The Combination of Dynamic and Static Typing from a Categorical Perspective

Harley Eades III and Michael Townsend
Augusta University
{heades,mitownsend}@augusta.edu

Categories and Subject Descriptors D.1.1 [*Programming Technique*]: [Applicative (Functional) Programming]; D.3.1 [*Formal Definitions and Theory*]: [Semantics]; F.3.3 [*Studies of Program Constructs*]: [Type structure]; F.4.1 [*Mathematical Logic and Formal Languages*]: [Lambda calculus and related systems]

General Terms typed lambda-calculus, untyped lambda-calculus, gradual typing, static typing, dynamic typing, categorical model, functional programming

Keywords static typing, dynamic typing, gradual typing, categorical semantics, retract

Abstract

Gradual typing was first proposed by Siek and Taha in 2006 as a way for a programming language to combine the strengths of both static and dynamic typing. However one question we must ask is, what is gradual typing? This paper contributes to answering this question by providing the first categorical model of gradual typing using the seminal work of Scott and Lambek on the categorical models of the untyped and typed λ-calculus. We then extract a functional programming language, called Grady, from the categorical model using the Curry-Howard-Lambek correspondence that combines both static and dynamic typing, but Grady is an annotated language and not a gradual type system. Finally, we show that Siek and Taha's gradual type system can be translated into Grady, and that their original annotated language is equivalent in expressive power to Grady.

1. Introduction

(Scott 1980) showed how to model the untyped λ-calculus within a cartesian closed category, $\mathcal{C}$, with a distinguished object we will call $?$ – read as the type of untyped terms – such that the object[1] $? \to ?$ is a retract of $?$. That is, there are morphisms $\mathsf{squash} : (? \to ?) \longrightarrow ?$ and $\mathsf{split} : ? \longrightarrow (? \to ?)$ where $\mathsf{squash}; \mathsf{split} = \mathsf{id} : (? \to ?) \longrightarrow (? \to ?)$[2]. For example, taking these morphisms as terms in the typed λ-calculus we can define the prototypical looping term $(\lambda x.x\,x)(\lambda x.x\,x)$ by $(\lambda x : ?.(\mathsf{split}\,x)\,x)\,(\mathsf{squash}\,(\lambda x : ?.(\mathsf{split}\,x)\,x))$.

In the same volume as Scott (Lambek 1980) showed that cartesian closed categories also model the typed λ-calculus. Suppose we want to model the typed λ-calculus with pairs and natural numbers. That is, given two types A_1 and A_2 there is a type $A_1 \times A_2$, and there is a type Nat. Furthermore, we have first and second projections, and zero and successor functions. This situation can easily be modeled by a cartesian closed category $\mathcal{C}$ – see Section 3 for the details – but also add to $\mathcal{C}$ the type of untyped terms $?$, squash, and split. At this point $\mathcal{C}$ is a model of both the typed and the untyped λ-calculus. However, the two theories are really just sitting side by side in $\mathcal{C}$ and cannot really interact much.

Suppose $\mathcal{T}$ is a discrete category with the objects Nat and 1 (the terminal object or empty product) and $\mathsf{T} : \mathcal{T} \longrightarrow \mathcal{C}$ is a full and faithful functor. This implies that $\mathcal{T}$ is a subcategory of $\mathcal{C}$, and that $\mathcal{T}$ is the category of atomic types. Then for any type A of $\mathcal{T}$ we add to $\mathcal{C}$ the morphisms $\mathsf{box} : \mathsf{T}A \longrightarrow ?$ and $\mathsf{unbox} : ? \longrightarrow \mathsf{T}A$ such that $\mathsf{box}; \mathsf{unbox} = \mathsf{id} : \mathsf{T}A \longrightarrow \mathsf{T}A$ making $\mathsf{T}A$ a retract of $?$. This is the bridge allowing the typed world to interact with the untyped one. We can think of box as injecting typed data into the untyped world, and unbox as taking it back. Notice that the only time we can actually get the typed data back out is if it were injected into the untyped world initially. In the model this is enforced through composition, but in the language this will be enforced at runtime, and hence, requires the language to contain dynamic typing. Thus, what we have just built up is a categorical model that offers a new perspective of how to combine static and dynamic typing.

(Siek and Taha 2006) define gradual typing to be the combination of both static and dynamic typing that allows for the programmer to program in dynamic style, and thus, annotations should be suppressed. This means that a gradually typed program can utilize both static types which will be enforced during compile time, but may also utilize dynamic typing that will be enforced during runtime. Therefore, gradual typing is the best of both worlds.

Siek and Taha's gradually typed functional language is the typed λ-calculus with the type of untyped terms $?$ and the following rules:

$$\frac{\Gamma \vdash t_1 : ? \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : ?} \qquad \frac{\Gamma \vdash t_1 : A_1 \to B \quad \Gamma \vdash t_2 : A_2 \quad A_1 \sim A_2}{\Gamma \vdash t_1\ t_2 : B}$$

The premise $A \sim B$ is read, the type A is consistent with the type B, and is defined in Figure 2. If we squint we can see split, squash, box, and unbox hiding in the definition of the previous rules, but they have been suppressed. We will show that when one uses either of the two typing rules then one is really implicitly using a casting morphism built from split, squash, box, and unbox. In fact, the consistency relation $A \sim B$ can be interpreted as such a morphism. Then the typing above can be read semantically as a saying if a

[1] We will use the terms "object" and "type" interchangeably.

[2] We denote composition of morphisms by $f; g : A \longrightarrow C$ given morphisms $f : A \longrightarrow B$ and $g : B \longrightarrow C$.

casting morphism exists, then the type really can be converted into the necessary type.

