Dependently-Typed Programming with Scott Encoding

Peng Fu, Aaron Stump
Computer Science, The University of Iowa

Abstract
We introduce Selfstar, a Curry-style dependent type system featuring self type \( \tau \mapsto T \), together with mutually recursive definitions and \( \tau : \tau \). We show how to obtain Scott-encoded datatypes and the corresponding elimination schemes with Selfstar. Examples such as numerals, vector are given to demonstrate the power of Selfstar as a dependently-typed programming language. Standard metatheorems such as type preservation are proved.

1. Introduction
Self type is originated from our previous work on S [5], since then self type has been studied in combination with different typing principles. In this paper, we study a system called Selfstar, which combines self type together with \( \tau : \tau \) and mutually recursive definitions. In Selfstar, every type is inhabited, so Selfstar is inconsistent as a logic. The only logical feature in Selfstar is the Leibniz convertibility, i.e. we define \( t_1 =_A t_2 \) to be \( \Pi C : A \to \ast.\Pi t_3 \to C t_1 \to C t_2 \). Note that we use “convertibility” instead of “equality” to indicate one can not interpret \( t_1 =_A t_2 \) as a formula. If we know the inhabitant of \( t_1 =_A t_2 \) is normalized at the term \( \lambda C.\lambda x.x \), then we can use \( t_1 =_A t_2 \) to cast the type \( P t_1 \) to \( P t_2 \) by applying the term \( (\lambda C.\lambda x.x) P \) to the inhabitant of \( P t_1 \). Note that \( (\lambda C.\lambda x.x) P \to_\beta \lambda x.x \), so the casting will not affect the inhabitant of \( P t_1 \).

Scott encoding (reported in [4]) does not suffer from the inefficiency problem arisen in Church encoding. For functional programming language, Scott encoding seems to be a better fit than Church encoding [9]. From the typing perspective, each Scott-encoded data contains its subdata, one would need recursive definition in order to define a type for Scott-encoded data. Elimination schemes for the Scott-encoded data are derivable in Selfstar, this means programmer can write down programs that have types like \( \Pi x : \text{Nat}.\text{add } x \ 0 =\text{Nat } x \), which increases the flexibility of type-level casting.

The main contributions of this paper are:

- We present Selfstar, which allows us to type Scott-encoded data and derive elimination schemes for Scott-encoded data. Selfstar simplifies the design of the functional programming language, since the primitive notion of inductive data and pattern matching is not needed in Selfstar.
- We prove type preservation and progress for Selfstar by applying the method we developed in the study of System S.

1.1 Motivation
In Curry style system \( T \) [6] equipped with polymorphic and dependent type, one has a primitive notion of \( \text{recursor} \), namely, rec : \( \Pi x : \text{Nat} \Pi U.(\text{Nat} \to U \to U) \to U \to U \) and two reductions rules: rec \( 0 \ f \ v \to v \) and rec \( (Sn) \ f \ v \to f \ n \ (\text{rec } n \ f \ v) \).

The recursor can be emulated with lambda terms. For example, \( \text{rec } : \lambda n.\lambda f.\lambda v.\lambda 0.\lambda n f v \) with the notion of numeral \( 0 := \lambda s.\lambda z.z \) and \( n := \lambda s.\lambda z.s n (\Pi n \ s \ z) \). One can verify that the definition of rec in lambda calculus behaves the same as the one in system \( T \). With recursive definition, we can define \( \text{Nat} := \forall U.(\text{Nat} \to U \to U) \to U \to U \). Note that the type of \( n \) is the same as the type of \( \text{rec } n \).

So far the type of the recursor is elementary, i.e., not involving dependency. To make real use of dependent type, we ask if it is possible to obtain a type likes \( \Pi x : \text{Nat} \forall U : \text{Nat} \to +.\Pi y : \text{Nat}.U y \to U (\text{Sym}) \to U 0 \to U x \). Note that \( S := \lambda n.\lambda s.\lambda z.s n (\Pi n \ s \ z) \). We want to emphasize the underlying computational behavior of this type should be the same as rec, thus we want a typing relation \( \text{rec } : \forall U : \text{Nat} \to +.\Pi x : \text{Nat}.\Pi y : \text{Nat}.U y \to U (\text{Sym}) \to U 0 \to U x \). We also want the type of \( \text{rec } n \), namely \( \Pi y : \text{Nat}.U y \to U (\text{Sym}) \to U 0 \to U n \) to be the same as the type of \( n \). So we want the following typing relation: \( n : (\Pi y : \text{Nat}.U y \to U (\text{Sym}) \to U 0 \to U n) \) for any \( n \). We know the following self type mechanism:

\[
\begin{align*}
\Gamma \vdash t : [x:T] & \quad \text{selfGen} \quad \Gamma \vdash t : [x:T] & \quad \text{selfInst}
\end{align*}
\]

