <: \forall(X <: A).B_2}\ \forall
$$

Fig. 5. Subtyping for Core Grady

depend on type consistency, this is purely a surface language feature. In addition, subtyping in the core is also non-transitive just like subtyping in Surface Grady. We axiomatize the super type $\mathbb{S}$ in the same way as Surface Grady.

Similarly to subtyping, typing for Core Grady is a simple extension of typing for Bounded System F. The most interesting rules here are the rules for splitting and squashing. Core Grady has the following types of casts:

$$\begin{array}{llll} \text{(boxing)} & \mathsf{box}_A : A \to ? & \text{(splitting)} & \mathsf{split}_S : ? \to S \\ \text{(unboxing)} & \mathsf{unbox}_A : ? \to A & \text{(squashing)} & \mathsf{squash}_S : S \to ? \end{array}$$

These casts are enough to do everything the surface language can do. In addition, the general casting rule used in the casting calculi found in the gradual typing literature, e.g. [14,15,2,16], denoted $t : A \Rightarrow B$ can be modeled by these four operations [**?**].

Unlike Surface Grady, Core Grady has a reduction relation. The surface language will then be translated into the core language where evaluation will take place. The reduction relation is defined in Fig. 7. Reduction is a extended version of call-by-name. We omit congruence rules for brevity. Reduction will not reduce under λ-abstractions or arguments to box or squash. However, it will reduce arguments to unbox and split in order to insure the retract rules apply as much as possible.

3.3 Cast Insertion

Surface Grady is translated into Core Grady by the cast insertion algorithm. Any where type consistency or one of the metafunctions defined in Fig. 1 are used a cast must be inserted.

We call a Core Grady expression that is built from box, unbox, split, squash, the functors $\mathsf{List}\,-$, $- \times -$, and $- \to -$, and the identity function a casting morphism. Each of the functors are definable as metafunctions:

(List functor)
$\mathsf{List}\, t := \mathsf{fix}\,(\lambda(r : \mathsf{List}\, A \to \mathsf{List}\, B).\lambda(x : \mathsf{List}\, A).\mathsf{case}\, x \colon \mathsf{List}\, A\, \mathsf{of}\, [] \to [], (y :: ys) \to (t\, y) :: (r\, ys))$ given $\Gamma \vdash_{\mathsf{CG}} t : A \to B$

(Product functor)
$t_1 \times t_2 := \lambda(x : A \times B).(t_1\,(\mathsf{fst}\, x), t_2\,(\mathsf{snd}\, x))$ given $\Gamma \vdash_{\mathsf{CG}} t_1 : A \to D$ and $\Gamma \vdash_{\mathsf{CG}} t_2 : B \to E$

(Internal Hom Functor)
$t_1 \to t_2 := \lambda(f : A \to B).\lambda(y : D).t_2\,(f\,(t_1\, y))$ given $\Gamma \vdash_{\mathsf{CG}} t_1 : D \to A$ and $\Gamma \vdash_{\mathsf{CG}} t_2 : B \to E$

The definition of the list functor is the usual definition of the map function for lists which requires the use of the fix point operator given in the introduction.

In the authors previous work [**?**] they showed a similar result to the following. In fact, their proofs are nearly identical, and so we do not give the proof here.

Lemma 2 (Casting Morphisms). *If $\Gamma \vdash A \sim B$, then there are casting morphisms $\Gamma \vdash_{\mathsf{CG}} c_1 : A \to B$ and $\Gamma \vdash_{\mathsf{CG}} c_2 : B \to A$.*

$$\frac{x : A \in \Gamma \quad \Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} x : A}\,\mathrm{var} \qquad \frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{box} : \forall(X <: \mathbb{S}).(X \to ?)}\,\mathsf{box}$$

$$\frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{unbox} : \forall(X <: \mathbb{S}).(? \to X)}\,\mathsf{unbox} \qquad \frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{squash}_S : S \to ?}\,\mathsf{squash}$$

$$\frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{split}_S : ? \to S}\,\mathsf{split} \qquad \frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{triv} : \mathsf{Unit}}\,\mathsf{Unit} \qquad \frac{\Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{CG}} 0 : \mathsf{Nat}}\,\mathrm{zero}$$

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : \mathsf{Nat}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{succ}\, t : \mathsf{Nat}}\,\mathsf{succ} \qquad \frac{\begin{array}{c}\Gamma \vdash_{\mathsf{CG}} t : \mathsf{Nat} \\ \Gamma \vdash_{\mathsf{CG}} t_1 : A \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathsf{CG}} t_2 : A\end{array}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{case}\, t : \mathsf{Nat}\,\mathsf{of}\, 0 \to t_1, (\mathsf{succ}\, x) \to t_2 : A}\,\mathsf{Nat}_e$$

$$\frac{\Gamma\,\mathrm{Ok} \quad \Gamma \vdash A : \star}{\Gamma \vdash_{\mathsf{CG}} [] : \forall(X <: ?).\mathsf{List}\, X}\,\mathrm{empty} \qquad \frac{\Gamma \vdash_{\mathsf{CG}} t_1 : A \quad \Gamma \vdash_{\mathsf{CG}} t_2 : \mathsf{List}\, A}{\Gamma \vdash_{\mathsf{CG}} t_1 :: t_2 : \mathsf{List}\, A}\,\mathsf{List}_i$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{CG}} t : \mathsf{List}\, A \\ \Gamma \vdash_{\mathsf{CG}} t_1 : B \quad \Gamma, x : A, y : \mathsf{List}\, A \vdash_{\mathsf{CG}} t_2 : B\end{array}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{case}\, t : \mathsf{List}\, A\,\mathsf{of}\, [] \to t_1, (x :: y) \to t_2 : B}\,\mathsf{List}_e$$

$$\frac{\Gamma \vdash_{\mathsf{CG}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{CG}} t_2 : A_2}{\Gamma \vdash_{\mathsf{CG}} (t_1, t_2) : A_1 \times A_2}\,\times_i \qquad \frac{\Gamma \vdash_{\mathsf{CG}} t : A_1 \times A_2}{\Gamma \vdash_{\mathsf{CG}} \mathsf{fst}\, t : A_1}\,\times_{e_1}$$

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : A_1 \times A_2}{\Gamma \vdash_{\mathsf{CG}} \mathsf{snd}\, t : A_2}\,\times_{e_2} \qquad \frac{\Gamma, x : A \vdash_{\mathsf{CG}} t : B}{\Gamma \vdash_{\mathsf{CG}} \lambda(x : A).t : A \to B}\,\to_i$$

$$\frac{\Gamma \vdash_{\mathsf{CG}} t_1 : A \to B \quad \Gamma \vdash_{\mathsf{CG}} t_2 : A}{\Gamma \vdash_{\mathsf{CG}} t_1\, t_2 : B}\,\to_e \qquad \frac{\Gamma, X <: A \vdash_{\mathsf{CG}} t : B}{\Gamma \vdash_{\mathsf{CG}} \Lambda(X <: A).t : \forall(X <: A).B}\,\forall_i$$

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : \forall(X <: B).C \quad \Gamma \vdash A <: B}{\Gamma \vdash_{\mathsf{CG}} [A]t : [A/X]C}\,\forall_e \qquad \frac{\Gamma \vdash_{\mathsf{CG}} t : A \quad \Gamma \vdash A <: B}{\Gamma \vdash_{\mathsf{CG}} t : B}\,\mathrm{sub}$$

$$\frac{}{\Gamma \vdash_{\mathsf{CG}} \mathsf{error}_A : A}\,\mathrm{error}$$

Fig. 6. Typing rules for Core Grady

$$\frac{}{\mathsf{unbox}_A\,(\mathsf{box}_A\ t) \rightsquigarrow t}\,\text{retract}_1 \qquad \frac{A \neq B}{\mathsf{unbox}_A\,(\mathsf{box}_B\ t) \rightsquigarrow \mathsf{error}_A}\,\text{error}_1$$

$$\frac{}{\mathsf{split}_S\,(\mathsf{squash}_S\ t) \rightsquigarrow t}\,\text{retract}_2 \qquad \frac{S_1 \neq S_2}{\mathsf{split}_{S_1}\,(\mathsf{squash}_{S_2}\ t) \rightsquigarrow \mathsf{error}_{S_1}}\,\text{error}_2$$

$$\frac{}{\mathsf{case}\,0\colon \mathsf{Nat\ of}\ 0 \to t_1, (\mathsf{succ}\ x) \to t_2 \rightsquigarrow t_1}\,\mathsf{Nat}_{e_1}$$

$$\frac{}{\mathsf{case}\,(\mathsf{succ}\ t)\colon \mathsf{Nat\ of}\ 0 \to t_1, (\mathsf{succ}\ x) \to t_2 \rightsquigarrow [t/x]t_2}\,\mathsf{Nat}_{e_2}$$

$$\frac{}{\mathsf{case}\,[\,]\colon \mathsf{List}\ A\ \mathsf{of}\ [\,] \to t_1, (x :: y) \to t_2 \rightsquigarrow t_1}\,\mathsf{List}_{e_1}$$

$$\frac{}{\mathsf{case}\,(t_1 :: t_2)\colon \mathsf{List}\ A\ \mathsf{of}\ [\,] \to t_3, (x :: y) \to t_4 \rightsquigarrow [t_1/x][t_2/y]t_4}\,\mathsf{List}_{e_2}$$

$$\frac{}{\mathsf{fst}\,(t_1, t_2) \rightsquigarrow t_1}\,\times_{e_1} \qquad \frac{}{\mathsf{snd}\,(t_1, t_2) \rightsquigarrow t_2}\,\times_{e_2} \qquad \frac{}{(\lambda(x : A_1).t_2)\ t_1 \rightsquigarrow [t_1/x]t_2}\,\beta$$

$$\frac{}{[A](\Lambda(X <: B).t) \rightsquigarrow [A/X]t}\,\text{type}_\beta$$

Fig. 7. Reduction rules for Core Grady

Using this result we can define the cast insertion algorithm which takes in a Surface Grady term and then returns a Core Grady term by inserting casting morphisms where type consistency is used. Thus, this algorithm is type directed. Its definition is given in Fig. 8. We only give the most interesting cases, because the others are either trivial identity cases or are similar to the ones given. We denote constructing the casting morphism for $\Gamma \vdash A \sim B$ by $\mathsf{caster}(A, B) = c$.

As an example the following is the result of applying the cast insertion algorithm – after some simplifications – to the fix point operator given in the introduction:

```
omega : (? → ?) → ?
omega = \(x : ? → ?) → (x (squash (? → ?) x));

ycomb : (? → ?) → ?
ycomb = \(f : ? → ?) → omega (\(x:?) → f ((split (? → ?) x) x));

fix : forall (X <: Simple).((X → X) → X)
fix = \(X <: Simple) → \(f:X → X) →
        unbox<X> (ycomb (\(y:?)→ box<X> (f (unbox<X> y))));
```