Contributions. This paper offers the following contributions:

- A new categorical model for gradual typing for functional languages. We show how to interpret (Siek and Taha 2006)'s gradual type system in the categorical model outlined above. As far as the authors are aware this is the first categorical model for gradual typing.
- We then extract a functional programming language called Grady from the categorical model via the Curry-Howard-Lambek correspondence. This is not a gradual type system, but can be seen as an alternative annotated language in which Siek and Taha's gradual type system can be translated to.
- A proof that Grady is as expressive as (Siek and Taha 2006)'s annotated language and vice versa. We give a type directed translation of Siek and Taha's annotated language to Grady and vice versa, then we show that these translations preserve evaluation.
- Having the untyped λ-calculus along side the typed λ-calculus can be a lot of fun. We show how to Church encode typed data, utilize the Y-combinator, and even obtain terminating recursion on natural numbers by combining the Y-combinator with a natural number eliminator. Thus, obtaining the expressive power of Gödel's system T (Girard et al. 1989).

Related work. TODO

2. Gradual Typing

We begin by introducing a slight variation of (Siek and Taha 2006)'s gradually typed functional language. It has been extended with product types and natural numbers, and instead of a big-step call-by-value operational semantics it uses a single-step type directed full $\beta\eta$-evaluator. One thing we strive for in this paper is to keep everything as simple as possible so that the underlying structure of these languages shines through. In this vein, the change in evaluation makes it easier to interpret the language into the categorical model.

The syntax of the gradual type system $\lambda^{?}_{\to}$ is defined in the following definition.

Definition 1. *Syntax for $\lambda^{?}_{\to}$:*

$$\begin{array}{lll} \textit{(types)} & A, B & ::= 1 \mid \mathsf{Nat} \mid ? \mid A \times B \mid A_1 \to A_2 \\ \textit{(terms)} & t & ::= x \mid \mathbf{triv} \mid 0 \mid \mathsf{succ}\, t \mid \lambda x : A.t \mid t_1\ t_2 \\ & & \quad \mid (t_1, t_2) \mid \mathsf{fst}\ t \mid \mathsf{snd}\ t \\ \textit{(contexts)} & \Gamma & ::= \cdot \mid x : A \mid \Gamma_1, \Gamma_2 \end{array}$$

This definition is the base syntax for every language in this paper. The typing rules are defined in Figure 1 and the type consistency relation is defined in Figure 2. The main changes of the version of $\lambda^{?}_{\to}$ defined here from the original due to (Siek and Taha 2006) is that products and natural numbers have been added. The definition of products follows how casting is done for functions. So it allows casting projections of products, for example, it is reasonable for terms like $\lambda x : (? \times ?).(\mathsf{succ}\,(\mathsf{fst}\ x))$ to type check.

We can view gradual typing as a surface language feature much like type inference, and we give it a semantics by translating it into an annotated core. (Siek and Taha 2006) do just that and give $\lambda^{?}_{\to}$ an operational semantics by translating it to a fully annotated core language called $\lambda^{\langle A \rangle}_{\to}$. Its syntax is an extension of the syntax of $\lambda^{?}_{\to}$ (Definition 1) where terms are the only syntactic class that differs, and so we do not repeat the syntax of types or contexts.

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \text{var} \qquad \frac{}{\Gamma \vdash \mathbf{triv} : 1}\ \text{unit} \qquad \frac{}{\Gamma \vdash 0 : \mathsf{Nat}}\ \text{zero}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\ t : \mathsf{Nat}}\ \text{succ} \qquad \frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 \times A_2}\ \times$$

$$\frac{\Gamma \vdash t : A_1 \times B \quad A_1 \sim A_2}{\Gamma \vdash \mathsf{fst}\ t : A_2}\ \times_{e_1}$$

$$\frac{\Gamma \vdash t : A \times B_1 \quad B_1 \sim B_2}{\Gamma \vdash \mathsf{snd}\ t : B_2}\ \times_{e_2}$$

$$\frac{\Gamma, x : A_1 \vdash t : A_2}{\Gamma \vdash \lambda x : A_1.t : A_1 \to A_2}\ \to$$

$$\frac{\Gamma \vdash t_1 : A_1 \to B \quad \Gamma \vdash t_2 : A_2 \quad A_1 \sim A_2}{\Gamma \vdash t_1\ t_2 : B}\ \to_e \qquad \frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{succ}\ t : ?}\ \text{succ}^{?}$$

$$\frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{fst}\ t : ?}\ \times^{?}_{e_1} \qquad \frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{snd}\ t : ?}\ \times^{?}_{e_2}$$

$$\frac{\Gamma \vdash t_1 : ? \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : ?}\ \to^{?}_{e}$$

Figure 1. Typing rules for $\lambda^{?}_{\to}$

$$\frac{}{A \sim A}\ \text{refl} \qquad \frac{}{A \sim ?}\ \text{box} \qquad \frac{}{? \sim A}\ \text{unbox}$$

$$\frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \to B_1 \sim A_2 \to B_2}\ \to \qquad \frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \times B_1 \sim A_2 \times B_2}\ \times$$

Figure 2. Type Consistency for $\lambda^{?}_{\to}$

Definition 2. *Syntax for $\lambda^{\langle A \rangle}_{\to}$:*

$$\begin{array}{lll} \textit{(simple values)} & s & ::= x \mid \mathbf{triv} \mid 0 \\ \textit{(values)} & v & ::= s \mid \langle ? \rangle s \\ \textit{(terms)} & t & ::= \ldots \mid \langle A \rangle t \end{array}$$

The typing rules for $\lambda^{\langle A \rangle}_{\to}$ can be found in Figure 3, and the reduction rules in Figure 4.

The major difference from the formalization of $\lambda^{\langle A \rangle}$ given here and Siek and Taha's is that it is single step and full β-reduction, but it is based on their original definition. The function $\mathsf{drop\text{-}cast}\ v$ is defined as follows:

$$\begin{array}{l} \mathsf{drop\text{-}cast}\ \langle ? \rangle s = s \\ \mathsf{drop\text{-}cast}\ s = s \end{array}$$

This function is used when casting values to their appropriate type.

