# 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 $\lambda$-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 $\lambda$-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 $\lambda$-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 $\lambda$-calculus. Suppose we want to model the typed $\lambda$-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 $\mathsf{Nat}$. 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 $?$, $\mathsf{squash}$, and $\mathsf{split}$. At this point $\mathcal{C}$ is a model of both the typed and the untyped $\lambda$-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 $\mathsf{Nat}$ 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 $\mathsf{box}$ as injecting typed data into the untyped world, and $\mathsf{unbox}$ 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 $\lambda$-calculus with the type of untyped terms $?$ and the following rules:

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

The premise $A \sim B$ is read, the type $A$ is consistent with the type $B$, and is defined by the following rules:

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

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

[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 $g : B \longrightarrow C$.

This is a reflexive and symmetric relation, but is a non-transitive relation. 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 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 $\lambda$-calculus along side the typed $\lambda$-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 $\lambda^?_\to$:*

$$\textit{(types)} \quad A, B ::= 1 \mid \mathsf{Nat} \mid ? \mid A \times B \mid A_1 \to A_2$$

$$\textit{(terms)} \quad t ::= x \mid \mathbf{triv} \mid 0 \mid \mathsf{succ}\ t \mid \lambda x : A.t \mid t_1\ t_2 \mid (t_1, t_2) \mid \mathsf{fst}\ t \mid \mathsf{snd}\ t$$

$$\textit{(contexts)} \quad \Gamma ::= \cdot \mid x : A \mid \Gamma_1, \Gamma_2$$

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.

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

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

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

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

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

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

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

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

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

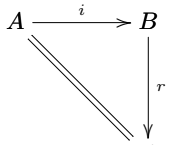
$$\frac{}{A \sim A}\ \text{refl} \qquad \frac{}{A \sim ?}\ \text{box} \qquad \frac{}{? \sim A}\ \text{unbox}$$

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

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

## 3. The Categorical Model

**Definition 2.** *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:*

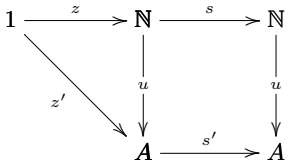



**Definition 3.** *An **untyped $\lambda$-model**, $(\mathcal{C}, ?, \mathsf{split}, \mathsf{squash})$, is a cartesian closed category $\mathcal{C}$ with a distinguished object $?$ and two morphisms $\mathsf{squash} : (? \to ?) \longrightarrow ?$ and $\mathsf{split} : ? \longrightarrow (? \to ?)$ making the object $? \to ?$ a retract of $?$.*

**Theorem 4** (Scott (1980))**.** *An untyped $\lambda$-model is a sound and complete model of the untyped $\lambda$-calculus.*

**Definition 5.** *An object $\mathbb{N}$ of a category $\mathcal{C}$ with a terminal object $1$ is a **natural number object (NNO)** if and only if there are morphisms $z : 1 \longrightarrow \mathbb{N}$ and $s : \mathbb{N} \longrightarrow \mathbb{N}$ such that for any other object $A$ and morphisms $z' : 1 \longrightarrow A$ and $s' : A \longrightarrow A$ there is a unique morphism $u : \mathbb{N} \longrightarrow A$ making the following diagram*

*commute:*




**Definition 6.** *A **gradual λ-model**, $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$, where $\mathcal{T}$ and $\mathcal{C}$ are cartesian closed categories with NNOS, $(\mathcal{C}, ?, \mathsf{split}, \mathsf{squash})$ is an untyped λ-model, $\mathsf{T} : \mathcal{T} \longrightarrow \mathcal{C}$ is a cartesian closed embedding – a full and faithful cartesian closed functor that is injective on objects and preserves the NNO – 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 $?$.*

## 4. Grady

## 5. Grady and Gradual Typing

## References


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

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

$termvar,\ x,\ y,\ z$
$index,\ k$

| | | | |
|---|---|---|---|
| $t$ | ::= | | term |
| | \| | $x$ | variable |
| | \| | $\mathbf{triv}$ | unit |
| | \| | $\mathsf{squash}_S$ | injection of the retract |
| | \| | $\mathsf{split}_S$ | surjection of the retract |
| | \| | $\mathsf{box}_C$ | generalize to the untyped universe |
| | \| | $\mathsf{unbox}_C$ | specialize the untyped universe to a specific type |
| | \| | $\langle A \rangle t$ | 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 | |

| | | | |
|---|---|---|---|
| $s$ | ::= | | simple values |
| | \| | $x$ | |
| | \| | $\mathbf{triv}$ | unit |
| | \| | $0$ | zero |

| | | | |
|---|---|---|---|
| $v$ | ::= | | values |
| | \| | $x$ | |
| | \| | $\mathbf{triv}$ | unit |
| | \| | $0$ | zero |
| | \| | $\langle ? \rangle s$ | untype cast |

| | | | |
|---|---|---|---|
| $h$ | ::= | | head-normal forms |
| | \| | $\mathbf{triv}$ | |
| | \| | $\mathsf{split}_S$ | |
| | \| | $\mathsf{squash}_S$ | |
| | \| | $\mathsf{box}_C$ | |
| | \| | $\mathsf{unbox}_C$ | |
| | \| | $\lambda x : A.t$ | |
| | \| | $(t_1, t_2)$ | |
| | \| | $\mathsf{fst}\ t$ | |
| | \| | $\mathsf{snd}\ t$ | |
| | \| | $\mathsf{succ}\ t$ | |
| | \| | $0$ | |

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

| | | | |
|---|---|---|---|
| $C$ | ::= | | atomic type |
| | \| | $1$ | unit type |
| | \| | $\mathsf{Nat}$ | natural number type |

| | | | |
|---|---|---|---|
| $A,\ B,\ S$ | ::= | | type |
| | \| | $1$ | unit type |
| | \| | $\mathsf{Nat}$ | natural number type |
| | \| | $?$ | untyped universe |
| | \| | $A_1 \to A_2$ | function type |
| | \| | $A_1 \times A_2$ | cartesian product type |
| | \| | $(A)$ S | |

| | | | |
|---|---|---|---|
| $\Gamma$ | ::= | | typing context |
| | \| | $\cdot$ | empty context |
| | \| | $\Gamma, x : A$ | cons |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{fst}\ t : ?}$$

$$\frac{\Gamma \vdash t : ?}{\Gamma \vdash \mathsf{snd}\ t : ?}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash v \rightsquigarrow v : A}\ rdA\_values$$

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

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \langle \mathsf{Nat} \rangle (\mathsf{succ}\ t) \rightsquigarrow \mathsf{succ}\ \langle \mathsf{Nat} \rangle t : \mathsf{Nat}}\ rdA\_castNat$$

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

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

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

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

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

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

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

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

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

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

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

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

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