So it is not surprising we define \( \text{Nat} := \Pi x : \text{Nat}.U y \to U (\text{Sym}) \to U 0 \to U x \). With \( \text{selfGen} \), \( \text{selfInst} \) and mutually recursive definition, one can verify that indeed, the type of \( n \) is the same as the type of \( \text{rec } n \) and the type of \( \text{rec } \Pi x : \forall U : \text{Nat} \to +.\Pi y : \text{Nat}.U y \to U (\text{Sym}) \to U 0 \to U x \). It is tempting to claim that rec represents the induction principle, but it is not for the following two reasons: 1. With mutually recursive definitions, the types can not be interpreted as formulas. 2. The dependent product \( \Pi \) is not exactly the first order quantifier \( \forall \).

System \( T \) is closed to the functional programming language, but still a little far from a functional programmer’s usual experience. In modern functional programming language, one would want to write a plus 2 function in the following style:

\[
\begin{align*}
data \text{Nat} & = \text{Zero} \\
& \mid \text{Succ } \text{Nat} \\
\text{plusTwo } & :: \text{Nat} \to \text{Nat} \\
\text{plusTwo } n & = \text{case } n \text{ of} \\
& \text{Succ } p \to \text{Succ } (\text{plusTwo } p) \\
& \text{Zero} \to \text{Succ } (\text{Succ } \text{Zero})
\end{align*}
\]
With Scott numerals, we can achieve the effects above. Assume Scott numerals and mutually recursive definition. We define:

\[
\begin{align*}
\text{Zero} &= \lambda s. \lambda z. z \\
\text{Succ } n &= \lambda s. \lambda z. s \text{ Succ } n \\
\text{plusTwo } n &= \text{case } n \lambda p. \text{ Succ } (\text{plusTwo } p) \\
&\quad \text{Succ } (\text{Succ } \text{Zero})
\end{align*}
\]

Of course, case := \(\lambda n.\lambda f.\lambda a. n \cdot f \cdot a\) and lam denotes the usual \(\lambda\). One can see the differences between the two programs above are mostly superficial. Now let us first give a elementary version of type for case, which is \((\text{Nat } \to U) \to U \to U\). The dependency version is \(\Pi x : \text{Nat} \cdot (\Pi y : \text{Nat}.U (\Pi y)) \to U\). And we define \(\text{Nat} := \lambda x. (\Pi y : \text{Nat}.U (\Pi y)) \to U\). Thus operations on Scott data is typable using the elimina- tion scheme. Thus operations on Scott-encoded data is typable using the selfInst rule.

Observe that the term for rec and case (even the iterator in system \(F\)) is the term \(\lambda n.\lambda f.\lambda a. n \cdot f \cdot a\). We call the type of this term elimination scheme. By elimination scheme is derivable in Selfstar, we mean the elimination scheme is inhabited by the term \(\lambda n.\lambda f.\lambda a. n \cdot f \cdot a\) in Selfstar (modulo type annotations).

2.1 System Selfstar

We give the full specification of Selfstar in this section. We use gray boxes to highlight certain important terms and rules.

Definition 1 (Syntax).

Terms \(t\) := \(\star \mid x \mid \lambda x. t \mid tt' | \mu t \mid \Pi x : t_1.t_2 \mid \text{case } x \cdot t\)

Closure \(\mu\) := \(\{x_i \mapsto t_i\}_{i \in N}\)

Value \(v\) := \(\star \mid \lambda x. t \mid \Pi x : t_1.t_2 \mid \text{case } x \cdot t \mid \mu(\Pi x : t_1.t_2) \mid \bar{\mu}(x.t)\)

Context \(\Gamma\) := \(\cdot \mid \Gamma, x : t \mid \Gamma, \bar{\mu} \)

Remarks:

- If \(\mu\) is \(\{x_i \mapsto t_i\}_{i \in N}\), then \(\bar{\mu}\) is \(\{(x_i : a_i) \mapsto t_i\}_{i \in N}\) for some term \(a_i\).
- For \(\{x_i \mapsto t_i\}_{i \in N}\), we require for any \(1 \leq i \leq n\), the free variable set \(\text{FV}(t_i) \subseteq \text{dom}(\bar{\mu}) = \{x_1, ..., x_n\}\). We also do not allow any reductions and substitutions inside the closure. We call this the locality restriction. Without locality requirement, it is hard to establish confluence for reductions (see \([1]\)).