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{succ}\, t_1 \Rightarrow \mathsf{succ}\,(\mathsf{unbox}_{\mathsf{Nat}}\, t_2) : \mathsf{Nat}} \qquad \frac{\Gamma \vdash t_1 \Rightarrow t_2 : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, t_1 \Rightarrow \mathsf{succ}\, t_2 : \mathsf{Nat}}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{fst}\, t_1 \Rightarrow \mathsf{fst}\,(\mathsf{split}_{(?\times?)}\, t_2) : ?} \qquad \frac{\Gamma \vdash t_1 \Rightarrow t_2 : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\, t_1 \Rightarrow \mathsf{fst}\, t_2 : A_1}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{snd}\, t_1 \Rightarrow \mathsf{snd}\,(\mathsf{split}_{(?\times?)}\, t_2) : ?} \qquad \frac{\Gamma \vdash t_1 \Rightarrow t_2 : A \times B}{\Gamma \vdash \mathsf{snd}\, t_1 \Rightarrow \mathsf{snd}\, t_2 : B}$$

$$\frac{\begin{array}{c}\Gamma \vdash t \Rightarrow t' : ? \quad \mathsf{caster}(B_1, B) = c_1 \quad \mathsf{caster}(B_2, B) = c_2 \\ \Gamma \vdash t_1 \Rightarrow t_1' : B_1 \quad \Gamma, x : ?, y : \mathsf{List}\,? \vdash t_2 \Rightarrow t_2' : B_2 \quad \Gamma \vdash B_1 \sim B \quad \Gamma \vdash B_2 \sim B\end{array}}{\Gamma \vdash (\mathsf{case}\, t\, \mathsf{of}\, [] \to t_1, (x :: y) \to t_2) \Rightarrow (\mathsf{case}\,(\mathsf{split}_{(\mathsf{List}\,?)}\, t')\, \mathsf{of}\, [] \to (c_1\, t_1'), (x :: y) \to (c_2\, t_2')) : B}$$

$$\frac{\begin{array}{c}\Gamma \vdash t \Rightarrow t : \mathsf{List}\, A \quad \mathsf{caster}(B_1, B) = c_1 \quad \mathsf{caster}(B_2, B) = c_2 \\ \Gamma \vdash t_1 \Rightarrow t_1' : B_1 \quad \Gamma, x : A, y : \mathsf{List}\, A \vdash t_2 \Rightarrow t_2' : B_2 \quad \Gamma \vdash B_1 \sim B \quad \Gamma \vdash B_2 \sim B\end{array}}{\Gamma \vdash (\mathsf{case}\, t\, \mathsf{of}\, [] \to t_1, (x :: y) \to t_2) \Rightarrow (\mathsf{case}\, t'\, \mathsf{of}\, [] \to (c_1\, t_1'), (x :: y) \to (c_2\, t_2')) : B}$$

$$\frac{\begin{array}{l}\Gamma \vdash t_1 \Rightarrow t_1' : ? \\ \Gamma \vdash t_2 \Rightarrow t_2' : A_2 \quad \mathsf{caster}(A_2, ?) = c\end{array}}{\Gamma \vdash t_1\, t_2 \Rightarrow (\mathsf{split}_{(?\to?)}\, t_1')\,(c\, t_2') : ?}$$

$$\frac{\begin{array}{l}\Gamma \vdash t_2 \Rightarrow t_2' : A_2 \\ \Gamma \vdash t_1 \Rightarrow t_1' : A_1 \to B \quad \Gamma \vdash A_2 \sim A_1 \quad \mathsf{caster}(A_2, A_1) = c\end{array}}{\Gamma \vdash t_1\, t_2 \Rightarrow t_1'\,(c\, t_2') : B}$$

Fig. 8. Cast Insertion Algorithm

4 Analyzing Grady

We now turn to Grady's metatheory. Specifically, we show that the gradual guarantee – see Theorem 1 – holds for Grady. This is the defining property of every gradual type system. In fact, Siek et al. [16] argue that the gradual guarantee is what separates type systems that simply combine dynamic and static typing from systems that not only combine dynamic and static typing, but also allow the programmer to program in dynamic style without the need to insert explicit casts. Intuitively, the gradual guarantee states that a gradually typed program should preserve its type and behavior when explicit casts are either inserted or removed.

First, we have some basic facts about Grady. Notice that if we remove the unknown type, box, and unbox from Grady, then we are left with Bounded System F. Suppose $\Gamma \vdash_F t : A$ holds if t and A are a Bounded System F – for its definition please see Pierce[11] – term and type respectively. We call a type A, static, if it does not contain the unknown type.

Lemma 3 (Inclusion of Bounded System F). *Suppose t is fully annotated and does not contain any applications of* box *or* unbox*, and A is static. Then*

i. $\Gamma \vdash_F t : A$ *if and only if* $\Gamma \vdash_{\mathsf{SG}} t : A$*, and*
ii. $t \rightsquigarrow_F^* t'$ *if and only if* $t \rightsquigarrow^* t'$*.*

Proof. We give proof sketches for both parts. The interesting cases are the right-to-left directions of each part. If we simply remove all rules mentioning the unknown type ? and the type consistency relation, and then remove box, unbox, and ? from the syntax of Surface Grady, then what we are left with is bounded system F. Since t is fully annotated and A is static, then $\Gamma \vdash_{\mathsf{SG}} t : A$ will hold within this fragment.

Moving on to part two, first, we know that t does not contain any occurrence of box or unbox and is fully annotated. This implies that t lives within the bounded system F fragment of Surface Grady. Thus, before evaluation of t Surface Grady will apply the cast insertion algorithm which will at most insert applications of the identity function into t producing a term $\widehat{t}$, but then after potentially more than one step of evaluation within Core Grady, those applications of the identity function will be β-reduced away resulting in $\widehat{t} \rightsquigarrow^* t \rightsquigarrow^* t'$. In addition, since t in Surface Grady is the exact same program as t in bounded system F, then we know $t \rightsquigarrow_F^* t'$ will hold.

The call-by-name untyped λ-calculus is a fragment of Grady. We now show that one can strictly program in this fragment. In fact, the untyped fragment of Grady is the call-by-name unitype λ-calculus. The translation of the untyped λ-calculus into Grady is as follows:

$$\begin{aligned}
\lceil x \rceil &= x \\
\lceil \lambda x.t \rceil &= \lambda(x : ?).\lceil t \rceil \\
\lceil t_1\, t_2 \rceil &= ((\lambda(x : ? \to ?).x)\, \lceil t_1 \rceil)\, \lceil t_2 \rceil
\end{aligned}$$

In the last case of the previous definition the annotation on x will force the $\lceil t_1 \rceil$ to be split into $? \to ?$, and thus, after the final application we will have a term of type ?. We now have the following result.

Lemma 4 (Inclusion of the Untyped λ-Calculus). *Suppose t is a closed term of the untyped λ-calculus. Then*

i. $\cdot \vdash_{\mathsf{SG}} \lceil t \rceil : ?$*, and*
ii. $t \rightsquigarrow_{\lambda}^{*} t'$ *if and only if* $\lceil t \rceil \rightsquigarrow^{*} \lceil t' \rceil$.

Proof. This proof is the same result proven by [15], and [16].

Another basic fact is that type preservation holds for Core Grady.

Lemma 5 (Type Preservation). *If* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$ *and* $t_1 \rightsquigarrow t_2$*, then* $\Gamma \vdash_{\mathsf{CG}} t_2 : A$.

Proof. This proof holds by induction on $\Gamma \vdash_{\mathsf{CG}} t_1 : A$ with further case analysis on the structure the derivation $t_1 \rightsquigarrow t_2$.

This result is needed in the proof of the gradual guarantee, specifically, in the proof of simulation of more precise programs (Lemma 9).

We claimed that consistent subtyping is the combination of both standard subtyping and type consistency. The following results makes this precise.

Lemma 6 (Left-to-Right Consistent Subtyping). *Suppose* $\Gamma \vdash A \lesssim B$.

i. $\Gamma \vdash A \sim A'$ *and* $\Gamma \vdash A' <: B$ *for some* A'.
ii. $\Gamma \vdash B' \sim B$ *and* $\Gamma \vdash A <: B'$ *for some* B'.

Proof. This is a proof by induction on $\Gamma \vdash A \lesssim B$. See Appendix B.1 for the complete proof.

Corollary 1 (Consistent Subtyping).

i. $\Gamma \vdash A \lesssim B$ *if and only if* $\Gamma \vdash A \sim A'$ *and* $\Gamma \vdash A' <: B$ *for some* A'.
ii. $\Gamma \vdash A \lesssim B$ *if and only if* $\Gamma \vdash B' \sim B$ *and* $\Gamma \vdash A <: B'$ *for some* B'.

Proof. The left-to-right direction of both cases easily follows from Lemma 6, and the right-to-left direction of both cases follows from induction on the subtyping derivation and Lemma 28.

The previous two results are similar to those proved by Siek and Taha [14] and Garcia [4].

We now embark on the proof of the gradual guarantee. Our proof follows the scheme adopted by Siek et al. [16] in their proof of the gradual guarantee for the gradual simply typed λ-calculus. That is, we prove the exact same results as they do.

Cast insertion preserves type up to type consistency. This is fairly straightforward to show. The only interesting case is the one for type application.

Lemma 7 (Type Preservation for Cast Insertion). *If $\Gamma \vdash_{\mathsf{SG}} t_1 : A$ and $\Gamma \vdash t_1 \Rightarrow t_2 : B$, then $\Gamma \vdash_{\mathsf{CG}} t_2 : B$ and $\Gamma \vdash A \sim B$.*

Proof. The cast insertion algorithm is type directed and with respect to every term t_1 it will produce a term t_2 of the core language with the type A – this is straightforward to show by induction on the form of $\Gamma \vdash_{\mathsf{SG}} t_1 : A$ making use of typing for casting morphisms Lemma 32 – except in the case of type application. Please see Appendix B.5 for the complete proof.

The proof of the gradual guarantee will be phrased in terms of precision for types and terms. A type B is less precise than a type A, denoted $A \sqsubseteq B$, if B replaces some subexpression(s) of A with the unknown type. Type precision is defined by the rules in Fig. 9. Term precision, which is defined for both the

$$\frac{\Gamma \vdash A \lesssim \mathbb{S}}{A \sqsubseteq ?}\,? \qquad \frac{}{S \sqsubseteq ?}\,\text{skel} \qquad \frac{}{A \sqsubseteq A}\,\text{refl} \qquad \frac{A \sqsubseteq C \quad B \sqsubseteq D}{(A \to B) \sqsubseteq (C \to D)}\,\to$$

$$\frac{A \sqsubseteq C \quad B \sqsubseteq D}{(A \times B) \sqsubseteq (C \times D)}\,\times \qquad \frac{A \sqsubseteq B}{(\mathsf{List}\,A) \sqsubseteq (\mathsf{List}\,B)}\,\mathsf{List}$$

$$\frac{B_1 \sqsubseteq B_2}{(\forall (X <: A).B_1) \sqsubseteq (\forall (X <: A).B_2)}\,\forall$$

Fig. 9. Type Precision

surface language and the core language, is similar to type precision, but, and this is where our definition differs from others, we must factor in the explicit casts. Term precision is defined in Fig. 10. Term precision for Core Grady is an extension of term precision for Surface Grady. Thus, we only show the new rules for casting and error. As we travel up a term precision chain the terms type tend toward the unknown type, as opposed to, when we travel down the chain the terms type tend toward some static type. The terms in this chain only differ by the insertion or removal of casts. This characterizes our desired property, which is, that a gradual program can transition towards dynamic or static typing without compromising it behavior.

The gradual guarantee is defined as follows.

Theorem 1 (Gradual Guarantee).

i. If $\cdot \vdash_{\mathsf{SG}} t : A$ and $t \sqsubseteq t'$, then $\cdot \vdash_{\mathsf{SG}} t' : B$ and $A \sqsubseteq B$.
ii. Suppose $\cdot \vdash_{\mathsf{CG}} t : A$ and $\cdot \vdash t \sqsubseteq t'$. Then
  a. if $t \rightsquigarrow^* v$, then $t' \rightsquigarrow^* v'$ and $\cdot \vdash v \sqsubseteq v'$,
  b. if $t \uparrow$, then $t' \uparrow$,
  c. if $t' \rightsquigarrow^* v'$, then $t \rightsquigarrow^* v$ where $\cdot \vdash v \sqsubseteq v'$, or $t \rightsquigarrow^* \mathsf{error}_A$, and

Term Precision for Surface Grady:

$$\frac{}{t \sqsubseteq t}\,\text{refl} \qquad \frac{t_1 \sqsubseteq t_2}{(\mathsf{succ}\; t_1) \sqsubseteq (\mathsf{succ}\; t_2)}\,\mathsf{succ}$$

$$\frac{t_1 \sqsubseteq t_4 \quad t_2 \sqsubseteq t_5 \quad t_3 \sqsubseteq t_6}{(\mathsf{case}\; t_1 \;\mathsf{of}\; 0 \to t_2, (\mathsf{succ}\; x) \to t_3) \sqsubseteq (\mathsf{case}\; t_4 \;\mathsf{of}\; 0 \to t_5, (\mathsf{succ}\; x) \to t_6)}\,\mathsf{Nat}$$

$$\frac{t_1 \sqsubseteq t_3 \quad t_2 \sqsubseteq t_4}{(t_1, t_2) \sqsubseteq (t_3, t_4)}\,\times_i \qquad \frac{t_1 \sqsubseteq t_2}{(\mathsf{fst}\; t_1) \sqsubseteq (\mathsf{fst}\; t_2)}\,\times_{e_1} \qquad \frac{t_1 \sqsubseteq t_2}{(\mathsf{snd}\; t_1) \sqsubseteq (\mathsf{snd}\; t_2)}\,\times_{e_2}$$

$$\frac{t_1 \sqsubseteq t_3 \quad t_2 \sqsubseteq t_4}{(t_1 :: t_2) \sqsubseteq (t_3 :: t_4)}\,\mathsf{List}_i$$

$$\frac{t_1 \sqsubseteq t_4 \quad t_2 \sqsubseteq t_5 \quad t_3 \sqsubseteq t_6}{(\mathsf{case}\; t_1 \;\mathsf{of}\; [] \to t_2, (x :: y) \to t_3) \sqsubseteq (\mathsf{case}\; t_4 \;\mathsf{of}\; 0 \to t_5, (x :: y) \to t_6)}\,\mathsf{List}_e$$

$$\frac{t_1 \sqsubseteq t_2 \quad A_1 \sqsubseteq A_2}{(\lambda(x : A_1).t) \sqsubseteq (\lambda(x : A_2).t_2)}\,\to_i \qquad \frac{t_1 \sqsubseteq t_3 \quad t_2 \sqsubseteq t_4}{(t_1\; t_2) \sqsubseteq (t_3\; t_4)}\,\to_2$$

$$\frac{t_1 \sqsubseteq t_2}{(\Lambda(X <: A).t_1) \sqsubseteq (\Lambda(X <: A).t_2)}\,\forall_i \qquad \frac{t_1 \sqsubseteq t_2 \quad A \sqsubseteq B}{[A]t_1 \sqsubseteq [B]t_2}\,\forall_e$$