Since the formalization of both $\lambda^{?}_{\to}$ and $\lambda^{\langle A \rangle}$ differ from their original definitions we give the definition of cast insertion in Figure 5, but this is only a slightly modified version from the one given by Siek and Taha.

3. The Categorical Model

The strength and main motivation for giving a categorical model to a programming language is that it can expose the fundamental structure of the language. This arises because a lot of the language features that often cloud the picture go away, for example, syntactic notions like variables disappear. This can often simplify things and

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \text{var} \qquad \frac{}{\Gamma \vdash \mathbf{triv} : 1}\ \text{unit} \qquad \frac{}{\Gamma \vdash 0 : \mathsf{Nat}}\ \text{zero}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, t : \mathsf{Nat}}\ \text{succ} \qquad \frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 \times A_2}\ \times$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\, t : A_1}\ \times_{e_1} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}\, t : A_2}\ \times_{e_2}$$

$$\frac{\Gamma, x : A_1 \vdash t : A_2}{\Gamma \vdash \lambda x : A_1.t : A_1 \to A_2}\ \to$$

$$\frac{\Gamma \vdash t_1 : A_1 \to A_2 \quad \Gamma \vdash t_2 : A_1}{\Gamma \vdash t_1\, t_2 : A_2}\ \to_e$$

$$\frac{\Gamma \vdash t : A \quad A \sim B}{\Gamma \vdash \langle B \rangle t : B}\ \text{cast}$$

Figure 3. Typing rules for $\lambda_{\to}^{\langle A \rangle}$

expose the underlying structure. Reynolds (**?**) was a big advocate for the use of category theory in programming language research for these reasons. For example, when giving the simply typed λ-calculus a categorical model we see that it is a cartesian closed category, but we also know that intuitionistic logic has the same model due to (Lambek 1980); on the syntactic side these two theories are equivalent as well due to (Howard 1980). Thus, the fundamental structure of the simply typed λ-calculus is intuitionistic logic. This also shows a relationship between seemingly unrelated theories. It is quite surprising that these two theories are related. Another more recent example of this can be found in the connection between dependent type theory and homotopy theory (**?**).

Another major benefit of studying the categorical model of programming languages is that it gives us a powerful tool to study language extensions. For example, purely functional programming in Haskell would not be where it is without the seminal work of Moggi and Wadler (**?**) on using monads – a purely categorical notion – to add side effects to Haskell. Thus, we believe that developing these types of models for new language designs and features can be hugely beneficial.

Interpreting a programming language into a categorical model requires three steps. First, the types are interpreted as objects. Then programs are interpreted as morphisms in the category, but this is a simplification. Every morphism, f, in a category has a source object and a target object, we usually denote this by $f : A \longrightarrow B$. Thus, in order to interpret programs as morphisms the program must have a source and target. So instead of interpreting raw terms as morphisms we interpret terms in their typing context. That is, we must show how to interpret every $\Gamma \vdash t : A$ as a morphism $t : [\![\Gamma]\!] \longrightarrow [\![A]\!]$. The third step is to show that whenever one program reduces to another their interpretations are isomorphic in the model. This means that whenever $\Gamma \vdash t_1 \rightsquigarrow t_2 : A$, then $[\![t_1]\!] \cong [\![t_2]\!] : [\![\Gamma]\!] \longrightarrow [\![A]\!]$. This is the reason why we defined our reduction in a typed fashion to aid us in understanding how it relates to the model. For a more thorough introduction see (Crole 1994).

We now give a categorical model for $\lambda_{\to}^{?}$ and $\lambda_{\to}^{\langle A \rangle}$. Then we complete the three steps summarized above. We will show how to interpret the typing of the former into the model, and then show how to do the same for the latter, furthermore, we show that reduction can be interpreted into the model as well, thus concluding soundness for $\lambda_{\to}^{\langle A \rangle}$ with respect to our model.

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash v \rightsquigarrow v : A}\ \text{value}$$

$$\frac{\Gamma \vdash \mathsf{drop\text{-}cast}\, v : C}{\Gamma \vdash \langle C \rangle v \rightsquigarrow \mathsf{drop\text{-}cast}\, v : C}\ \text{value-cast}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \langle \mathsf{Nat} \rangle (\mathsf{succ}\, t) \rightsquigarrow \mathsf{succ}\, \langle \mathsf{Nat} \rangle t : \mathsf{Nat}}\ \mathsf{Nat}\text{-cast}$$

$$\frac{\Gamma \vdash t : A_1 \to B_1 \quad (A_1 \to B_1) \sim (A_2 \to B_2)}{\Gamma \vdash \langle A_2 \to B_2 \rangle t \rightsquigarrow \lambda y : A_2.\langle B_2 \rangle (t\, \langle A_1 \rangle y) : A_2 \to B_2}\ \to\text{-cast}$$

$$\frac{\Gamma \vdash t : A_1 \times B_1 \quad (A_1 \times B_1) \sim (A_2 \times B_2)}{\Gamma \vdash \langle A_2 \times B_2 \rangle t \rightsquigarrow (\langle A_2 \rangle (\mathsf{fst}\, t), \langle B_2 \rangle (\mathsf{snd}\, t)) : A_2 \times B_2}\ \times\text{-cast}$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_2 : A \quad A \sim B}{\Gamma \vdash \langle B \rangle t_1 \rightsquigarrow \langle B \rangle t_2 : B}\ \text{cast}$$

$$\frac{\Gamma, x : A_1 \vdash t_2 : A_2 \quad \Gamma \vdash t_1 : A_1}{\Gamma \vdash (\lambda x : A_1.t_2)\, t_1 \rightsquigarrow [t_1/x]t_2 : A_2}\ \beta$$

$$\frac{\Gamma \vdash t : A_1 \to A_2 \quad x \notin \mathsf{FV}(t)}{\Gamma \vdash \lambda x : A_1.t\, x \rightsquigarrow t : A_1 \to A_2}\ \eta$$