- \(\text{FV}(\mu t) = \text{FV}(t) \cup \text{dom}(\mu)\).


\[
\begin{align*}
\frac{}{\Gamma \vdash \text{wf}} & \quad \frac{}{\Gamma, \bar{\mu} \vdash t \vdash \text{wf}} \\
\frac{}{\Gamma \vdash t : *} & \quad \frac{}{\Gamma, x : t \vdash \text{wf}} \\
\end{align*}
\]

Figure 1. Well-formed Context \(\Gamma \vdash \text{wf}\)

\[
\begin{align*}
\frac{}{\Gamma \vdash * : *} & \quad \frac{}{(x : t) \in \Gamma \vdash x : t} \\
\frac{}{(\Gamma, x : t) \vdash t : *} & \quad \frac{x \in \text{FV}(\bar{\mu})}{(\Gamma, \bar{\mu}) \vdash t : *} \\
\frac{}{\Gamma, \bar{\mu} \vdash t \vdash \text{wf}} & \quad \frac{}{\Gamma, \bar{\mu} \vdash t : *} \\
\end{align*}
\]

Self

Figure 2. Typing \(\Gamma \vdash t : t'\)

\[
\begin{align*}
\frac{}{\Gamma \vdash \text{wf}} & \quad \frac{}{\Gamma, \bar{\mu} \vdash t \vdash \text{wf}} \\
\frac{}{\Gamma \vdash * : *} & \quad \frac{}{(x : t) \in \Gamma \vdash x : t} \\
\frac{}{(\Gamma, x : t) \vdash t : *} & \quad \frac{(\Gamma, \bar{\mu}) \vdash t : *}{\Gamma, \bar{\mu} \vdash \text{wf}} \\
\end{align*}
\]

Figure 3. Executions

2.2 Dependently-typed programming with Selfstar

Selfstar uses the self type mechanism to obtain inductive data, resulting in a design that is simpler than most dependently-typed core languages. Intuitively, it is hard imagine how to emulate inductive datatypes and pattern matching without any build-in mechanisms. But as we observe in section \([1]\), Scott encoding together with mutually recursive definitions are enough to perform pattern matching on inductive data. The real difficulty lies in the typing. We want to make sure that both Scott-encoded data and definable operations on these data are typable in Selfstar. The self type allows us to type Scott data and to derive the corresponding elimination schemes. Thus operations on Scott data is typable using the elimination scheme.

2.1 System Selfstar

We give the full specification of Selfstar in this section. We use gray boxes to highlight certain important terms and rules.

Definition 1 (Syntax).

Terms \(t\) := \(\star \mid x \mid \lambda x. t \mid tt' | \mu t \mid \Pi x : t_1.t_2 \mid \text{case } x \cdot t\)

Closure \(\mu\) := \(\{x_i \mapsto t_i\}_{i \in N}\)

Value \(v\) := \(\star \mid \lambda x. t \mid \Pi x : t_1.t_2 \mid \text{case } x \cdot t \mid \mu(\Pi x : t_1.t_2) \mid \bar{\mu}(x.t)\)

Context \(\Gamma\) := \(\cdot \mid \Gamma, x : t \mid \Gamma, \bar{\mu} \)

Remarks:

- If \(\mu\) is \(\{x_i \mapsto t_i\}_{i \in N}\), then \(\bar{\mu}\) is \(\{(x_i : a_i) \mapsto t_i\}_{i \in N}\) for some term \(a_i\).
- For \(\{x_i \mapsto t_i\}_{i \in N}\), we require for any \(1 \leq i \leq n\), the free variable set \(\text{FV}(t_i) \subseteq \text{dom}(\bar{\mu}) = \{x_1, ..., x_n\}\). We also do not allow any reductions and substitutions inside the closure. We call this the locality restriction. Without locality requirement, it is hard to establish confluence for reductions (see \([1]\)).

- \(\text{FV}(\mu t) = \text{FV}(t) \cup \text{dom}(\mu)\).
Γ ⊢ t_1 →* t_2
Γ ⊢ t_1 = t_2

FV(t) # dom(μ)
Γ ⊢ μt = t

Now let us see some concrete examples of Scott encodings in

Definition 2 (Scott’s Derivative). Let μ_s be the following recursive
definitions:
Nat : * ⊢ μ_s : Nat

(t : a_i) ∈ μ_i means (x_i : a_i) ⇒ t_i ∈ μ_i. μ_it denotes μ_1...μ_n t.

Typing does not depend on well-formedness of the context, so the self
type formation rule self is not circular in this sense. We will
show: if Γ ⊢ t : τ then Γ ⊢ t' : τ then Γ ⊢ t' : τ [Appendix A].

We use call-by-value strategy for the execution.

≡ denotes →o U =, where →o denotes the reflexive transitive
and symmetry closure of →o.

The equality rules incorporates execution to automatize a portion
of equality reasoning.