Term Precision for Core Grady:

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : ?}{\Gamma \vdash (\mathsf{unbox}_A\; t) \sqsubseteq t}\,\mathsf{box} \qquad \frac{\Gamma \vdash_{\mathsf{CG}} t : A}{\Gamma \vdash t \sqsubseteq (\mathsf{box}_A\; t)}\,\mathsf{unbox} \qquad \frac{\Gamma \vdash_{\mathsf{CG}} t : ?}{\Gamma \vdash (\mathsf{split}_S\; t) \sqsubseteq t}\,\mathsf{split}$$

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : S}{\Gamma \vdash t \sqsubseteq (\mathsf{squash}_S\; t)}\,\mathsf{squash} \qquad \frac{\Gamma \vdash_{\mathsf{CG}} t : B \quad A \sqsubseteq B}{\Gamma \vdash \mathsf{error}_A \sqsubseteq t}\,\mathsf{error}$$

Fig. 10. Term Precision

d. if $t' \uparrow$, then $t \uparrow$ or $t \rightsquigarrow^* \mathsf{error}_A$.

Proof. This result follows from the same proof as [16], and so, we only give a brief summary. Part i. holds by Lemma 8, and Part ii. follows from simulation of more precise programs (Lemma 9).

Part one states that one may insert casts into a closed gradual program, t, yielding a less precise program, t', and the program will remain typable, but at a less precise type. This part follows from the following generalization:

Lemma 8 (Gradual Guarantee Part One). *If $\Gamma \vdash_{\mathsf{SG}} t : A$, $t \sqsubseteq t'$, and $\Gamma \sqsubseteq \Gamma'$ then $\Gamma' \vdash_{\mathsf{SG}} t' : B$ and $A \sqsubseteq B$.*

Proof. This is a proof by induction on $\Gamma \vdash_{\mathsf{SG}} t : A$; see Appendix B.4 for the complete proof.

The previous result depends on context precision, denoted $\Gamma \sqsubseteq \Gamma'$, but we omit its definition, because it is the straightforward extension of type precision.

The remaining parts of the gradual guarantee follow from the next result.

Lemma 9 (Simulation of More Precise Programs). *Suppose $\Gamma \vdash_{\mathsf{CG}} t_1 : A$, $\Gamma \vdash t_1 \sqsubseteq t'_1$, $\Gamma \vdash_{\mathsf{CG}} t'_1 : A'$, and $t_1 \rightsquigarrow t_2$. Then $t'_1 \rightsquigarrow^* t'_2$ and $\Gamma \vdash t_2 \sqsubseteq t'_2$ for some t'_2.*

Proof. This proof holds by induction on $\Gamma \vdash_{\mathsf{CG}} t_1 : A_1$. See Appendix B.6 for the complete proof.

This result simply states that programs may become less precise by adding or removing casts, but they will behave in an expected manner.

The results given here are the main results in the proof of the gradual guarantee, but they depend on a number of auxiliary results. Unfortunately, they are too numerous to list here, but they are all given in Appedix A. Previous proofs of the gradual guarantee, e.g. [16] and [4], make heavy use of inversion for typing. However, as type systems become more complex inversion for typing is harder to obtain. Instead of using inversion for typing we proved inversion principles for the other judgments. This turned out to be a lot easier, because judgments like type and term precision, consistent subtyping, etc, have less complex definitions. All of the inversion principles we used can be found in Appedix A.

5 Conclusion

In this paper we extended our previous work [**?**] on a new foundation of gradual typing to a gradual type system with bounded quantification called Grady. We pointed out the existence of explicit casts in the surface language of gradual type systems that may be used to extend the expressiveness of the language. We then proved the gradual guarantee for Grady. Finally, we gave several interesting gradually typed bounded polymorphic programs in Grady's concrete implementation.

Future work. The simplistic nature of Core Grady suggests that we may extend Grady with even more features. We plan to study an extension of Core Grady with dependent types by first extending its categorical model given in our previous work [?], and then extending Grady itself. The idea we have in mind is to make the static fragment dependent, but the untype fragment remain as it is in Core Grady. This is akin to Krishnaswami et al.'s method of integrating linear and dependent types [7]. However, some interesting problems arise, for example, as we have seen here non-termination exists in gradually typed programs. We can type the Y combinator after all. Thus, a dependently type gradual type system would need to employ some method of tracking non-termination, and preventing non-terminating programs from running in proofs. This was exactly the task of the Trelly's project; see [?]. We believe that the method developed by Stephanie Weirich and her students [3] might be the solution for gradual typing. Combing gradual typing and dependent types would allow for the verification of gradually typed programs.

References

1. Abadi, M., Cardelli, L., Pierce, B., Plotkin, G.: Dynamic typing in a statically-typed language. In: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 213–227. POPL '89, ACM, New York, NY, USA (1989)
2. Ahmed, A., Findler, R.B., Siek, J.G., Wadler, P.: Blame for all. In: Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 201–214. POPL '11, ACM, New York, NY, USA (2011), `http://doi.acm.org/10.1145/1926385.1926409`
3. Casinghino, C., Sjöberg, V., Weirich, S.: Combining proofs and programs in a dependently typed language. In: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 33–45. POPL '14, ACM, New York, NY, USA (2014), `http://doi.acm.org/10.1145/2535838.2535883`
4. Garcia, R., Clark, A.M., Tanter, E.: Abstracting gradual typing. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 429–442. POPL '16, ACM, New York, NY, USA (2016)
5. Henglein, F.: Dynamic typing: syntax and proof theory. Science of Computer Programming 22(3), 197 – 230 (1994)
6. Henglein, F., Rehof, J.: Safe polymorphic type inference for a dynamically typed language: Translating scheme to ml. In: Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture. pp. 192–203. FPCA '95, ACM, New York, NY, USA (1995), `http://doi.acm.org/10.1145/224164.224203`
7. Krishnaswami, N.R., Pradic, P., Benton, N.: Integrating linear and dependent types. In: Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 17–30. POPL '15, ACM, New York, NY, USA (2015), `http://doi.acm.org/10.1145/2676726.2676969`
8. Lambek, J., Scott, P.: Introduction to Higher-Order Categorical Logic. Cambridge Studies in Advanced Mathematics, Cambridge University Press (1988)
9. Lambek, J.: From lambda calculus to cartesian closed categories. To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism pp. 376–402 (1980)

10. Matthews, J., Ahmed, A.: Parametric polymorphism through run-time sealing or, theorems for low, low prices! In: Proceedings of the Theory and Practice of Software, 17th European Conference on Programming Languages and Systems. pp. 16–31. ESOP'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008), `http://dl.acm.org/citation.cfm?id=1792878.1792881`
11. Pierce, B.C.: Types and Programming Languages. The MIT Press, 1st edn. (2002)
12. Rehof, J.: Polymorphic Dynamic Typing. masters thesis, DIKU, Department of Computer Science University of Copenhagen, Universitetsparken 1 DK-2100 Copenhagen Ø, Denmark (August 1995)
13. Scott, D.: Relating theories of the lambda-calculus. In: To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism (eds. Hindley and Seldin). pp. 403–450. Academic Press (1980)
14. Siek, J., Taha, W.: Gradual Typing for Objects, pp. 2–27. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
15. Siek, J.G., Taha, W.: Gradual typing for functional languages. In: Scheme and Functional Programming Workshop. 1, vol. 6, pp. 81–92 (2006)
16. Siek, J.G., Vitousek, M.M., Cimini, M., Boyland, J.T.: Refined Criteria for Gradual Typing. In: Ball, T., Bodik, R., Krishnamurthi, S., Lerner, B.S., Morrisett, G. (eds.) 1st Summit on Advances in Programming Languages (SNAPL 2015). Leibniz International Proceedings in Informatics (LIPIcs), vol. 32, pp. 274–293. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2015)

A Auxiliary Results with Proofs

Lemma 10 (Kinding).

i. If $\Gamma \vdash A \sim B$*, then* $\Gamma \vdash A : \star$ *and* $\Gamma \vdash B : \star$*.*
ii. If $\Gamma \vdash A \lesssim B$*, then* $\Gamma \vdash A : \star$ *and* $\Gamma \vdash B : \star$*.*
iii. If $\Gamma \vdash_{\mathsf{SG}} t : A$*, then* $\Gamma \vdash A : \star$*.*

Proof. This proof holds by straightforward induction the form of each assumed judgment.

Lemma 11 (Strengthening for Kinding). *If* $\Gamma, x : A \vdash B : \star$*, then* $\Gamma \vdash B : \star$*.*

Proof. This proof holds by straightforward induction on the form of $\Gamma, x : A \vdash B : \star$.

Lemma 12 (Inversion for Type Precision). *Suppose* $\Gamma \vdash A : \star$*,* $\Gamma \vdash B : \star$*, and* $A \sqsubseteq B$*. Then:*

i. if $A = ?$*, then* $\Gamma \vdash B \lesssim \mathbb{S}$*.*
ii. if $A = A_1 \to B_1$*, then* $B = ?$ *and* $\Gamma \vdash A \lesssim \mathbb{S}$*, or* $B = A_2 \to B_2$*,* $A_1 \sqsubseteq A_2$*, and* $B_1 \sqsubseteq B_2$*.*
iii. if $A = A_1 \times B_1$*, then* $B = ?$ *and* $\Gamma \vdash A \lesssim \mathbb{S}$*, or* $B = A_2 \times B_2$*,* $A_1 \sqsubseteq A_2$*, and* $B_1 \sqsubseteq B_2$*.*
iv. if $A = \mathsf{List}\, A_1$*, then* $B = ?$ *and* $\Gamma \vdash A \lesssim \mathbb{S}$*, or* $B = \mathsf{List}\, A_2$ *and* $A_1 \sqsubseteq A_2$*.*

v. if $A = \forall(X <: A_1).B_1$, then $B = \forall(X <: A_1).B_1$ and $B_1 \sqsubseteq B_2$.

Proof. This proof holds by straightforward induction on the form of $A \sqsubseteq B$.

Lemma 13 (Surface Grady Inversion for Term Precision). *Suppose $t \sqsubseteq t'$. Then:*

i. if $t = \mathsf{succ}\, t_1$, then $t' = \mathsf{succ}\, t_2$ and $t_1 \sqsubseteq t_2$.
ii. if $t = (\mathsf{case}\, t_1\, \mathsf{of}\, 0 \rightarrow t_2, (\mathsf{succ}\, x) \rightarrow t_3)$, then $t' = (\mathsf{case}\, t'_1\, \mathsf{of}\, 0 \rightarrow t'_2, (\mathsf{succ}\, x) \rightarrow t'_3)$, $t_1 \sqsubseteq t'_1$, $t_2 \sqsubseteq t'_2$, and $t_3 \sqsubseteq t'_3$.
iii. if $t = (t_1, t_2)$, then $t' = (t'_1, t'_2)$, $t_1 \sqsubseteq t'_1$, and $t_2 \sqsubseteq t'_2$.
iv. if $t = \mathsf{fst}\, t_1$, then $t' = \mathsf{fst}\, t'_1$ and $t_1 \sqsubseteq t'_1$.
v. if $t = \mathsf{snd}\, t_1$, then $t' = \mathsf{snd}\, t'_1$ and $t_1 \sqsubseteq t'_1$.
vi. if $t = t_1 :: t_2$, then $t' = t'_1 :: t'_2$, $t_1 \sqsubseteq t'_1$, and $t_2 \sqsubseteq t'_2$.
vii. if $t = (\mathsf{case}\, t_1\, \mathsf{of}\, [] \rightarrow t_2, (x :: y) \rightarrow t_3)$, then $t' = (\mathsf{case}\, t'_1\, \mathsf{of}\, [] \rightarrow t'_2, (x :: y) \rightarrow t'_3)$, $t_1 \sqsubseteq t'_1$, $t_2 \sqsubseteq t'_2$, and $t_3 \sqsubseteq t'_3$.
viii. if $t = \lambda(x : A_1).t_1$, then $t' = \lambda(x : A_1).t'_1$ and $t_1 \sqsubseteq t'_1$.
ix. if $t = (t_1\, t_2)$, then $t' = (t'_1\, t'_2)$, $t_1 \sqsubseteq t'_1$, and $t_2 \sqsubseteq t'_2$.
x. if $t = \Lambda(X <: A_1).t_1$, then $t' = \Lambda(X <: A_1).t'_1$ and $t_1 \sqsubseteq t'_1$.
xi. if $t = [A]t_1$, then $t' = [A]t'_1$ and $t_1 \sqsubseteq t'_1$.