$$\frac{\Gamma, x : A_1 \vdash t \rightsquigarrow t' : A_2}{\Gamma \vdash \lambda x : A_1.t \rightsquigarrow \lambda x : A_1.t' : A_1 \to A_2}\ \to$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \to A_2 \quad \Gamma \vdash t_2 : A_1}{\Gamma \vdash t_1\, t_2 \rightsquigarrow t_1'\, t_2 : A_2}\ \to_{e_1}$$

$$\frac{\Gamma \vdash t_1 : A_1 \to A_2 \quad \Gamma \vdash t_2 \rightsquigarrow t_2' : A_1}{\Gamma \vdash t_1\, t_2 \rightsquigarrow t_1\, t_2' : A_2}\ \to_{e_2}$$

$$\frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\, t \rightsquigarrow \mathsf{fst}\, t' : A_1}\ \times_{e_1}$$

$$\frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}\, t \rightsquigarrow \mathsf{snd}\, t' : A_2}\ \times_{e_2}$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash (\mathsf{fst}\, t, \mathsf{snd}\, t) \rightsquigarrow t : A_1 \times A_2}\ \times_{\eta}$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1', t_2) : A_1 \times A_2}\ \times_1$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 \rightsquigarrow t_2' : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1, t_2') : A_1 \times A_2}\ \times_2$$

Figure 4. Reduction rules for $\lambda_{\to}^{\langle A \rangle}$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x \Rightarrow x : A}\ \text{var} \qquad \frac{}{\Gamma \vdash 0 \Rightarrow 0 : A}\ \text{zero}$$

$$\frac{}{\Gamma \vdash \mathbf{triv} \Rightarrow \mathbf{triv} : A}\ \text{unit}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\, t_1 \Rightarrow \mathsf{succ}\, t_2 : \mathsf{Nat}}\ \text{succ}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{succ}\, t_1 \Rightarrow \langle ? \rangle \mathsf{succ}\, \langle \mathsf{Nat} \rangle t_2 : ?}\ \text{succ}^{?}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_3 : A_1 \quad \Gamma \vdash t_2 \Rightarrow t_4 : A_2}{\Gamma \vdash (t_1, t_2) \Rightarrow (t_3, t_4) : A_1 \times A_2}\ \times$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A_1 \times B \quad A_1 \sim A_2 \quad A_1 \neq A_2}{\Gamma \vdash \mathsf{fst}\, t_1 \Rightarrow \mathsf{fst}\, \langle A_2 \times B \rangle t_2 : A_2}\ \times^{\sim}_{e_1}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A \times B}{\Gamma \vdash \mathsf{fst}\, t_1 \Rightarrow \mathsf{fst}\, t_2 : A}\ \times_{e_1}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A \times B_1 \quad B_1 \sim B_2 \quad B_1 \neq B_2}{\Gamma \vdash \mathsf{snd}\, t_1 \Rightarrow \mathsf{snd}\, \langle A \times B_2 \rangle t_2 : B_2}\ \times^{\sim}_{e_2}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A \times B}{\Gamma \vdash \mathsf{snd}\, t_1 \Rightarrow \mathsf{snd}\, t_2 : B}\ \times_{e_2}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{fst}\, t_1 \Rightarrow \mathsf{fst}\, \langle ? \times ? \rangle t_2 : ?}\ \times^{?}_{e_1}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{snd}\, t_1 \Rightarrow \mathsf{snd}\, \langle ? \times ? \rangle t_2 : ?}\ \times^{?}_{e_2}$$

$$\frac{\Gamma, x : A_1 \vdash t_1 \Rightarrow t_2 : A_2}{\Gamma \vdash \lambda x : A_1 . t_1 \Rightarrow \lambda x : A_1 . t_2 : A_1 \to A_2}\ \to$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_3 : ? \quad \Gamma \vdash t_2 \Rightarrow t_4 : A}{\Gamma \vdash t_1\, t_2 \Rightarrow (\langle A \to ? \rangle t_3)\, t_4 : ?}\ \to^{?}_{e}$$

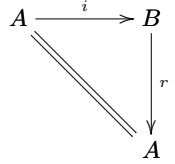
$$\frac{\begin{array}{l}\Gamma \vdash t_1 \Rightarrow t_3 : A_1 \to B \\ \Gamma \vdash t_2 \Rightarrow t_4 : A_2 \quad A_1 \sim A_2 \quad A_1 \neq A_2\end{array}}{\Gamma \vdash t_1\, t_2 \Rightarrow t_1\, \langle A_1 \rangle t : B}\ \to^{\sim}_{e}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_3 : A_1 \to A_2 \quad \Gamma \vdash t_2 \Rightarrow t_4 : A_1}{\Gamma \vdash t_1\, t_2 \Rightarrow t_3\, t_4 : A_2}\ \to_{e}$$

Figure 5. Cast Insertion

The model we develop here builds on the seminal work of (Lambek 1980) and (Scott 1980). (Lambek 1980) showed that the typed λ-calculus can be modeled by a cartesian closed category. In the same volume as Lambek, Scott essentially showed that the untyped λ-calculus is actually typed. That is, typed theories are more fundamental than untyped ones. He accomplished this by adding a single type, ?, and two functions $\mathsf{squash} : (? \to ?) \to ?$ and $\mathsf{split} : ? \to (? \to ?)$, such that, $\mathsf{squash}; \mathsf{split} = \mathsf{id} : (? \to ?) \to (? \to ?)$. At this point he was able to translate the untyped λ-calculus into this unityped one. Categorically, he modeled split and squash as the morphisms in a retract within a cartesian closed category – the same model as typed λ-calculus.

Definition 3. *Suppose $\mathcal{C}$ is a category. Then an object A is a **retract** of an object B if there are morphisms $i : A \longrightarrow B$ and $r : B \longrightarrow A$ such that the following diagram commutes:*



Thus, $? \to ?$ is a retract of ?, but we extend this slightly to include $? \times ?$ being a retract of ?. This is only a slight extension of Scott's model, because our languages will have products where he did not consider products, because he was considering the traditional definition of the untyped λ-calculus.