At type level, we want to have the ability to open the closure
when it appear in the context. closure reduction allow us to do this,
without this type level reduction, we can not prove type
preservation.

2.2 Scott Encodings in Selfstar

Now let us see some concrete examples of Scott encodings in
Selfstar. For convenient, we write a → b for Πx : a.b with
x /∈ FV(b).

Definition 3 (Elimination Scheme for Scott’s Derivative).
μ_s ⊢ Rec : IIC : Nat → *.IIn : Nat.(C n) → (C (S n)) → C 0 → IIn : Nat.C n
Rec := μS.λx.C.x n C s z

Typeing: Let Γ ⊢ μ_s, C : Nat → * s : IIn : Nat.(C n) → (C (S n)) ; C 0 n Nat.

Thus n C s z : C n.

Definition 4 (Scott Numerals). Let μ_s be the following recursive
definitions:
Nat : * ⊢ μ_s : Nat

Equality:

μ ∈ Γ
Γ ⊢ μt →o t
Γ ⊢ μt = t
Γ ⊢ μt →o t t'

Figure 4. Equality

Figure 5. Closure Reductions

Remarks:
• (t : a_i) ∈ μ_i means (x_i : a_i) ⇒ t_i ∈ μ_i. μ_it denotes μ_1...μ_n t.

Typing does not depend on well-formedness of the context, so the self
type formation rule self is not circular in this sense. We will
show: if Γ ⊢ t : τ then Γ ⊢ t' : τ then Γ ⊢ t' : τ [Appendix A].

We use call-by-value strategy for the execution.

≡ denotes →o U =, where →o denotes the reflexive transitive
and symmetry closure of →o.

The equality rules incorpates execution to automatize a portion
of equality reasoning.

At type level, we want to have the ability to open the closure
when it appear in the context. closure reduction allow us to do this,
without this type level reduction, we can not prove type
preservation.

(0 : Nat) ⇒ λC.λx.λz.z

Definition 7 (Leibniz Convertability).
Eq := λA.λx.λy.IIC : A → * → C x → C y.

Definition 8. μ_s ⊢ addZ : Πx : Nat.Eq Nat (add x 0) x.
(vec(U, n) : * → 
| x. C : IIp : Nat.vec(U, p) → *) →  
(Πm : Nat.PII : U.Πy : vec(U, m).C(S m) (cons m u y)) →  
(C 0 nil) → (C n x)  
(cons : I n : Nat.U → vec(U, n) → vec(U, S n)) →  
λα.λx.λy.λx.ynvl

(Πn : Nat.PII : U.Πy : vec(U, m).C(S m) (cons m u y)) →  
C 0 nil → C (S n)  
(λC.λy.λx.y n v l) →  
πC.Πn : Nat.PII : U.Πy : vec(U, m).C(S m) (cons m u y)) →  
C 0 nil → C (S n)  
(λC.λy.λx.y n v l).  

So by selfGen, we have λC.λx.λy.x n v l : vec(U, S n). Thus  

Definition 10 (Elimination Scheme for Vector),  
λC.λx.λy.x n v l :  
ΠC.Πn : Nat.PII : U.Πy : vec(U, m).C(S m) (cons m u y)) →  
C 0 nil → C (S n)  
(λC.λy.λx.y n v l).  

By induction on derivation of  

By induction on derivation of  

Solution 3.1. The Analytical System

It is cumbersome to directly prove the equality in Selfstar is Church-Rosser. We develop an analytical system and we prove that the analytical system is equivalent (theorem 15) to the equality system in Selfstar. Then we prove the analytical system is confluent, which implies the Church-Rosser of the equality in Selfstar.

The beta-reductions (figure 6) include definition substitutions and the ordinary beta-reduction in lambda calculus. The μ-reductions (figure 7) are for moving the closure inside the term structure.

Let denote → γ ⊔ → γ. Let e → e denote (→ ⊔ → 1) * . The following lemmas show the relation between → and =.

Lemma 13. If Γ ⊢ t₁ → t₂, then Γ ⊢ t₁ → t₂.

Proof. By induction on derivation of Γ ⊢ t₁ → t₂.
Lemma 14. If $\Gamma \vDash t_1 = t_2$, then $\Gamma \vDash t_1 \leftrightarrow^* t_2$.

Proof. By induction on the derivation of $\Gamma \vDash t_1 = t_2$.

Lemma 15. If $\Gamma \vdash t_1 \leftrightarrow t_2$, then $\Gamma \vdash t_1 = t_2$.

Proof. By induction on the derivation of $\Gamma \vDash t_1 \leftrightarrow t_2$.

The following theorem shows that the analytic system is equivalent to the equality system.