Proof. This proof holds by straightforward induction on the form of $t \sqsubseteq t'$.

Lemma 14 (Inversion for Type Consistency). *Suppose $\Gamma \vdash A \sim B$. Then:*

i. if $A = ?$, then $\Gamma \vdash B \lesssim \mathbb{S}$.
ii. if $A = \mathsf{List}\, A'$, then $B = ?$ and $\Gamma \vdash A \lesssim \mathbb{S}$, or $B = \mathsf{List}\, B'$ and $\Gamma \vdash A' \sim B'$.
iii. if $A = A_1 \rightarrow B_1$, then $B = ?$ and $\Gamma \vdash A \lesssim \mathbb{S}$, or $B = A_2 \rightarrow B_2$, $\Gamma \vdash A_2 \sim A_1$, and $\Gamma \vdash B_1 \sim B_2$.
iv. if $A = A_1 \rightarrow B_1$, then $B = ?$ and $\Gamma \vdash A \lesssim \mathbb{S}$, or $B = A_2 \rightarrow B_2$, $\Gamma \vdash A_2 \sim A_1$, and $\Gamma \vdash B_1 \sim B_2$.
v. if $A = A_1 \times B_1$, then $B = ?$ and $\Gamma \vdash A \lesssim \mathbb{S}$, or $B = A_2 \times B_2$, $\Gamma \vdash A_1 \sim A_2$, and $\Gamma \vdash B_1 \sim B_2$.
vi. if $A = \forall(X <: A_1).B_1$, then $B = \forall(X <: A_1).B_2$ and $\Gamma, X <: A_1 \vdash B_1 \sim B_2$.

Proof. This proof holds by straightforward induction on the the form of $\Gamma \vdash A \sim B$.

Lemma 15 (Inversion for Consistent Subtyping). *Suppose $\Gamma \vdash A \lesssim B$. Then:*

i. if $A = ?$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ or $\Gamma \vdash B \lesssim \mathbb{S}$.
ii. if $A = X$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ and $\Gamma \vdash A : \star$, or $X <: B' \in \Gamma$ and $\Gamma \vdash B' \sim B$.
iii. if $A = \mathsf{Nat}$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ and $\Gamma \vdash A : \star$, or $B = \mathbb{S}$.
iv. if $A = \mathsf{Unit}$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ and $\Gamma \vdash A : \star$, or $B = \mathbb{S}$.
v. if $A = \mathsf{List}\, A_1$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ and $\Gamma \vdash A : \star$, $B = \mathbb{S}$ and $\Gamma \vdash A_1 \lesssim \mathbb{S}$, or $B = \mathsf{List}\, A'_1$ and $\Gamma \vdash A_1 \lesssim A'_1$.

vi. if $A = A_1 \to B_1$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ and $\Gamma \vdash A : \star$, $B = \mathbb{S}$, $\Gamma \vdash A_1 \lesssim \mathbb{S}$ and $\Gamma \vdash B_1 \lesssim \mathbb{S}$, or $B = A_1' \to B_1'$, $\Gamma \vdash A_1' \lesssim A_1$, and $\Gamma \vdash B_1 \lesssim B_1'$.
vii. if $A = A_1 \times B_1$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ and $\Gamma \vdash A : \star$, $B = \mathbb{S}$, $\Gamma \vdash A_1 \lesssim \mathbb{S}$ and $\Gamma \vdash B_1 \lesssim \mathbb{S}$, or $B = A_1' \times B_1'$, $\Gamma \vdash A_1 \lesssim A_1'$, and $\Gamma \vdash B_1 \lesssim B_1'$.
viii. if $A = \forall(X <: A_1).B_1$, then $B = A$ and $\Gamma \vdash A : \star$, $B = \top$ and $\Gamma \vdash A : \star$, or $B = \forall(X <: A_1).B_1'$ and $\Gamma, X <: A_1 \vdash B_1 \lesssim B_1'$.

Proof. This proof holds by straightforward induction on the the form of $\Gamma \vdash A \lesssim B$.

Lemma 16 (Symmetry for Type Consistency). *If $\Gamma \vdash A \sim B$, then $\Gamma \vdash B \sim A$.*

Proof. This holds by straightforward induction on the form of $\Gamma \vdash A \sim B$.

Lemma 17. *If $\Gamma \vdash A <: B$, then $\Gamma \vdash A \lesssim B$.*

Proof. This proof holds by straightforward induction on $\Gamma \vdash A <: B$.

Lemma 18. *if $\Gamma \vdash A \sim B$, then $\Gamma \vdash A \lesssim B$.*

Proof. By straightforward induction on $\Gamma \vdash A \sim B$.

Lemma 19 (Type Precision and Consistency). *Suppose $\Gamma \vdash A : \star$ and $\Gamma \vdash B : \star$. Then if $A \sqsubseteq B$, then $\Gamma \vdash A \sim B$.*

Proof. This proof holds by straightforward induction on $A \sqsubseteq B$.

Corollary 2 (Type Precision and Subtyping). *Suppose $\Gamma \vdash A : \star$ and $\Gamma \vdash B : \star$. Then if $A \sqsubseteq B$, then $\Gamma \vdash A \lesssim B$.*

Proof. This easily follows from the previous two lemmas.

Lemma 20. *Suppose $\Gamma \vdash A : \star$, $\Gamma \vdash B : \star$, and $\Gamma \vdash C : \star$. If $A \sqsubseteq B$ and $A \sqsubseteq C$, then $\Gamma \vdash B \sim C$.*

Proof. It must be the case that either $B \sqsubseteq C$ or $C \sqsubseteq B$, but in both cases we know $\Gamma \vdash B \sim C$ by Lemma 19.

Lemma 21 (Transitivity for Type Precision). *If $A \sqsubseteq B$ and $B \sqsubseteq C$, then $A \sqsubseteq C$.*

Proof. This proof holds by straightforward induction on $A \sqsubseteq B$ with a case analysis over $B \sqsubseteq C$.

Lemma 22. *If $\Gamma \vdash A \sim B$, then $A \sqsubseteq B$ or $B \sqsubseteq A$.*

Proof. This proof holds by straightforward induction over $\Gamma \vdash A \sim B$.

Lemma 23. *If $\Gamma \vdash A \lesssim B$ and $A \sqsubseteq A'$, then $B \sqsubseteq A'$ or $A' \sqsubseteq B$.*

Proof. Suppose $\Gamma \vdash A \lesssim B$ and $A \sqsubseteq A'$. The former implies that $A \sqsubseteq B$ or $B \sqsubseteq A$ by Lemma 6 and Lemma 22. At this point the result easily follows.

Lemma 24. *Suppose $A \sqsubseteq B$. Then*

i. If $\mathsf{nat}(A) = \mathsf{Nat}$, then $\mathsf{nat}(B) = \mathsf{Nat}$.
ii. If $\mathsf{list}(A) = \mathsf{List}\ C$, then $\mathsf{list}(B) = \mathsf{List}\ C'$ and $C \sqsubseteq C'$.
iii. If $\mathsf{fun}(A) = A_1 \to A_2$, then $\mathsf{fun}(B) = A'_1 \to A'_2$, $A_1 \sqsubseteq A'_1$, and $A_2 \sqsubseteq A'_2$.

Proof. This proof holds by straightforward induction on $A \sqsubseteq B$.

Lemma 25. *If $\Gamma \vdash A \sim B$, $\Gamma \vdash C : \star$, and $A \sqsubseteq C$, then $\Gamma \vdash C \sim B$.*

Proof. Suppose $\Gamma \vdash A \sim B$ and $A \sqsubseteq C$. Then we know that $A \sqsubseteq B$ or $B \sqsubseteq A$. If the former, then we know that $\Gamma \vdash C \sim B$. If the latter, then we obtain $B \sqsubseteq C$ by transitivity, and $\Gamma \vdash B \sim C$ which implies that $\Gamma \vdash C \sim B$ by symmetry.

Lemma 26. *If Γ' Ok, $\Gamma \sqsubseteq \Gamma'$ and $\Gamma \vdash A \sim B$, then $\Gamma' \vdash A \sim B$.*

Proof. This proof holds by straightforward induction on $\Gamma \vdash A \sim B$.

Lemma 27 (Subtyping Context Precision). *If $\Gamma \vdash A \lesssim B$ and $\Gamma \sqsubseteq \Gamma'$, then $\Gamma' \vdash A \lesssim B$.*

Proof. Context precision does not manipulate the bounds on type variables, and thus, with respect to subtyping Γ and Γ' are essentially equivalent.

Lemma 28 (Simply Typed Consistent Types are Subtypes of $\mathbb{S}$). *If $\Gamma \vdash A \lesssim \mathbb{S}$ and $\Gamma \vdash A \sim B$, then $\Gamma \vdash B \lesssim \mathbb{S}$.*

Proof. This holds by straightforward induction on the form of $\Gamma \vdash A \lesssim \mathbb{S}$.

Lemma 29 (Type Precision Preserves $\mathbb{S}$).

i. If $\Gamma \vdash B : \star$, $\Gamma \vdash A \lesssim \mathbb{S}$ and $A \sqsubseteq B$, then $\Gamma \vdash B \lesssim \mathbb{S}$.
ii. If $\Gamma \vdash A : \star$, $\Gamma \vdash B \lesssim \mathbb{S}$ and $A \sqsubseteq B$, then $\Gamma \vdash A \lesssim \mathbb{S}$.

Proof. Both cases follow by induction on the assumed consistent subtyping derivation.

Lemma 30 (Congruence of Type Consistency Along Type Precision).

i. If $A_1 \sqsubseteq A'_1$ and $\Gamma \vdash A_1 \sim A_2$ then $\Gamma \vdash A'_1 \sim A_2$.
ii. If $A_2 \sqsubseteq A'_2$ and $\Gamma \vdash A_1 \sim A_2$ then $\Gamma \vdash A_1 \sim A'_2$.

Proof. Both parts hold by induction on the assumed type consistency judgment. See Appendix B.2 for the complete proof.

Corollary 3 (Congruence of Type Consistency Along Type Precision Condensed). *If $A_1 \sqsubseteq A'_1$, $A_2 \sqsubseteq A'_2$, and $\Gamma \vdash A_1 \sim A_2$ then $\Gamma \vdash A'_1 \sim A'_2$.*

Lemma 31 (Congruence of Subtyping Along Type Precision). *Suppose $\Gamma \vdash B : \star$ and $A \sqsubseteq B$.*

i. If $\Gamma \vdash A \lesssim C$ then $\Gamma \vdash B \lesssim C$.
ii. If $\Gamma \vdash C \lesssim A$ then $\Gamma \vdash C \lesssim B$.

Proof. This is a proof by induction on the form of $A \sqsubseteq B$; see Appendix B.3 for the complete proof.

Corollary 4 (Congruence of Subtyping Along Type Precision). *If $A_1 \sqsubseteq A_2$, $B_1 \sqsubseteq B_2$, and $\Gamma \vdash A_1 \lesssim B_1$, then $\Gamma \vdash A_2 \lesssim B_2$.*

Lemma 32 (Typing Casting Morphisms). *If $\Gamma \vdash A \sim B$ and $\mathsf{caster}(A, B) = c$, then $\Gamma \vdash_{\mathsf{CG}} c : A \to B$.*

Proof. This proof holds similarly to how we constructed casting morphisms in the categorical model. See Lemma 2.