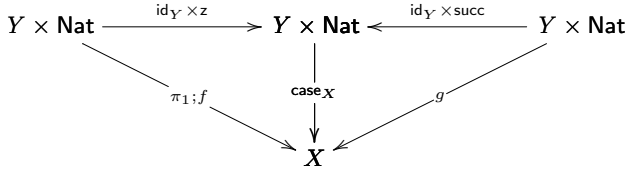
We can now define our categorical model of the untyped λ-calculus with products.

Definition 4. *An **untyped λ-model**, $(\mathcal{C}, ?, \mathsf{split}, \mathsf{squash})$, is a cartesian closed category $\mathcal{C}$ with a distinguished object ? and morphisms $\mathsf{squash} : S \longrightarrow ?$ and $\mathsf{split} : ? \longrightarrow S$ making the object S a retract of ?, where S is either $? \to ?$ or $? \times ?$.*

Theorem 5 (Scott (1980))**.** *An untyped λ-model is a sound and complete model of the untyped λ-calculus.*

Since all of the languages we are studying here contain the natural numbers we must be able to interpret them into our model. We give a novel approach to modeling the natural numbers with their (non-recursive) eliminator using what we call a Scott natural number object. Now the natural number eliminator is not part of $\lambda^{?}_{\to}$ or $\lambda^{\langle A \rangle}_{\to}$, but we want Grady to contain it, and Grady will directly correspond to the model.

Definition 6. *Suppose $\mathcal{C}$ is a cartesian closed category. A **Scott natural number object (SNNO)** is an object Nat of $\mathcal{C}$ and morphisms $\mathsf{z} : 1 \longrightarrow \mathsf{Nat}$ and $\mathsf{succ} : \mathsf{Nat} \longrightarrow \mathsf{Nat}$ of $\mathcal{C}$, such that, for any morphisms $f : Y \longrightarrow X$ and $g : Y \times \mathsf{Nat} \longrightarrow X$ of $\mathcal{C}$ there is a unique morphism $\mathsf{case}_X : Y \times \mathsf{Nat} \longrightarrow X$ making the following diagrams commute:*



Informally, the two diagrams essentially assert that we can define case_X as follows:

$$\begin{array}{rcl} \mathsf{case}_X\, y\, 0 & = & f\, y \\ \mathsf{case}_X\, y\, (\mathsf{succ}\, x) & = & g\, y\, x \end{array}$$

This formalization of natural numbers is inspired by the definition of Scott Numerals (**?**) where the notion of a case distinction is built

into the encoding. We can think of Y in the source object of case as the type of additional inputs that will be passed to both f and g, but we can think of Nat in the source object of case as the type of the scrutiny. Thus, since in the base case there is no predecessor, f, will not require the scrutiny, and so it is ignored.

One major difference between SNNOs and the more traditional natural number objects is that in the definition of the latter g is defined by well-founded recursion. However, SNNOs do not allow this, but in the presence of fixpoints we are able to regain this feature without having to bake it into the definition of natural number objects. However, to allow this we have found that when combining fixpoints and case analysis to define terminating functions on the natural numbers it is necessary to uniformly construct the input to both f and g due to the reduction rule of the Y combinator. Thus, we extend the type of f to $Y \times \mathsf{Nat}$, but then ignore the second projection when reaching the base case.

So far we can model the untyped and the typed λ-calculi within a cartesian closed category, but we do not have any way of moving typed data into the untyped part and vice versa. To accomplish this we add two new morphisms $\mathsf{box}_C : C \longrightarrow ?$ and $\mathsf{unbox}_C : ? \longrightarrow C$ such that $\mathsf{box}_C ; \mathsf{unbox}_C = \mathsf{id} : C \longrightarrow C$ for every atomic type C. Thus, each atomic type is a retract of ?. This enforces that the only time we can really consider something as typed is if it were boxed up in the first place. We can look at this from another perspective as well. If the programmer tries to unbox something that is truly untyped, then their program may actually type check, but they will obtain a dynamic type error at runtime, because the unbox will never have been matched up with the correct boxed data. For example, we can cast 3 to type Bool by $\mathsf{unbox}_{\mathsf{Bool}}(\mathsf{box}_{\mathsf{Nat}}\, 3)$, but if this program is every run, then we will obtain a dynamic type error. Note that we can type the previous program in $\lambda^{\langle A \rangle}_{\rightarrow}$ as well, but if we run the program it will result in a dynamic type error too.

Now we combine everything we have discussed so far to obtain the categorical model.

Definition 7. *A **gradual λ-model**, $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$, where $\mathcal{T}$ is a discrete category with at least two objects* Nat *and* 1*, $\mathcal{C}$ is a cartesian closed category with a SNNO, $(\mathcal{C}, ?, \mathsf{split}, \mathsf{squash})$ is an untyped λ-model, $\mathsf{T} : \mathcal{T} \longrightarrow \mathcal{C}$ is an embedding – a full and faithful functor that is injective on objects – and for every object A of $\mathcal{T}$ there are morphisms $\mathsf{box}_A : TA \longrightarrow ?$ and $\mathsf{unbox}_A : ? \longrightarrow TA$ making TA a retract of ?.*

We call the category $\mathcal{T}$ the category of atomic types. We call an object, A, **atomic** iff there is some object A' in $\mathcal{T}$ such that $A = \mathsf{T}A'$. Note that we do not consider ? an atomic type. The model really is the cartesian closed category $\mathcal{C}$, but it is extended with the structure of both the typed and the untyped λ-calculus with the ability to cast data.

Interpreting the typing rules for $\lambda^{?}_{\rightarrow}$ will require the interpretation of type consistency. Thus, we must be able to cast any type A to ?, but as stated the model only allows atomic types to be casted. It turns out that this can be lifted to any type.