Theorem 16. $\Gamma \vdash t_1 = t_2$ if and only if $\Gamma \vdash t_1 \leftrightarrow^* t_2$.


Suppose $\rightarrow$ is confluent. By theorem\,[16] we know that $\Gamma \vdash \Pi x : t_1.t_2 \rightarrow^* \Pi x : t_1'.t_2'$ implies $\Gamma \vdash \Pi x : t_1.t_2 \rightarrow^* \Pi x : t_1'.t_2'$. The confluence of $\rightarrow$ implies Church-Rosser of $\rightarrow^*$, namely, there exists a $t$ such that $\Gamma \vdash \Pi x : t_1.t_2 \rightarrow^* t$ and $\Gamma \vdash \Pi x : t_1'.t_2' \rightarrow^* t$. By definition of $\rightarrow$, we know $t$ must be of the form $\Pi x : t_1.t_2$, with $\Gamma \vdash t_1 \rightarrow t_3$, $\Gamma \vdash t_1' \rightarrow t_3$, $\Gamma \vdash t_2 \rightarrow t_4$ and $\Gamma \vdash t_2' \rightarrow t_4$. So by lemma\,[15] we have $\Gamma \vdash t_1 = t_1'$ and $\Gamma \vdash t_2 = t_2'$.

Now let us focus on the proof of the confluence of $\rightarrow$. The confluence argument is similar to the one described in\,[3]. We are going to use the following lemma to conclude the confluence of $\rightarrow_\mu \cup \rightarrow_\beta$.

Lemma 17 (Hardin’s interpretation lemma[7]). Let $\rightarrow_1 \cup \rightarrow_2 = \rightarrow_3$, $\rightarrow_1$ being confluent and strongly normalizing. We denote by $\nu(a)$ the $\rightarrow_1$-normal form of $a$. Suppose that there is some relation $\rightarrow_1$ on the $\rightarrow_1$-normal forms satisfying:

$\rightarrow_1 \subseteq \rightarrow_2 \cup \rightarrow_3$ and $a \rightarrow_2 b$ implies $\nu(a) \rightarrow_1 \nu(b)$

Then the confluence of $\rightarrow_1$ implies the confluence of $\rightarrow_2$.

Proof. Suppose $\rightarrow_1$ is confluent. Assume $\rightarrow_1 \subseteq \rightarrow_2 \cup \rightarrow_3$ and $a \rightarrow_2 b$. So by $\rightarrow_1$, $\nu(a) \rightarrow_1 \nu(a')$ and $\nu(a') \rightarrow_1 \nu(a'')$. Note that $\rightarrow_1 \subseteq \rightarrow_2 \cup \rightarrow_3$ implies $\nu(t) = \nu(t')$. By the confluence and strong normalization of $\rightarrow_1$).

By the confluence of $\rightarrow_1$, there exists a $b$ such that $\nu(a') \rightarrow_1 b$ and $\nu(a'') \rightarrow_1 b$. Since $\rightarrow_1 \subseteq \rightarrow_2 \cup \rightarrow_3$, we get $a' \rightarrow_2 \nu(a') \rightarrow_1 b$ and $a'' \rightarrow_2 \nu(a'') \rightarrow_1 b$. Hence $\rightarrow_2$ is confluent.

The idea behinds interpretation method is that it allows us to modulo the $\rightarrow_1$-reduction, we only need to focus on proving the confluence of $\rightarrow_2$. This is essential since in our case, $\rightarrow_\beta \cup \rightarrow_\mu$ can not be directly parallelized, namely, one can not use Tait-Martin Löf’s method (reported in\,[2]) directly to prove the confluence of $\rightarrow_\beta \cup \rightarrow_\mu$, the parallelized version does not enjoy diamond property. With the interpretation method, after modulo the $\rightarrow_\mu$-reduction, we introduce a new reduction $\rightarrow_\beta$ (corresponds to $\rightarrow_2$), we can then use the parallel reduction method to prove confluence of $\rightarrow_\beta$.

Lemma 18. $\rightarrow_\mu$ is confluent and terminating.

So $\rightarrow_\mu$ correspond to $\rightarrow_2$ in the interpretation lemma. Since $\rightarrow_\beta$, is strongly normalizing and confluent, we can define a normalization function which effectively computes the mu-normal form.

Definition 19 ($\mu$-Normal Forms).

$\mu(x) ::= * | x | \mu x.1 | \lambda x.1 | n.n' | \Pi x : n.n' | \lambda x.n$

Note: for the $\mu x_i$ in definition\,[19] we assume $x_i \in \text{dom}(\mu)$.

$\rightarrow^*$ is the reflexive symmetric transitive closure of $\rightarrow$.

Definition 20 ($\mu$ normalization function).