Lemma 33 (Substitution for Consistent Subtyping). *If $\Gamma, X <: B_1 \vdash B_2 \lesssim B_3$ and $\Gamma \vdash A_1 \lesssim B_1$, then $\Gamma \vdash [A_1/X]B_2 \lesssim [A_1/X]B_3$.*

Proof. This holds by straightforward induction on the form of $\Gamma, X <: B_1 \vdash B_2 \lesssim B_3$.

Lemma 34 (Substitution for Reflexive Type Consistency). *If $\Gamma, X <: B_1 \vdash B \sim B$, $\Gamma \vdash A_1 \sim A_2$, and $\Gamma \vdash A_2 <: B_1$, then $\Gamma \vdash [A_1/X]B \sim [A_2/X]B$.*

Proof. This holds by straightforward induction on the form of B.

Lemma 35 (Substitution for Type Consistency). *If $\Gamma, X <: B_1 \vdash B_2 \sim B_3$, $\Gamma \vdash A_1 \sim A_2$, and $\Gamma \vdash A_1 <: B_1$, then $\Gamma \vdash [A_1/X]B_2 \sim [A_2/X]B_3$.*

Proof. This holds by straightforward induction on $\Gamma, X <: B_1 \vdash B_2 \sim B_3$ using both substitution for consistent subtyping (Lemma 33) and substitution for reflexive type consistent (Lemma 34).

Lemma 36 (Typing for Type Precision). *If $\Gamma \vdash_{\mathsf{SG}} t_1 : A$, $t_1 \sqsubseteq t_2$, and $\Gamma \sqsubseteq \Gamma'$, then $\Gamma' \vdash_{\mathsf{SG}} t_2 : B$ and $A \sqsubseteq B$.*

Proof. This proof holds by induction on $\Gamma \vdash_{\mathsf{SG}} t_1 : A$ with a case analysis over $t_1 \sqsubseteq t_2$.

Lemma 37 (Substitution for Term Precision).

i. If $\Gamma, x : A \vdash t_1 \sqsubseteq t_2$ and $\Gamma \vdash t'_1 \sqsubseteq t'_2$, then $\Gamma \vdash [t'_1/x]t_1 \sqsubseteq [t'_2/x]t_2$.
ii. If $\Gamma, X <: A_2 \vdash t_1 \sqsubseteq t_2$ and $A_1 \sqsubseteq A'_1$, then $\Gamma \vdash [A_1/X]t_1 \sqsubseteq [A'_1/X]t_2$.

Proof. This proof of part one holds by straightforward induction on $\Gamma, x : A \vdash t_1 \sqsubseteq t_2$, and the proof of part two holds by straightforward induction on $\Gamma, X <: A_2 \vdash t_1 \sqsubseteq t_2$.

Lemma 38 (Typeability Inversion).

i. If $\Gamma \vdash_{\mathsf{CG}} \mathsf{succ}\, t : A$, then $\Gamma \vdash_{\mathsf{CG}} t : A'$ for some A'.
ii. If $\Gamma \vdash_{\mathsf{CG}} \mathsf{case}\, t \colon \mathsf{Nat}\, \mathsf{of}\, 0 \to t_1, (\mathsf{succ}\, x) \to t_2 : A$, then $\Gamma \vdash_{\mathsf{CG}} t : A_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : A_2$, and $\Gamma, x : \mathsf{Nat} \vdash_{\mathsf{CG}} t_2 : A_3$ for types A_1, A_2, A_3.
iii. If $\Gamma \vdash_{\mathsf{CG}} (t_1, t_2) : A$, then $\Gamma \vdash_{\mathsf{CG}} t_1 : A_1$ and $\Gamma \vdash_{\mathsf{CG}} t_2 : A_2$ for types A_1 and A_2.
iv. If $\Gamma \vdash_{\mathsf{CG}} \Lambda(X <: B).t : A$, then $\Gamma, X <: B \vdash_{\mathsf{CG}} t : A_1$ for some type A_1.
v. If $\Gamma \vdash_{\mathsf{CG}} [B]t : A$, then $\Gamma \vdash_{\mathsf{CG}} t : A_1$ for some type A_1.
vi. If $\Gamma \vdash_{\mathsf{CG}} \lambda(x : B).t : A$, then $\Gamma, x : B \vdash_{\mathsf{CG}} t : A_1$ for some type A_1.
vii. If $\Gamma \vdash_{\mathsf{CG}} t_1\, t_2 : A$, then $\Gamma \vdash_{\mathsf{CG}} t_1 : A_1$ and $\Gamma \vdash_{\mathsf{CG}} t_2 : A_2$ for types A_1 and A_2.
viii. If $\Gamma \vdash_{\mathsf{CG}} \mathsf{fst}\, t : A$, then $\Gamma \vdash_{\mathsf{CG}} t : A_1$ for some type A_1.
ix. If $\Gamma \vdash_{\mathsf{CG}} \mathsf{snd}\, t : A$, then $\Gamma \vdash_{\mathsf{CG}} t : A_1$ for some type A_1.
x. If $\Gamma \vdash_{\mathsf{CG}} t_1 :: t_2 : A$, then $\Gamma \vdash_{\mathsf{CG}} t_1 : A_1$ and $\Gamma \vdash_{\mathsf{CG}} t_2 : A_2$ for some types A_1 and A_2.
xi. If $\Gamma \vdash_{\mathsf{CG}} \mathsf{case}\, t \colon \mathsf{List}\, B\, \mathsf{of}\, [] \to t_1, (x :: y) \to t_2 : A$, then $\Gamma \vdash_{\mathsf{CG}} t : A_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : A_2$, and $\Gamma, x : A, y : \mathsf{List}\, A \vdash_{\mathsf{CG}} t_2 : A_3$ for types A_1, A_2, A_3.

Lemma 39 (Inversion for Term Precision for Core Grady). *Suppose $\Gamma \vdash t_1 \sqsubseteq t_2$.*

i. If $t_1 = x$, then one of the following is true:
  a. $t_2 = x$, $x : A \in \Gamma$, and Γ Ok
  b. $t_2 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
  c. $t_2 = \mathsf{squash}_K\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : K$
ii. If $t_1 = \mathsf{split}_{K_1}$, then one of the following is true:
  a. $t_2 = \mathsf{split}_{K_2}$ and $K_1 \sqsubseteq K_2$
  b. $t_2 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
  c. $t_2 = \mathsf{squash}_K\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : K$
iii. If $t_1 = \mathsf{squash}_{K_1}$, then one of the following is true:
  a. $t_2 = \mathsf{squash}_{K_2}$ and $K_1 \sqsubseteq K_2$
  b. $t_2 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
  c. $t_2 = \mathsf{squash}_K\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : K$
iv. If $t_1 = \mathsf{box}$, then one of the following is true:
  a. $t_2 = \mathsf{box}$
  b. $t_2 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
  c. $t_2 = \mathsf{squash}_K\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : K$
v. If $t_1 = \mathsf{unbox}$, then one of the following is true:
  a. $t_2 = \mathsf{unbox}$
  b. $t_2 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

vi. If $t_1 = 0$*, then one of the following is true:*

a. $t_2 = 0$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

vii. If $t_1 = \mathsf{triv}$*, then one of the following is true:*

a. $t_2 = \mathsf{triv}$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

viii. If $t_1 = []$*, then one of the following is true:*

a. $t_2 = []$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

ix. If $t_1 = \mathsf{succ}\, t'_1$*, then one of the following is true:*

a. $t_2 = \mathsf{succ}\, t'_2$ *and* $\Gamma \vdash t'_1 \sqsubseteq t'_2$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

x. If $t_1 = \mathsf{case}\, t'_1 : \mathsf{Nat}\, \mathsf{of}\, 0 \to t'_2, (\mathsf{succ}\, x) \to t'_3$*, then one of the following is true:*

a. $t_2 = \mathsf{case}\, t'_4 : \mathsf{Nat}\, \mathsf{of}\, 0 \to t'_5, (\mathsf{succ}\, x) \to t'_6$*,* $\Gamma \vdash t'_1 \sqsubseteq t'_4$*,* $\Gamma \vdash t'_2 \sqsubseteq t'_5$*, and* $\Gamma, x : \mathsf{Nat} \vdash t'_3 \sqsubseteq t'_6$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xi. If $t_1 = (t'_1, t'_2)$*, then one of the following is true:*

a. $t_2 = (t'_3, t'_4)$*,* $\Gamma \vdash t'_1 \sqsubseteq t'_3$*, and* $\Gamma \vdash t'_2 \sqsubseteq t'_4$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xii. If $t_1 = \mathsf{fst}\, t'_1$*, then one of the following is true:*

a. $t_2 = \mathsf{fst}\, t'_2$ *and* $\Gamma \vdash t'_1 \sqsubseteq t'_2$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xiii. If $t_1 = \mathsf{snd}\, t'_1$*, then one of the following is true:*

a. $t_2 = \mathsf{snd}\, t'_2$ *and* $\Gamma \vdash t'_1 \sqsubseteq t'_2$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xiv. If $t_1 = t'_1 :: t'_2$*, then one of the following is true:*

a. $t_2 = t'_3 :: t'_4$*,* $\Gamma \vdash t'_1 \sqsubseteq t'_3$*, and* $\Gamma \vdash t'_2 \sqsubseteq t'_4$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xv. If $t_1 = \mathsf{case}\, t'_1 : \mathsf{List}\, A_1\, \mathsf{of}\, [] \to t'_2, (x :: y) \to t'_3$*, then one of the following is true:*

a. $t_2 = \mathsf{case}\, t'_4 : \mathsf{List}\, A_2\, \mathsf{of}\, [] \to t'_5, (x :: y) \to t'_6$*,* $\Gamma \vdash t'_1 \sqsubseteq t'_4$*,* $\Gamma \vdash t'_2 \sqsubseteq t'_5$*, and* $\Gamma, x : A_2, y : \mathsf{List}\, A_2 \vdash t'_3 \sqsubseteq t'_6$*, and* $A_1 \sqsubseteq A_2$

b. $t_2 = \mathsf{box}_A\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$

c. $t_2 = \mathsf{squash}_K\, t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xvi. If $t_1 = \lambda(x : A_1).t_1$*, then one of the following is true:*

a. $t_2 = \lambda(x : A_2).t_2$ *and* $\Gamma, x : A_2 \vdash t_1 \sqsubseteq t_2$ *and* $A_1 \sqsubseteq A_2$
b. $t_2 = \mathsf{box}_A\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
c. $t_2 = \mathsf{squash}_K\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xvii. If $t_1 = t_1'\ t_2'$*, then one of the following is true:*
a. $t_2 = t_3'\ t_4'$*,* $\Gamma \vdash t_3 \sqsubseteq t_3'$*, and* $\Gamma \vdash t_4 \sqsubseteq t_4'$
b. $t_1' = \mathsf{unbox}_A$ *and* $t_2 = t_2'$
c. $t_1' = \mathsf{split}_K$ *and* $t_2 = t_2'$
d. $t_2 = \mathsf{box}_A\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
e. $t_2 = \mathsf{squash}_K\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xviii. If $t_1 = \mathsf{unbox}_A\ t_1'$*, then one of the following is true:*
a. $t_2 = t_1'$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1' : ?$
b. $t_2 = \mathsf{box}_A\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
c. $t_2 = \mathsf{squash}_K\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xix. If $t_1 = \mathsf{split}_K\ t_1'$*, then one of the following is true:*
a. $t_2 = t_1'$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1' : K$
b. $t_2 = \mathsf{box}_A\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
c. $t_2 = \mathsf{squash}_K\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xx. If $t_1 = \Lambda(X <: A).t_1'$*, then one of the following is true:*
a. $t_2 = \Lambda(X <: A).t_2'$ *and* $\Gamma, X <: A_2 \vdash t_1' \sqsubseteq t_2'$
b. $t_2 = \mathsf{box}_A\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
c. $t_2 = \mathsf{squash}_K\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xxi. If $t_1 = [A_1]t_1'$*, then one of the following is true:*
a. $t_2 = [A_2]t_2'$*,* $\Gamma \vdash t_1' \sqsubseteq t_2'$*, and* $A_1 \sqsubseteq A_2$
b. $t_2 = \mathsf{box}_A\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
c. $t_2 = \mathsf{squash}_K\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

xxii. If $t_1 = \mathsf{error}_{A_1}$*, then one of the following is true:*
a. $\Gamma \vdash_{\mathsf{CG}} t_2 : A_2$ *and* $A_1 \sqsubseteq A_2$
b. $t_2 = \mathsf{box}_A\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
c. $t_2 = \mathsf{squash}_K\ t_1$ *and* $\Gamma \vdash_{\mathsf{CG}} t_1 : K$

Proof. The proof of this result holds by straightforward induction on $\Gamma \vdash t_1 \sqsubseteq t_2$.