To cast any type A to ? we will build casting morphisms that first take the object A to its skeleton, and then takes the skeleton to ?.

Definition 8. *Suppose $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ is a gradual λ-model. Then the **skeleton** of an object A of $\mathcal{C}$ is an object S that is constructed by replacing each atomic type in A with ?. Given an object A we denote its skeleton by* skeleton A.

One should think of the skeleton of an object as the supporting type structure of the object, but we do not know what kind of data is actually in the structure. For example, the skeleton of the object Nat is ?, and the skeleton of $(\mathsf{Nat} \times 1) \rightarrow \mathsf{Nat} \rightarrow \mathsf{Nat}$ is $(? \times ?) \rightarrow ? \rightarrow ?$.

The next definition defines a means of constructing a casting morphism that casts a type A to its skeleton and vice versa. This definition is by mutual recursion on the input type.

Definition 9. *Suppose $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ is a gradual λ-model. Then for any object A whose skeleton is S we define the functions $\widehat{\mathsf{box}}\, A$ and $\widehat{\mathsf{unbox}}\, S\, A$ by mutual recursion on A as follows:*

$\widehat{\mathsf{box}}\, ?\, A = \mathsf{box}_A$
when A is atomic

$\widehat{\mathsf{box}}\, ?\, ? = \mathsf{id}_?$

$\widehat{\mathsf{box}}\, (A_1 \rightarrow A_2) = (\widehat{\mathsf{unbox}}\, (\mathsf{skeleton}\, A_1)\, A_1) \rightarrow (\widehat{\mathsf{box}}\, S_2\, A_2)$

$\widehat{\mathsf{box}}\, (A_1 \times A_2) = (\widehat{\mathsf{box}}\, A_1) \times (\widehat{\mathsf{box}}\, A_2)$

$\widehat{\mathsf{unbox}}\, ?\, A = \mathsf{unbox}_A$
when A is atomic

$\widehat{\mathsf{unbox}}\, ?\, ? = \mathsf{id}_?$

$\widehat{\mathsf{unbox}}\, (S_1 \rightarrow S_2)\, (A_1 \rightarrow A_2) = (\widehat{\mathsf{box}}\, A_1) \rightarrow (\widehat{\mathsf{unbox}}\, S_2\, A_2)$

$\widehat{\mathsf{unbox}}\, (S_1 \times S_2)\, (A_1 \times A_2) = (\widehat{\mathsf{unbox}}\, S_1\, A_1) \times (\widehat{\mathsf{unbox}}\, S_2\, A_2)$

4. Grady

5. Grady and Gradual Typing

References

Roy L. Crole. *Categories for Types*. Cambridge University Press, Jan 1994. ISBN 9781139172707. doi: 10.1017/CBO9781139172707.

Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, April 1989.

W. A. Howard. The formulae-as-types notion of construction. *To H. B. Curry: Essays on Combznatory Logic, Lambda-Calculus, and Formalism*, pages 479–490, 1980.

Joachim Lambek. From lambda calculus to cartesian closed categories. *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 376–402, 1980.

Dana Scott. Relating theories of the lambda-calculus. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism (eds. Hindley and Seldin)*, pages 403–450. Academic Press, 1980.

Jeremy G Siek and Walid Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop*, volume 6, pages 81–92, 2006.

A. The Complete Spec of Grady

$termvar,\ x,\ y,\ z$			
$index,\ k$			
$t,\ v,\ s$	::=		term
	|	x	variable
	|	$\mathbf{triv}$	unit
	|	squash_S	injection of the retract
	|	split_S	surjection of the retract
	|	box_C	generalize to the untyped universe
	|	unbox_C	specialize the untyped universe to a speci
	|	$\langle A \rangle t$	type cast
	|	$\lambda x : A.t$	λ-abstraction

	$t_1\ t_2$		function application
|	(t_1, t_2)		pair constructor
|	$\mathsf{fst}\ t$		first projection
|	$\mathsf{snd}\ t$		second projection
|	$\mathsf{succ}\ t$		successor function
|	0		zero
|	(t)	S	

h	::=		head-normal forms
	|	$\mathbf{triv}$	
	|	split_S	
	|	squash_S	
	|	box_C	
	|	unbox_C	
	|	$\lambda x : A.t$	
	|	(t_1, t_2)	
	|	$\mathsf{fst}\ t$	
	|	$\mathsf{snd}\ t$	
	|	$\mathsf{succ}\ t$	
	|	0	

T	::=		terminating types
	|	1	unit type
	|	Nat	natural number type
	|	$T_1 \to T_2$	function type
	|	$T_1 \times T_2$	cartesian product type
	|	(T)	S

$A,\ B,\ C,\ S,\ X,\ Y,\ Z$	::=		type
	|	1	unit type
	|	Nat	natural number type
	|	$?$	untyped universe
	|	$A_1 \to A_2$	function type
	|	$A_1 \times A_2$	cartesian product type
	|	(A)	S

Γ	::=		typing context
	|	$.$	empty context
	|	$\Gamma, x : A$	cons

vd	::=		
	|	$\vdash$	
	|	$\nvdash$	

$\boxed{A \sim B}$ A is consistent with B

$$\frac{}{A \sim A}$$

$$\frac{}{A \sim ?}$$

$$\frac{}{? \sim A}$$

$$\frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \to B_1 \sim A_2 \to B_2}$$

$$\frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \times B_1 \sim A_2 \times B_2}$$