$\mu(\ast) := \ast$
$\mu(x) := x$
$\mu(\lambda y.m(t)) := \lambda y.m(t)$
$\mu(t_1.m(t_2)) := m(t_1)m(t_2)$
$\mu(x.m(t)) := x.m(t)$
$\mu(\Pi x : t.t') := \Pi x : m(t).m(t').$
$\mu(\Pi x : t.t') := \Pi x : m(t).m(t').$
$\mu((\mu y.m(t))) := \mu y.m(t)
$\mu((\Pi x : t.t')) := \Pi x : m(\Pi x : t.t')).$

We shall devise a new notion of reduction on $\mu$-normal form, then show that this reduction is confluent (corresponds to $\rightarrow_1$ in the interpretation lemma and satisfying the $\uparrow$ property), thus by the interpretation lemma, we can show $\rightarrow_\beta \cup \rightarrow_\mu$ is confluent. A natural way to define reduction on $\mu$-normal form is that right after a beta-reduction, one immediately mu-normalizes the contractum, which can form a notion of reduction on $\mu$-normal form.

Definition 21 ($\beta$ Reduction on $\mu$-Normal Forms).

$\Gamma \vdash n \rightarrow_\beta t$
$\Gamma \vdash n \rightarrow_\beta \mu m(t)$

The following lemma shows that $\rightarrow_\beta \mu$ corresponds to $\rightarrow_1$ in the interpretation lemma.

Lemma 22. If $\Gamma \vdash t \rightarrow_\beta t'$, then $\Gamma \vdash m(t) \rightarrow_\beta \mu m(t').$

Lemma 23. $\rightarrow_\beta \mu$ is confluent.

Theorem 24. $\rightarrow_\beta \mu$ is confluent.

Proof. We know $\rightarrow_\beta \mu$ is confluent. Since $\rightarrow_\mu$ is strongly normalizing and confluent, by lemma\,[22] and Hardin’s interpretation lemma\,[17], we conclude $\rightarrow_\beta \mu$ is confluent.

3.2 Confluence Analysis

Definition 25. $\Gamma \vdash t_1 \rightarrow_\beta t_2$ if $t_1 \equiv \Pi x.t' \equiv [t/x]t'$ for some fix term $t$.

Note that $\rightarrow_\beta$, models the selfInst rule, $\rightarrow_\beta^{-1}$ models the selfGen rule. The notion of $\iota$-reduction does not build in structure congruence, namely, we do not allow reduction rules like: if $T \rightarrow_\beta T'$, then $\lambda x.T \rightarrow_\beta \lambda x.T'$. The purpose of $\iota$-reduction is to emulate the typing rule selfInst and selfGen. This rewriting point of view on typing is inspired by Kuan et.al.\,[10] and Stump et.al.\,[12].

Lemma 26 (Confluence). $\rightarrow_\beta$ is confluent.

Proof. This is obvious since $\rightarrow_\beta$, is deterministic.

The goal of this section is to show $\rightarrow_0 \cup \beta \mu$ is confluent. We make extensive use of the notion of commutativity, which provides a simple way to prove the confluence of a reduction system that has several confluent subreductions.

Definition 27 (Commutativity). Let $\rightarrow_1$, $\rightarrow_2$ be two notions of reduction, $\rightarrow_1$ (strongly) commute with $\rightarrow_2$ if $a \rightarrow_1 b_1$ and $a \rightarrow_2 b_2$, then there exists a $c$ such that $b_1 \rightarrow_2 c$ and $b_2 \rightarrow_1 c$.

Proposition 28 (Hindley-Rosen\,[9] \[11\]). Let $\rightarrow_1$, $\rightarrow_2$ be two notions of reduction. Suppose both $\rightarrow_1$ and $\rightarrow_2$ are confluent, and $\rightarrow_1^\uparrow$ commutes with $\rightarrow_2^\uparrow$. Then $\rightarrow_1 \cup \rightarrow_2$ is confluent.

$\rightarrow_\beta \mu$ denotes $\rightarrow_\beta \cup \rightarrow_\mu$, we will use this convention throughout the paper.
Proposition 29 (Weak Commutativity [2]). Let $\leftrightarrow$ denote the reflexive closure of $\rightarrow$. Let $\rightarrow_1, \rightarrow_2$ be two notions of reduction. $\rightarrow_1$ weak commutes with $\rightarrow_2$ if $a \rightarrow_1 b_1$ and $a \rightarrow_2 b_2$, then there exists a $c$ such that $b_1 \leftrightarrow c$ and $b_2 \rightarrow_1 c$.

If $\rightarrow_1$ weak commutes with $\rightarrow_2$, then $\rightarrow_1^\ast$ and $\rightarrow_2^\ast$ commute.