B Proofs

B.1 Proof of Left-to-Right Consistent Subtyping (Lemma 6)

This is a proof by induction on $\Gamma \vdash A \lesssim B$. We only show a few of the most interesting cases.

Case.

$$\frac{\Gamma \vdash A \lesssim \mathbb{S}}{\Gamma \vdash A \lesssim ?}\ \mathsf{box}$$

In this case $B = ?$.
Part i. Choose $A' = ?$.
Part ii. Choose $B' = A$.

Case.

$$\frac{\Gamma \vdash B \lesssim \mathbb{S}}{\Gamma \vdash ? \lesssim B}\ \mathsf{unbox}$$

In this case $A = ?$.
Part i. Choose $A' = B$.
Part ii. Choose $B' = ?$.
Case.

$$\frac{\Gamma \vdash A_2 \lesssim A_1 \quad \Gamma \vdash B_1 \lesssim B_2}{\Gamma \vdash (A_1 \to B_1) \lesssim (A_2 \to B_2)}\ \to$$

In this case $A = A_1 \to B_1$ and $B = A_2 \to B_2$.
Part i. By part two of the induction hypothesis we know that $\Gamma \vdash A_1' \sim A_1$ and $\Gamma \vdash A_2 <: A_1'$, and by part one of the induction hypothesis $\Gamma \vdash B_1 \sim B_1'$ and $\Gamma \vdash B_1' <: B_2$. By symmetry of type consistency we may conclude that $\Gamma \vdash A_1 \sim A_1'$ which along with $\Gamma \vdash B_1 \sim B_1'$ implies that $\Gamma \vdash (A_1 \to B_1) \sim (A_1' \to B_1')$, and by reapplying the rule we may conclude that $\Gamma \vdash (A_1' \to B_1') <: (A_2 \to B_2)$.
Part ii. Similar to part one, except that we first applying part one of the induction hypothesis to the first premise, and then the second part to the second premise.

B.2 Proof of Congruence of Type Consistency Along Type Precision (Lemma 30)

The proofs of both parts are similar, and so we only show a few cases of the first part, but the omitted cases follow similarly.
Proof of part one. This is a proof by induction on the form of $A_1 \sqsubseteq A_1'$.

Case.

$$\frac{\Gamma \vdash A_1 \lesssim \mathbb{S}}{A_1 \sqsubseteq ?}\ ?$$

In this case $A_1' = ?$. Suppose $\Gamma \vdash A_1 \sim A_2$. Then it suffices to show that $\Gamma \vdash ? \sim A_2$, and hence, we must show that $\Gamma \vdash A_2 \lesssim \mathbb{S}$, but this follows by Lemma 28.
Case.

$$\frac{A \sqsubseteq C \quad B \sqsubseteq D}{(A \to B) \sqsubseteq (C \to D)}\ \to$$

In this case $A_1 = A \to B$ and $A'_1 = C \to D$. Suppose $\Gamma \vdash A_1 \sim A_2$. Then by inversion for type consistency it must be the case that either $A_2 = ?$ and $\Gamma \vdash A_1 \lesssim \mathbb{S}$, or $A_2 = A' \to B'$, $\Gamma \vdash A \sim A'$, and $\Gamma \vdash B \sim B'$.
Consider the former. Then it suffices to show that $\Gamma \vdash A'_1 \sim ?$, and hence we must show that $\Gamma \vdash A'_1 \lesssim \mathbb{S}$, but this follows from Lemma 29.
Consider the case when $A_2 = A' \to B'$, $\Gamma \vdash A \sim A'$, and $\Gamma \vdash B \sim B'$. It suffices to show that $\Gamma \vdash (C \to D) \sim (A' \to B')$ which follows from $\Gamma \vdash A' \sim C$ and $\Gamma \vdash D \sim B'$. Thus, it suffices to show that latter. By assumption we know the following:

$$\begin{array}{c} A \sqsubseteq C \text{ and } \Gamma \vdash A \sim A' \\ B \sqsubseteq D \text{ and } \Gamma \vdash B \sim B' \end{array}$$

Now by two applications of the induction hypothesis we obtain $\Gamma \vdash C \sim A'$ and $\Gamma \vdash D \sim B'$. By symmetry the former implies $\Gamma \vdash A \sim C$ and we obtain our result.

B.3 Proof of Congruence of Subtyping Along Type Precision (Lemma 31)

This is a proof by induction on the form of $A \sqsubseteq B$. The proof of part two follows similarly to part one. We only give the most interesting cases. All others follow similarly.
Proof of part one. We only show the most interesting case, because all others are similar.

Case.

$$\frac{A_1 \sqsubseteq A_2 \quad B_1 \sqsubseteq B_2}{(A_1 \to B_1) \sqsubseteq (A_2 \to B_2)} \to$$

In this case $A = A_1 \to B_1$ and $B = A_2 \to B_2$. Suppose $\Gamma \vdash A \lesssim C$. Thus, by inversion for consistency subtyping it must be the case that $C = \top$ and $\Gamma \vdash A : \star$, $C = ?$ and $\Gamma \vdash A \lesssim \mathbb{S}$, or $C = A'_1 \to B'_1$, $\Gamma \vdash A'_1 \lesssim A_1$, and $\Gamma \vdash B_1 \lesssim B'_1$. The case when $C = \top$ is trivial, and the case when $C = ?$ is similarly to the proof of Lemma 30.
Consider the case when $C = A'_1 \to B'_1$, $\Gamma \vdash A'_1 \lesssim A_1$, and $\Gamma \vdash B_1 \lesssim B'_1$. By assumption we know the following:

$$\begin{array}{c} A_1 \sqsubseteq A_2 \text{ and } \Gamma \vdash A'_1 \lesssim A_1 \\ B_1 \sqsubseteq B_2 \text{ and } \Gamma \vdash B_1 \lesssim B'_1 \end{array}$$

So by part two and one, respectively, of the induction hypothesis we know that $\Gamma \vdash A'_1 \lesssim A_2$ and $\Gamma \vdash B_2 \lesssim B'_1$. Thus, by reapplying the rule above we may now conclude that $\Gamma \vdash (A_2 \to B_2) \lesssim (A'_1 \to B'_2)$ to obtain our result.

B.4 Proof of Gradual Guarantee Part One (Lemma 8)

This is a proof by induction on $\Gamma \vdash_{\mathsf{SG}} t : A$. We only show the most interesting cases, because the others follow similarly.

Case.

$$\frac{x : A \in \Gamma \quad \Gamma\,\mathrm{Ok}}{\Gamma \vdash_{\mathsf{SG}} x : A}\,\textsc{var}$$

In this case $t = x$. Suppose $t \sqsubseteq t'$. Then it must be the case that $t' = x$. If $x : A \in \Gamma$, then there is a type A' such that $x : A' \in \Gamma'$ and $A \sqsubseteq A'$. Thus, choose $B = A'$ and the result follows.
Case.

$$\frac{\Gamma \vdash_{\mathsf{SG}} t_1 : A' \quad \mathsf{nat}(A') = \mathsf{Nat}}{\Gamma \vdash_{\mathsf{SG}} \mathsf{succ}\, t_1 : \mathsf{Nat}}\,\mathsf{succ}$$

In this case $A = \mathsf{Nat}$ and $t = \mathsf{succ}\, t_1$. Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. Then by definition it must be the case that $t' = \mathsf{succ}\, t_2$ where $t_1 \sqsubseteq t_2$. By the induction hypothesis $\Gamma' \vdash_{\mathsf{SG}} t_2 : B'$ where $A' \sqsubseteq B'$. Since $\mathsf{nat}(A') = \mathsf{Nat}$ and $A' \sqsubseteq B'$, then it must be the case that $\mathsf{nat}(B') = \mathsf{Nat}$ by Lemma 24. At this point we obtain our result by choosing $B = \mathsf{Nat}$, and reapplying the rule above.
Case.

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{SG}} t_1 : C \quad \mathsf{nat}(C) = \mathsf{Nat} \quad \Gamma \vdash A_1 \sim A \\ \Gamma \vdash_{\mathsf{SG}} t_2 : A_1 \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathsf{SG}} t_3 : A_2 \quad \Gamma \vdash A_2 \sim A\end{array}}{\Gamma \vdash_{\mathsf{SG}} \mathsf{case}\, t_1\, \mathsf{of}\, 0 \to t_2, (\mathsf{succ}\, x) \to t_3 : A}\,\mathsf{Nat}_e$$

In this case $t = \mathsf{case}\, t_1\, \mathsf{of}\, 0 \to t_2, (\mathsf{succ}\, x) \to t_3$. Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. This implies that $t' = \mathsf{case}\, t'_1\, \mathsf{of}\, 0 \to t'_2, (\mathsf{succ}\, x) \to t'_3$ such that $t_1 \sqsubseteq t'_1$, $t_2 \sqsubseteq t'_2$, and $t_3 \sqsubseteq t'_3$. Since $\Gamma \sqsubseteq \Gamma'$ then $(\Gamma, x : \mathsf{Nat}) \sqsubseteq (\Gamma', x : \mathsf{Nat})$. By the induction hypothesis we know the following:

$$\begin{array}{l}\Gamma' \vdash_{\mathsf{SG}} t'_1 : C' \text{ for } C \sqsubseteq C' \\ \Gamma' \vdash_{\mathsf{SG}} t_2 : A'_1 \text{ for } A_1 \sqsubseteq A'_1 \\ \Gamma', x : \mathsf{Nat} \vdash_{\mathsf{SG}} t_3 : A'_2 \text{ for } A_2 \sqsubseteq A'_2\end{array}$$

By assumption we know that $\Gamma \vdash A_1 \sim A$, $\Gamma \vdash A_2 \sim A$, and $\Gamma \sqsubseteq \Gamma'$, hence, by Lemma 26 we know $\Gamma' \vdash A_1 \sim A$ and $\Gamma' \vdash A_2 \sim A$. By the induction hypothesis we know that $A_1 \sqsubseteq A'_1$ and $A_2 \sqsubseteq A'_2$, so by using Lemma 25 we may obtain that $\Gamma' \vdash A'_1 \sim A$ and $\Gamma' \vdash A'_2 \sim A$. At this point choose $B = A$ and we obtain our result by reapplying the rule.
Case.

$$\frac{\Gamma \vdash_{\mathsf{SG}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{SG}} t_2 : A_2 \quad \mathsf{list}(A_2) = \mathsf{List}\, A_3 \quad \Gamma \vdash A_1 \sim A_3}{\Gamma \vdash_{\mathsf{SG}} t_1 :: t_2 : \mathsf{List}\, A_3}\ \mathsf{List}_i$$

In this case $A = \mathsf{List}\, A_3$ and $t = t_1 :: t_2$. Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. Then it must be the case that $t' = t'_1 :: t'_2$ where $t_1 \sqsubseteq t'_1$ and $t_2 \sqsubseteq t'_2$. Then by the induction hypothesis we know the following:

$$\Gamma' \vdash_{\mathsf{SG}} t'_1 : A'_1 \text{ where } A_1 \sqsubseteq A'_1$$
$$\Gamma' \vdash_{\mathsf{SG}} t'_2 : A'_2 \text{ where } A_2 \sqsubseteq A'_2$$

By Lemma 24 $\mathsf{list}(A'_2) = \mathsf{List}\, A'_3$ where $A_3 \sqsubseteq A'_3$. Now by Lemma 26 and Lemma 25 we know that $\Gamma' \vdash A'_1 \sim A_3$, and by using the same lemma again, $\Gamma' \vdash A'_1 \sim A'_3$ because $\Gamma' \vdash A_3 \sim A'_1$ holds by symmetry. Choose $B = \mathsf{List}\, A'_3$ and the result follows.
Case.

$$\frac{\Gamma \vdash_{\mathsf{SG}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{SG}} t_2 : A_2}{\Gamma \vdash_{\mathsf{SG}} (t_1, t_2) : A_1 \times A_2}\ \times_i$$