$\boxed{\Gamma vd t : A}$ t has type A in context Γ

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \text{var}$$

$$\frac{}{\Gamma \vdash \mathsf{box}_C : C \to ?}\ Box$$

$$\frac{}{\Gamma \vdash \mathsf{unbox}_C : ? \to C}\ Unbox$$

$$\frac{}{\Gamma \vdash \mathsf{squash}_S : S \to ?}\ squash$$

$$\frac{}{\Gamma \vdash \mathsf{split}_S : ? \to S}\ split$$

$$\frac{}{\Gamma \vdash \mathbf{triv} : 1}\ \text{unit}$$

$$\frac{}{\Gamma \vdash 0 : \mathsf{Nat}}\ \text{zero}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\ t : \mathsf{Nat}}\ \text{succ}$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 \times A_2}\ \times$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\ t : A_1}\ \times_{e_1}$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}\ t : A_2}\ \times_{e_2}$$

$$\frac{\Gamma, x : A_1 \vdash t : A_2}{\Gamma \vdash \lambda x : A_1.t : A_1 \to A_2}\ \to$$

$$\frac{\Gamma \vdash t_1 : A_1 \to A_2 \quad \Gamma \vdash t_2 : A_1}{\Gamma \vdash t_1\ t_2 : A_2}\ \to_e$$

$$\frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{succ}\ t : ?}\ \text{succ}^?$$

$$\frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{fst}\ t : ?}\ \times^?_{e_1}$$

$$\frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{snd}\ t : ?}\ \times^?_{e_2}$$

$$\frac{\Gamma \vdash t_1 : ? \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1\ t_2 : ?}\ \to^?_e$$

$$\frac{\Gamma \vdash t_1 : A_1 \to B \quad \Gamma \vdash t_2 : A_2 \quad A_1 \sim A_2}{\Gamma \vdash t_1\ t_2 : B}\ \to_e$$

$$\frac{\Gamma \vdash t : A_1 \times B \quad A_1 \sim A_2}{\Gamma \vdash \mathsf{fst}\ t : A_2}\ \times_{e_1}$$

$$\frac{\Gamma \vdash t : A \times B_1 \quad B_1 \sim B_2}{\Gamma \vdash \mathsf{snd}\ t : B_2}\ \times_{e_2}$$

$$\frac{\Gamma \vdash t : A \quad A \sim B}{\Gamma \vdash \langle B \rangle t : B}\ \text{cast}$$

$\boxed{\Gamma \vdash t_1 \rightsquigarrow t_2 : A}$ t_1 reduces to t_2 with type A in context Γ

$$\frac{\Gamma \vdash s : A}{\Gamma \vdash s \rightsquigarrow s : A}\ rd\_values$$

$$\frac{\Gamma \vdash t : C}{\Gamma \vdash \mathsf{unbox}_C\ (\mathsf{box}_C\ t) \rightsquigarrow t : C}\ rd\_retractT$$

$$\frac{\Gamma \vdash t : S}{\Gamma \vdash \mathsf{split}_S\ (\mathsf{squash}_S\ t) \rightsquigarrow t : S}\ rd\_retractU$$

$$\frac{\Gamma, x : A_1 \vdash t_2 : A_2 \quad \Gamma \vdash t_1 : A_1}{\Gamma \vdash (\lambda x : A_1.t_2)\ t_1 \rightsquigarrow [t_1/x]t_2 : A_2}\ rd\_beta$$

$$\frac{\Gamma \vdash t : A_1 \to A_2 \quad x \notin \mathsf{FV}(t)}{\Gamma \vdash \lambda x : A_1.t\ x \rightsquigarrow t : A_1 \to A_2}\ rd\_eta$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \mathsf{fst}\ (t_1, t_2) \rightsquigarrow t_1 : A_1}\ rd\_proj1$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \mathsf{snd}\,(t_1, t_2) \rightsquigarrow t_2 : A_2}\ rd\_proj2$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash (\mathsf{fst}\,t, \mathsf{snd}\,t) \rightsquigarrow t : A_1 \times A_2}\ rd\_etaP$$

$$\frac{\Gamma, x : A_1 \vdash t \rightsquigarrow t' : A_2}{\Gamma \vdash \lambda x : A_1.t \rightsquigarrow \lambda x : A_1.t' : A_1 \to A_2}\ rd\_lam$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \to A_2 \quad \Gamma \vdash t_2 : A_1}{\Gamma \vdash t_1\ t_2 \rightsquigarrow t_1'\ t_2 : A_2}\ rd\_app1$$

$$\frac{\Gamma \vdash t_1 : A_1 \to A_2 \quad \Gamma \vdash t_2 \rightsquigarrow t_2' : A_1}{\Gamma \vdash t_1\ t_2 \rightsquigarrow t_1\ t_2' : A_2}\ rd\_app2$$

$$\frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\,t \rightsquigarrow \mathsf{fst}\,t' : A_1}\ rd\_fst$$

$$\frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}\,t \rightsquigarrow \mathsf{snd}\,t' : A_2}\ rd\_snd$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1', t_2) : A_1 \times A_2}\ rd\_pair1$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 \rightsquigarrow t_2' : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1, t_2') : A_1 \times A_2}\ rd\_pair2$$

$\boxed{\Gamma \vdash t_1 \rightsquigarrow t_2 : A}$ Reduction for annotated Siek16

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash v \rightsquigarrow v : A}\ \text{value}$$

$$\frac{\Gamma \vdash \mathsf{drop\text{-}cast}\ v : C}{\Gamma \vdash \langle C \rangle v \rightsquigarrow \mathsf{drop\text{-}cast}\ v : C}\ \text{value-cast}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \langle \mathsf{Nat} \rangle (\mathsf{succ}\ t) \rightsquigarrow \mathsf{succ}\ \langle \mathsf{Nat} \rangle t : \mathsf{Nat}}\ \text{Nat-cast}$$

$$\frac{\Gamma \vdash t : A_1 \to B_1 \quad (A_1 \to B_1) \sim (A_2 \to B_2)}{\Gamma \vdash \langle A_2 \to B_2 \rangle t \rightsquigarrow \lambda y : A_2.\langle B_2 \rangle (t\ \langle A_1 \rangle y) : A_2 \to B_2}\ \to\text{-cast}$$