Lemma 30. $\beta, \mu$ commutes with $\rightarrow_\ast$. Thus $\beta, \mu, o, \beta, \mu$ is confluent.

Lemma 31. $\rightarrow_\ast$ has diamond property, thus is confluent.

Lemma 32. $\rightarrow_\ast$ commutes with $\rightarrow_o$, weak commutes with $\rightarrow_o$, $\rightarrow_o$. $\rightarrow_o, \rightarrow_o$ is confluent.

Theorem 33. $\beta, \mu, o, \beta, \mu$ is confluent.

Let $=_{\beta, \mu, o, \beta, \mu}$ denotes the reflexive transitive symmetric closure of $\rightarrow_o \cup \rightarrow_o \cup \rightarrow_\beta \cup \rightarrow_o$. The goal of confluence analysis is to establish the following theorem.

Theorem 34 ($\alpha$-elimination, Compatibility).

If $\Gamma \vdash \Pi x : t_1 t_2 =_{\beta, \mu, o, \beta, \mu} \Pi x : t_1' t_2'$, then $\Gamma \vdash t_1 =_{\beta, \mu, o} t_1'$ and $\Gamma \vdash t_2 =_{\beta, \mu, o} t_2'$.

Proof: If $\Gamma \vdash \Pi x : t_1 t_2 =_{\beta, \mu, o, \beta, \mu} \Pi x : t_1' t_2'$, then by the confluence of $=_{\beta, \mu, o, \beta, \mu}$, there exists a $\tau$ such that $\Gamma \vdash \Pi x : (\beta, \mu, o, \beta, \mu) \tau t_1 t_2$ and $\Gamma \vdash \Pi x : (\beta, \mu, o, \beta, \mu) \tau t_1' t_2'$. Since all the reductions on $\Pi x : t_1 t_2$ preserve the structure of the dependent type, one will never have a chance to use $\rightarrow_o$-reduction, thus $\Gamma \vdash \Pi x : t_1 t_2 (\beta, \mu, o, \beta, \mu) \tau t_1$ and $\Gamma \vdash \Pi x : t_1' t_2' =_{\beta, \mu, o, \beta, \mu} t_1$. $\rightarrow_o$-s rule, we $\rightarrow_o$-s the $\rightarrow_o$-s of $t_1 t_2$.

3.3 Type Preservation

The proof of type preservation proceeds as usual. The inversion lemma and substitution lemma are standard. Note that in the final preservation proof, we use the compatibility theorem.

Lemma 35 (Inversion).

- If $\Gamma \vdash \lambda x. t : t'$, then $\Gamma, x : t_1 \vdash t : t_2$ and $\Gamma \vdash \Pi x : t_1 t_2 =_{\beta, \mu, o, \beta, \mu} o$ for some $t_1, t_2$.
- If $\Gamma \vdash t_1 t_2 : t'$, then $\Gamma \vdash t_1 : \Pi x : t_1' t_2'$ and $\Gamma \vdash t_2 : t_2'$, $\Gamma \vdash [t_2/x] =_{\beta, \mu, o, \beta, \mu} o$ for some $t_1', t_2'$.
- If $\Gamma \vdash x : t$, then $\Gamma \vdash x : t \rightarrow_o \Gamma \vdash x : t$ for some $t$.

Lemma 36 (Substitution). If $\Gamma, x : t_1, \Gamma_2 \vdash t : t_2$ and $\Gamma \vdash t' : t_1$, then $\Gamma_1, [t'/x] \Gamma_2 \vdash [t'/x] t : [t'/x] t'$.

Theorem 37 (Type Preservation). If $\Gamma \vdash w f$ and $\Gamma \vdash t : t'$ and $\Gamma \vdash t : t'$, then $\Gamma \vdash t : t'$.

Proof: We list one interesting case here.

$\Gamma \vdash t_1 : \Pi x : t''_1 t''_2 \Rightarrow \Gamma \vdash t_1' : [t_1'/x] t''_2$

Suppose $\Gamma \vdash (\lambda x. t_1) v \rightarrow \Pi x : v [t_1'/x] t''_2$. Then we know $\Gamma \vdash (\lambda x. t_1) v : [t_1'/x] t''_2$ and $\Gamma \vdash \lambda x. t_1 : \Pi x : t''_1 t''_2$ and $\Gamma \vdash v : t'_1$. By inversion on $\Gamma \vdash \lambda x. t_1 : \Pi x : t''_1 t''_2$, we have $\Gamma, x : a \vdash t_1 : b$ and $\Gamma \vdash \Pi x : a. b =_{\beta, \mu, o, \beta, \mu} o$ for some $t_1', t_2'$. By $\beta, \mu, o, \beta, \mu$-s, we have $\Gamma \vdash a =_{\beta, \mu, o} t_1'$ and $\Gamma \vdash b =_{\beta, \mu, o} t_2'$. So we have $\Gamma, x : a \vdash t_1 : t_2'$ and $\Gamma \vdash v : a$. So by lemma 79, we know $\Gamma \vdash [v/x] t : [v/x] t''_2$, as required.