In this case $A = A_1 \times A_2$ and $t = (t_1, t_2)$. Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. This implies that $t' = (t'_1, t'_2)$ where $t_1 \sqsubseteq t'_1$ and $t_2 \sqsubseteq t'_2$.
By the induction hypothesis we know:

$$\Gamma' \vdash_{\mathsf{SG}} t'_1 : A'_1 \text{ and } A_1 \sqsubseteq A'_1$$
$$\Gamma' \vdash_{\mathsf{SG}} t'_2 : A'_2 \text{ and } A_2 \sqsubseteq A'_2$$

Then choose $B = A'_1 \times A'_2$ and the result follows by reapplying the rule above and the fact that $(A_1 \times A_2) \sqsubseteq (A'_1 \times A'_2)$.
Case.

$$\frac{\Gamma, x : A_1 \vdash_{\mathsf{SG}} t_1 : B_1}{\Gamma \vdash_{\mathsf{SG}} \lambda(x : A_1).t_1 : A_1 \to B_1}\ \to_i$$

In this case $A_1 \to B_2$ and $t = \lambda(x : A_1).t_1$. Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. Then it must be the case that $t' = \lambda(x : A_2).t_2$, $t_1 \sqsubseteq t_2$, and $A_1 \sqsubseteq A_2$. Since $\Gamma \sqsubseteq \Gamma'$ and $A_1 \sqsubseteq A_2$, then $(\Gamma, x : A_1) \sqsubseteq (\Gamma', x : A_2)$ by definition. Thus, by the induction hypothesis we know the following:

$$\Gamma', x : A_2 \vdash_{\mathsf{SG}} t'_1 : B_2 \text{ and } B_1 \sqsubseteq B_2$$

Choose $B = A_2 \to B_2$ and the result follows by reapplying the rule above and the fact that $(A_1 \to B_1) \sqsubseteq (A_2 \to B_2)$.
Case.

$$\frac{\Gamma \vdash_{\mathsf{SG}} t_1 : \forall (X <: C_0).C_2 \quad \Gamma \vdash C_1 \lesssim C_0}{\Gamma \vdash_{\mathsf{SG}} [C_1]t_1 : [C_1/X]C_2}\ \forall_e$$

In this case $t = [C_1]t_1$. Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. Then it must be the case that $t' = [C_1']t_2$ such that $t_1 \sqsubseteq t_2$ and $C_1 \sqsubseteq C_1'$. By the induction hypothesis:

$$\Gamma' \vdash_{\mathsf{SG}} t_2 : C \text{ where } \forall(X <: C_0).C_2 \sqsubseteq C$$

Thus, it must be the case that $C = \forall(X <: C_0).C_2'$ such that $C_2 \sqsubseteq C_2'$. By assumption we know that $\Gamma \vdash C_1 \lesssim C_0$ and $C_1 \sqsubseteq C_1'$, and thus, by Corollary 4 and Lemma 27 we know $\Gamma' \vdash C_1' \lesssim C_0$. Thus, choose $B = C$, and the result follows by reapplying the rule above, and the fact that $A \sqsubseteq C$, because $C_2 \sqsubseteq C_2'$.
Case.

$$\frac{\Gamma \vdash_{\mathsf{SG}} t : A' \quad \Gamma \vdash A' \lesssim A}{\Gamma \vdash_{\mathsf{SG}} t : A}\ \text{SUB}$$

Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. By the induction hypothesis we know that $\Gamma' \vdash_{\mathsf{SG}} t' : A''$ for $A' \sqsubseteq A''$. We know $A'' \sqsubseteq A$ or $A \sqsubseteq A''$, because we know that $\Gamma \vdash A' \lesssim A$ and $A' \sqsubseteq A''$. Suppose $A'' \sqsubseteq A$, then by Corollary 2 $\Gamma' \vdash A'' \lesssim A$, and then by subsumption $\Gamma' \vdash_{\mathsf{SG}} t' : A$, hence, choose $B = A$ and the result follows. If $A \sqsubseteq A''$, then choose $B = A''$ and the result follows.
Case.

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{SG}} t_1 : C \quad \mathsf{fun}(C) = A_1 \to B_1 \\ \Gamma \vdash_{\mathsf{SG}} t_2 : A_2 \quad \Gamma \vdash A_2 \sim A_1\end{array}}{\Gamma \vdash_{\mathsf{SG}} t_1\, t_2 : B_1}\ \to_e$$

In this case $A = B_1$ and $t = t_1\, t_2$. Suppose $t \sqsubseteq t'$ and $\Gamma \sqsubseteq \Gamma'$. The former implies that $t' = t_1'\, t_2'$ such that $t_1 \sqsubseteq t_1'$ and $t_2 \sqsubseteq t_2'$. By the induction hypothesis we know the following:

$$\begin{array}{c}\Gamma' \vdash_{\mathsf{SG}} t_1' : C' \text{ for } C \sqsubseteq C' \\ \Gamma' \vdash_{\mathsf{SG}} t_2' : A_2' \text{ for } A_2 \sqsubseteq A_2'\end{array}$$

We know by assumption that $\Gamma \vdash A_2 \sim A_1$ and hence $\Gamma' \vdash A_2 \sim A_1$ because bounds on type variables are left unchanged by context precision. Since $C \sqsubseteq C'$ and $\mathsf{fun}(C) = A_1 \to B_1$, then $\mathsf{fun}(C') = A_1' \to B_1'$ where $A_1 \sqsubseteq A_1'$ and $B_1 \sqsubseteq B_1'$ by Lemma 24. Furthermore, we know $\Gamma' \vdash A_2 \sim A_1$ and $A_2 \sqsubseteq A_2'$ and $A_1 \sqsubseteq A_1'$, then we know $\Gamma' \vdash A_2' \sim A_1'$ by Corollary 3. So choose $B = B_1'$. Then reapply the rule above and the result follows, because $B_1 \sqsubseteq B_1'$.

B.5 Proof of Type Preservation for Cast Insertion (Lemma 7)

The cast insertion algorithm is type directed and with respect to every term t_1 it will produce a term t_2 of the core language with the type A – this is straightforward to show by induction on the form of $\Gamma \vdash_{\mathsf{SG}} t_1 : A$ making use of typing for casting morphisms Lemma 32 – except in the case of type application. We only consider this case here.

This is a proof by induction on the form of $\Gamma \vdash_{\mathsf{SG}} t_1 : A$. Suppose the form of $\Gamma \vdash_{\mathsf{SG}} t_1 : A$ is as follows:

$$\frac{\Gamma \vdash_{\mathsf{SG}} t_1' : \forall(X <: B_1).B_2 \quad \Gamma \vdash A_1 \lesssim B_1}{\Gamma \vdash_{\mathsf{SG}} [A_1]t_1' : [A_1/X]B_2}\ \forall_e$$

In this case $t_1 = [A_1]t_1'$ and $A = [A_1/X]B_2$. Cast insertion is syntax directed, and hence, inversion for it holds trivially. Thus, it must be the case that the form of $\Gamma \vdash t_1 \Rightarrow t_2 : B$ is as follows:

$$\frac{\Gamma \vdash t_1' \Rightarrow t_2' : \forall(X <: B_1).B_2' \quad \Gamma \vdash A_1 \sim A_2 \quad \Gamma \vdash A_2 <: B_1}{\Gamma \vdash ([A_1]t_1') \Rightarrow ([A_2]t_2') : [A_2/X]B_2'}$$

So $t_2 = [A_2]t_2'$ and $B = [A_2/X]B_2'$. Since we know $\Gamma \vdash_{\mathsf{SG}} t_1' : \forall(X <: B_1).B_2$ and $\Gamma \vdash t_1' \Rightarrow t_2' : \forall(X <: B_1).B_2'$ we can apply the induction hypothesis to obtain $\Gamma \vdash_{\mathsf{CG}} t_2' : \forall(X <: B_1).B_2'$ and $\Gamma \vdash (\forall(X <: B_1).B_2) \sim (\forall(X <: B_1).B_2')$, and thus, $\Gamma, X <: B_1 \vdash B_2 \sim B_2'$ by inversion for type consistency. If $\Gamma, X <: B_1 \vdash B_2 \sim B_2'$ holds, then $\Gamma \vdash [A_1/X]B_2 \sim [A_2/X]B_2'$ when $\Gamma \vdash A_1 \sim A_2$ by substitution for type consistency (Lemma 35). Since we know $\Gamma \vdash_{\mathsf{CG}} t_2' : \forall(X <: B_1).B_2'$ by the induction hypothesis and $\Gamma \vdash A_2 <: B_1$ by assumption, then we know $\Gamma \vdash_{\mathsf{CG}} [A_2]t_2' : [A_2/X]B_2'$ by applying the Core Grady typing rule $\forall_e$.

B.6 Proof of Simulation of More Precise Programs (Lemma 9)

This is a proof by induction on $\Gamma \vdash_{\mathsf{CG}} t_1 : A_1$. We only give the most interesting cases. All others follow similarly. Throughout the proof we implicitly make use of typability inversion (Lemma 38) when applying the induction hypothesis.

Case.

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : \mathsf{Nat}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{succ}\, t : \mathsf{Nat}}\ \mathsf{succ}$$

In this case $t_1 = \mathsf{succ}\, t$ and $A = \mathsf{Nat}$. Suppose $\Gamma \vdash_{\mathsf{CG}} t_1' : A'$. By inversion for term precision we must consider the following cases:
i. $t_1' = \mathsf{succ}\, t'$ and $\Gamma \vdash t \sqsubseteq t'$
ii. $t_1' = \mathsf{box}_{\mathsf{Nat}}\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : \mathsf{Nat}$

Proof of part i. Suppose $t_1' = \mathsf{succ}\, t'$, $\Gamma \vdash t \sqsubseteq t'$, and $t_1 \rightsquigarrow t_2$. Then $t_2 = \mathsf{succ}\, t''$ and $t \rightsquigarrow t''$. Then by the induction hypothesis we know that there is some t''' such that $t' \rightsquigarrow^* t'''$ and $\Gamma \vdash t'' \sqsubseteq t'''$. Choose $t_2' = \mathsf{succ}\, t'''$ and the result follows.

Proof of part ii. Suppose $t_1' = \mathsf{box}_{\mathsf{Nat}}\, t_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : \mathsf{Nat}$, and $t_1 \rightsquigarrow t_2$. Then choose $t_2' = \mathsf{box}_{\mathsf{Nat}}\, t_2$, and the result follows, because we know by type preservation that $\Gamma \vdash_{\mathsf{CG}} t_2 : \mathsf{Nat}$, and hence, $\Gamma \vdash t_2 \sqsubseteq t_2'$.
Case.

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{CG}} t : \mathsf{Nat} \\ \Gamma \vdash_{\mathsf{CG}} t_3 : A \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathsf{CG}} t_4 : A\end{array}}{\Gamma \vdash_{\mathsf{CG}} \mathsf{case}\, t \colon \mathsf{Nat}\, \mathsf{of}\, 0 \to t_3, (\mathsf{succ}\, x) \to t_4 : A}\ \mathsf{Nat}_e$$

In this case $t_1 = \mathsf{case}\, t \colon \mathsf{Nat}\, \mathsf{of}\, 0 \to t_3, (\mathsf{succ}\, x) \to t_4$. Suppose $\Gamma \vdash_{\mathsf{CG}} t'_1 : A'$. Then inversion of term precision implies that one of the following must hold:

- $t'_1 = \mathsf{case}\, t' \colon \mathsf{Nat}\, \mathsf{of}\, 0 \to t'_3, (\mathsf{succ}\, x) \to t'_4$, $\Gamma \vdash t \sqsubseteq t'$, $\Gamma \vdash t_3 \sqsubseteq t'_3$, and $\Gamma, x : \mathsf{Nat} \vdash t_4 \sqsubseteq t'_4$
- $t'_1 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
- $t'_1 = \mathsf{squash}_K\, t_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : K$, and $A = K$

Proof of part i. Suppose $t'_1 = \mathsf{case}\, t' \colon \mathsf{Nat}\, \mathsf{of}\, 0 \to t'_3, (\mathsf{succ}\, x) \to t'_4$, $\Gamma \vdash t \sqsubseteq t'$, $\Gamma \vdash t_3 \sqsubseteq t'_3$, and $\Gamma, x : \mathsf{Nat} \vdash t_4 \sqsubseteq t'_4$.

We case split over $t_1 \rightsquigarrow t_2$.

Case. Suppose $t = 0$ and $t_2 = t_3$. Since $\Gamma \vdash t_1 \sqsubseteq t'_1$ we know that it must be the case that $t' = 0$ and $t'_1 \rightsquigarrow t'_3$ by inversion for term precision or t'_1 would not be typable which is a contradiction. Thus, choose $t'_2 = t'_3$ and the result follows.