$$\frac{\Gamma \vdash t : A_1 \times B_1 \quad (A_1 \times B_1) \sim (A_2 \times B_2)}{\Gamma \vdash \langle A_2 \times B_2 \rangle t \rightsquigarrow (\langle A_2 \rangle (\mathsf{fst}\,t), \langle B_2 \rangle (\mathsf{snd}\,t)) : A_2 \times B_2}\ \times\text{-cast}$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_2 : A \quad A \sim B}{\Gamma \vdash \langle B \rangle t_1 \rightsquigarrow \langle B \rangle t_2 : B}\ \text{cast}$$

$$\frac{\Gamma, x : A_1 \vdash t_2 : A_2 \quad \Gamma \vdash t_1 : A_1}{\Gamma \vdash (\lambda x : A_1.t_2)\ t_1 \rightsquigarrow [t_1/x]t_2 : A_2}\ \beta$$

$$\frac{\Gamma \vdash t : A_1 \to A_2 \quad x \notin \mathsf{FV}(t)}{\Gamma \vdash \lambda x : A_1.t\ x \rightsquigarrow t : A_1 \to A_2}\ \eta$$

$$\frac{\Gamma, x : A_1 \vdash t \rightsquigarrow t' : A_2}{\Gamma \vdash \lambda x : A_1.t \rightsquigarrow \lambda x : A_1.t' : A_1 \to A_2}\ \to$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \to A_2 \quad \Gamma \vdash t_2 : A_1}{\Gamma \vdash t_1\ t_2 \rightsquigarrow t_1'\ t_2 : A_2}\ \to_{e_1}$$

$$\frac{\Gamma \vdash t_1 : A_1 \to A_2 \quad \Gamma \vdash t_2 \rightsquigarrow t_2' : A_1}{\Gamma \vdash t_1\ t_2 \rightsquigarrow t_1\ t_2' : A_2}\ \to_{e_2}$$

$$\frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\,t \rightsquigarrow \mathsf{fst}\,t' : A_1}\ \times_{e_1}$$

$$\frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}\,t \rightsquigarrow \mathsf{snd}\,t' : A_2}\ \times_{e_2}$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash (\mathsf{fst}\,t, \mathsf{snd}\,t) \rightsquigarrow t : A_1 \times A_2}\ \times_{\eta}$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1', t_2) : A_1 \times A_2}\ \times_1$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 \rightsquigarrow t_2' : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1, t_2') : A_1 \times A_2}\ \times_2$$

$\boxed{\Gamma \vdash t_1 \Rightarrow t_2 : A}$ Cast insertion from Siek16

$$\frac{x : A\ \in\ \Gamma}{\Gamma \vdash x \Rightarrow x : A}\ \text{var}$$

$$\frac{}{\Gamma \vdash 0 \Rightarrow 0 : A}\ \text{zero}$$

$$\frac{}{\Gamma \vdash \mathbf{triv} \Rightarrow \mathbf{triv} : A}\ \text{unit}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\ t_1 \Rightarrow \mathsf{succ}\ t_2 : \mathsf{Nat}}\ \text{succ}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_3 : A_1 \quad \Gamma \vdash t_2 \Rightarrow t_4 : A_2}{\Gamma \vdash (t_1, t_2) \Rightarrow (t_3, t_4) : A_1 \times A_2}\ \times$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A_1 \times B \quad A_1 \sim A_2 \quad A_1 \neq A_2}{\Gamma \vdash \mathsf{fst}\ t_1 \Rightarrow \mathsf{fst}\ \langle A_2 \times B \rangle t_2 : A_2}\ \times_{e_1}^{\sim}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A \times B}{\Gamma \vdash \mathsf{fst}\ t_1 \Rightarrow \mathsf{fst}\ t_2 : A}\ \times_{e_1}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A \times B_1 \quad B_1 \sim B_2 \quad B_1 \neq B_2}{\Gamma \vdash \mathsf{snd}\ t_1 \Rightarrow \mathsf{snd}\ \langle A \times B_2 \rangle t_2 : B_2}\ \times_{e_2}^{\sim}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : A \times B}{\Gamma \vdash \mathsf{snd}\ t_1 \Rightarrow \mathsf{snd}\ t_2 : B}\ \times_{e_2}$$

$$\frac{\Gamma, x : A_1 \vdash t_1 \Rightarrow t_2 : A_2}{\Gamma \vdash \lambda x : A_1.t_1 \Rightarrow \lambda x : A_1.t_2 : A_1 \to A_2}\ \to$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_3 : ? \quad \Gamma \vdash t_2 \Rightarrow t_4 : A}{\Gamma \vdash t_1\ t_2 \Rightarrow (\langle A \to ? \rangle t_3)\ t_4 : ?}\ \to_e^?$$

$$\frac{\begin{array}{l}\Gamma \vdash t_1 \Rightarrow t_3 : A_1 \to B \\ \Gamma \vdash t_2 \Rightarrow t_4 : A_2 \quad A_1 \sim A_2 \quad A_1 \neq A_2\end{array}}{\Gamma \vdash t_1\ t_2 \Rightarrow t_1\ \langle A_1 \rangle t : B}\ \to_e^{\sim}$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_3 : A_1 \to A_2 \quad \Gamma \vdash t_2 \Rightarrow t_4 : A_1}{\Gamma \vdash t_1\ t_2 \Rightarrow t_3\ t_4 : A_2}\ \to_e$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{succ}\ t_1 \Rightarrow \langle ? \rangle \mathsf{succ}\ \langle \mathsf{Nat} \rangle t_2 : ?}\ \text{succ}^?$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{fst}\ t_1 \Rightarrow \mathsf{fst}\ \langle ? \times ? \rangle t_2 : ?}\ \times_{e_1}^?$$

$$\frac{\Gamma \vdash t_1 \Rightarrow t_2 : ?}{\Gamma \vdash \mathsf{snd}\ t_1 \Rightarrow \mathsf{snd}\ \langle ? \times ? \rangle t_2 : ?}\ \times_{e_2}^?$$