Case. Suppose $t = \mathsf{succ}\, t''$ and $t_2 = [t''/x]t_4$. Since $\Gamma \vdash t_1 \sqsubseteq t'_1$ we know that $t' = \mathsf{succ}\, t'''$, or t'_1 would not be typable, and $\Gamma \vdash t'' \sqsubseteq t'''$ by inversion for term precision. In addition, $t'_1 \rightsquigarrow [t'''/x]t'_4$. Choose $t_2 = [t'''/x]t'_4$. Then it suffices to show that $\Gamma \vdash [t''/x]t_4 \sqsubseteq [t'''/x]t'_4$ by substitution for term precision (Lemma 37).

Case. Suppose a congruence rule was used. Then $t_2 = \mathsf{case}\, t'' \colon \mathsf{Nat}\, \mathsf{of}\, 0 \to t''_3, (\mathsf{succ}\, x) \to t''_4$. This case will follow straightforwardly by induction and a case split over which congruence rule was used.

Proof of part ii. Suppose $t'_1 = \mathsf{box}_A\, t_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : A$, and $t_1 \rightsquigarrow t_2$. Then choose $t'_2 = \mathsf{box}_A\, t_2$, and the result follows, because we know by type preservation that $\Gamma \vdash_{\mathsf{CG}} t_2 : A$, and hence, $\Gamma \vdash t_2 \sqsubseteq t'_2$.

Proof of part iii. Similar to the previous case.
Case.

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : A \times B}{\Gamma \vdash_{\mathsf{CG}} \mathsf{fst}\, t : A}\ \times_{e_1}$$

In this case $t_1 = \mathsf{fst}\, t$. Suppose $\Gamma \vdash t_1 \sqsubseteq t'_1$ and $\Gamma \vdash_{\mathsf{CG}} t'_1 : A'$. Then inversion for term precision implies that one of the following must hold:

- $t'_1 = \mathsf{fst}\, t'$ and $\Gamma \vdash t \sqsubseteq t'$
- $t'_1 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
- $t'_1 = \mathsf{squash}_K\, t_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : K$, and $A = K$

We only consider the proof of part i, because the others follow similarly to the previous case. Case split over $t_1 \rightsquigarrow t_2$.

Case. Suppose $t = (t'_3, t''_3)$ and $t_2 = t'_3$. By inversion for term precision it must be the case that $t' = (t'_4, t''_4)$ because $\Gamma \vdash t_1 \sqsubseteq t'_1$ or else t'_1 would not be typable. In addition, this implies that $\Gamma \vdash t'_3 \sqsubseteq t'_4$ and $\Gamma \vdash t''_3 \sqsubseteq t''_4$. Thus, $t'_1 \rightsquigarrow t'_4$. Thus, choose $t'_2 = t'_4$ and the result follows.

Case. Suppose a congruence rule was used. Then $t_2 = \mathsf{fst}\, t''$. This case will follow straightforwardly by induction and a case split over which congruence rule was used.

Case.

$$\frac{\Gamma, x : A_1 \vdash_{\mathsf{CG}} t : A_2}{\Gamma \vdash_{\mathsf{CG}} \lambda(x : A_1).t : A_1 \to A_2} \to_i$$

In this case $t_1 = \lambda(x : A_1).t$ and $A = A_1 \to A_2$. Suppose $\Gamma \vdash t_1 \sqsubseteq t'_1$ and $\Gamma \vdash_{\mathsf{CG}} t'_1 : A'$. Then inversion of term precision implies that one of the following must hold:

- $t'_1 = \lambda(x : A'_1).t'$
- $t'_1 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
- $t'_1 = \mathsf{squash}_K\, t_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : K$, and $A = K$

We only consider the proof of part i. The reduction relation does not reduce under λ-expressions. Hence, $t_2 = t_1$, and thus, choose $t'_2 = t'_1$, and the case trivially follows.

Case.

$$\frac{\Gamma \vdash_{\mathsf{CG}} t_3 : A_1 \to A_2 \quad \Gamma \vdash_{\mathsf{CG}} t_4 : A_1}{\Gamma \vdash_{\mathsf{CG}} t_3\, t_4 : A_2} \to_e$$

In this case $t_1 = t_3\, t_4$. Suppose $\Gamma \vdash t_1 \sqsubseteq t'_1$ and $\Gamma \vdash_{\mathsf{CG}} t'_1 : A'$. Then by inversion for term prevision we know one of the following is true:

i. $t'_1 = t'_3\, t'_4$, $\Gamma \vdash t_3 \sqsubseteq t'_3$, and $\Gamma \vdash t_4 \sqsubseteq t'_4$
ii. $t'_1 = \mathsf{box}_{A_2}\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
iii. $t_3 = \mathsf{unbox}_{A_2}$, $t'_1 = t_4$, and $\Gamma \vdash_{\mathsf{CG}} t_4 : ?$
iv. $t_3 = \mathsf{split}_{K_2}$, $t'_1 = t_4$, and $\Gamma \vdash_{\mathsf{CG}} t_4 : ?$
v. $t'_1 = \mathsf{squash}_{K_2}\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : K_2$

Proof of part i. Suppose $t'_1 = t'_3\, t'_4$, $\Gamma \vdash t_3 \sqsubseteq t'_3$, and $\Gamma \vdash t_4 \sqsubseteq t'_4$.
We case split on the from of $t_1 \rightsquigarrow t_2$.

Case. Suppose $t_3 = \lambda(x : A_1).t_5$ and $t_2 = [t_4/x]t_5$. Then by inversion for term precision we know that $t'_3 = \lambda(x : A'_1).t'_5$ and $\Gamma, x : A'_2 \vdash t_5 \sqsubseteq t'_5$, because $\Gamma \vdash t_3 \sqsubseteq t'_3$ and the requirement that t'_1 is typable. Choose $t'_2 = [t'_4/x]t'_5$ and it is easy to see that $t'_1 \rightsquigarrow [t'_4/x]t'_4$. We know that $\Gamma, x : A'_2 \vdash t_5 \sqsubseteq t'_5$ and $\Gamma \vdash t_4 \sqsubseteq t'_4$, and hence, by Lemma 37 we know that $\Gamma \vdash [t_4/x]t_5 \sqsubseteq [t'_4/x]t'_5$, and we obtain our result.

Case. Suppose $t_3 = \mathsf{unbox}_A$, $t_4 = \mathsf{box}_A\, t_5$, and $t_2 = t_5$. Then by inversion for term prevision $t'_3 = \mathsf{unbox}_A$, $t'_4 = \mathsf{box}_A\, t'_5$, and $\Gamma \vdash t_5 \sqsubseteq t'_5$. Note that $t'_4 = \mathsf{box}_A\, t'_5$ and $\Gamma \vdash t_5 \sqsubseteq t'_5$ hold even though there are two potential rules that could have been used to construct $\Gamma \vdash t_4 \sqsubseteq t'_4$. Choose $t'_2 = t'_5$ and it is easy to see that $t'_1 \rightsquigarrow t'_5$. Thus, we obtain our result.

Case. Suppose $t_3 = \mathsf{unbox}_A$, $t_4 = \mathsf{box}_B\, t_5$, $A \neq B$, and $t_2 = \mathsf{error}_B$. Then $t'_3 = \mathsf{unbox}_A$ and $t'_4 = \mathsf{box}_B\, t'_5$. Choose $t'_2 = \mathsf{error}_B$ and it is easy to see that $t'_1 \rightsquigarrow t'_5$. Finally, we can see that $\Gamma \vdash t_2 \sqsubseteq t'_2$ by reflexivity.

Case. Suppose $t_3 = \mathsf{split}_U$, $t_4 = \mathsf{squash}_U\, t_5$, and $t_2 = t_5$. Similar to the case for boxing and unboxing.

Case. Suppose $t_3 = \mathsf{split}_{U_1}$, $t_4 = \mathsf{squash}_{U_2}\, t_5$, $U_1 \neq U_2$, and $t_2 = t_5$. Similar to the case for boxing and unboxing.

Case. Suppose a congruence rule was used. Then $t_2 = t'_5\, t'_6$. This case will follow straightforwardly by induction and a case split over which congruence rule was used.

Proof of part ii. We know that $t_1 = t_3\, t_4$. Suppose $t'_1 = \mathsf{box}_{A_2}\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$. If $t_1 \rightsquigarrow t_2$, then $t'_1 = (\mathsf{box}_{A_2}\, t_1) \rightsquigarrow (\mathsf{box}_{A_2}\, t_2)$. Thus, choose $t'_2 = \mathsf{box}_{A_2}\, t_2$.

Proof of part iii. We know that $t_1 = t_3\, t_4$. Suppose $t_3 = \mathsf{unbox}_{A_2}$, $t'_1 = t_4$, and $\Gamma \vdash_{\mathsf{CG}} t_4 : ?$. Then $t_1 = \mathsf{unbox}_{A_2}\, t_4$. We case split over $t_1 \rightsquigarrow t_2$. We have three cases to consider.

Suppose $t_4 = \mathsf{box}_{A_2}\, t_5$ and $t_2 = t_5$. Then choose $t'_2 = t_4 = t'_1$, and we obtain our result.

Suppose $t_4 = \mathsf{box}_{A_3}\, t_5$, $A_2 \neq A_3$, and $t_2 = \mathsf{error}_{A_2}$. Then choose $t'_2 = t_4 = t'_1$, and we obtain our result.

Suppose a congruence rule was used. Then $t_2 = t_3\, t'_4$. This case will follow straightforwardly by induction.

Proof of part iv. Similar to part iii.

Proof of part v. Similar to part ii.
Case.

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : \forall(X <: A_2).A_3 \quad \Gamma \vdash A_1 <: A_2}{\Gamma \vdash_{\mathsf{CG}} [A_1]t : [A_1/X]A_3}\ \forall_e$$

In this case $t_1 = [A_1]t$ and $A = [A_1/X]A_3$. Suppose $\Gamma \vdash t_1 \sqsubseteq t'_1$ and $\Gamma \vdash_{\mathsf{CG}} t'_1 : A'$.

- $t'_1 = [A'_1]t'$, $\Gamma \vdash t \sqsubseteq t'$, and $A_1 \sqsubseteq A'_1$
- $t'_1 = \mathsf{box}_A\, t_1$ and $\Gamma \vdash_{\mathsf{CG}} t_1 : A$
- $t'_1 = \mathsf{squash}_K\, t_1$, $\Gamma \vdash_{\mathsf{CG}} t_1 : K$, and $A = K$

We only consider the proof of part i. We case split over the form of $t_1 \rightsquigarrow t_2$.

Case. Suppose $t = \Lambda(X <: A_2).t_3$ and $t_2 = [A_1/X]t_3$. Then inversion for term precision on $\Gamma \vdash t \sqsubseteq t'$ and the fact that $\Gamma \vdash_{\mathsf{CG}} t : \forall(X < : A_2).A_3$ and $t'_1 = [A'_1]t'$ then it can only be the case that $t' = \Lambda(X < : A_2).t'_3$ and $\Gamma, X <: A_2 \vdash t_3 \sqsubseteq t'_3$, or t'_1 would not be typable which is a contradiction. Then by substitution for term precision we know that $\Gamma \vdash [A_1/X]t_3 \sqsubseteq [A'_1/X]t'_3$ by substitution for term precision (Lemma 37), because we know that $A_1 \sqsubseteq A'_1$. Choose $t'_2 = [A'_1/X]t'_3$ and the result follows, because $t'_1 \rightsquigarrow t'_2$.

Case. Suppose a congruence rule was used. Then $t_2 = [A_1]t''$. This case will follow straightforwardly by induction and a case split over which congruence rule was used.

Case.

$$\frac{\Gamma \vdash_{\mathsf{CG}} t : A_1 \quad \Gamma \vdash A_1 <: A_2}{\Gamma \vdash_{\mathsf{CG}} t : A_2}\ \textsc{sub}$$

In this case $t_1 = t$ and $A = A_2$. Suppose $\Gamma \vdash t_1 \sqsubseteq t'_1$ and $\Gamma \vdash_{\mathsf{CG}} t'_1 : A'$. Assume $t_1 \rightsquigarrow t_2$. Then by the induction hypothesis there is a t'_2 such that $t'_1 \rightsquigarrow^* t'_2$ and $\Gamma \vdash t_2 \sqsubseteq t'_2$, thus, we obtain our result.