4. Conclusion

We introduce Selfstar, which incorporates the self type construct together with $* : *$ and mutually recursion. Scott-encoded datatypes and the corresponding elimination schemes are derivable within Selfstar. We also demonstrate the process of proving the type preservation theorem.

References


A. Well-Form Type

Lemma 38. If $\Gamma \vdash w f$ and $\Gamma \vdash t : t'$, then $\Gamma \vdash t' : *$.

Proof. By induction on derivation of $\Gamma \vdash t : t'$. We list a few nontrivial cases.

Case:

$\Gamma \vdash t : [t/x] t'$

SelfInst

By IH, we have $\Gamma \vdash t : [t/x] t' : *$. So by inversion, we have $\Gamma, x : [t/x] t' : *$. So by lemma 79, we know $\Gamma \vdash [t/x] t' : *$.

Case:

$\Gamma, x : t_1 t_2 : t_1 \vdash t_2 : \ast$ Lam

By IH, we know $\Gamma, x : t_1 t_2 : *$. Since $\Gamma \vdash t_1 : *$, by $\Pi$ rule, we have $\Gamma \vdash \Pi x : t_1 t_2 : *$.

Case:
\[ \Gamma \vdash t : \Pi x : t_1.t_2 \quad \Gamma \vdash t': t_1 \quad \text{App} \]

By IH, we have \( \Gamma \vdash \Pi x : t_1.t_2 : * \). By inversion on \( \Gamma \vdash \Pi x : t_1.t_2 : * \), we have \( \Gamma, x : t \vdash t_2 : * \). So by lemma \( 79 \), we have \( \Gamma \vdash [t'/x]t_2 : * \).

Case:

\[ \Gamma, \mu \vdash t : t' \quad \{ \Gamma, \mu \vdash t : a_j \}_{(t_j.a_j) \in \bar{a}} \]

\[ \Gamma \vdash \mu t : \mu t' \quad \text{Mu} \]

By IH, we have \( \Gamma, \mu \vdash t' : * \). So \( \Gamma \vdash \mu t' : \mu * \), thus \( \Gamma \vdash \mu t' : * \).

\[ \Box \]

### B. Progress

**Lemma 39.** If \( \Gamma \vdash \Pi x : tt_1.t_2 \), then \( \forall \lambda x.t \).

**Proof.** Case analysis on \( \forall \). Suppose \( \forall \equiv \ast \). By inversion, \( \vdash \ast : \ast \) and \( \vdash \ast : \beta \ast \ast \) or \( \Pi x : t_1.t_2 \), which contradicts Church-Rosser of \( =_{\beta,\mu,\iota,o} \). Suppose \( \forall \equiv \overline{\mu} (\Pi x : t_3.t_4) \). By inversion, we have \( \overline{\mu} \vdash \Pi x : t_3.t_4 \) and \( \vdash \bar{\mu} t_a : \overline{\mu} (\Pi x : t_3.t_4) \equiv \beta_{\beta,\mu,\iota,o} \Pi x : t_1.t_2 \). By inversion on \( \overline{\mu} \vdash \Pi x : t_3.t_4 : t_a \), we have \( \mu \equiv \ast \equiv \beta_{\beta,\mu,\iota,o} t_a \).

So we have \( \vdash \bar{\mu} t_{a} \equiv \beta_{\beta,\mu,\iota,o} \overline{\mu} \bar{\mu} t_a \equiv \beta_{\beta,\mu,\iota,o} \Pi x : t_1.t_2 \). Again, this contradicts Church-Rosser of \( =_{\beta,\mu,\iota,o} \). For other cases like: \( \forall \equiv \Pi x : t.t' \), \( \forall t \), \( \bar{\mu} \), we argue similarly.

\[ \Box \]

**Theorem 40 (Progress).** If \( \vdash t : t' \), then either \( \vdash t \rightsquigarrow t' \) or \( t \) is a value.

**Proof.** By induction on the derivation of \( \vdash t : t' \), we list a few cases.

Case:

\[ \mu \vdash t : t' \quad \{ \mu \vdash t_j : a_j \}_{(t_j.a_j) \in \bar{a}} \]

\[ \vdash \mu t : \mu t' \quad \text{Mu} \]

Identify \( t \) as \( \mu t' \), where \( t' \) does not contain any closure at head position. Case analysis on \( t' \), if it is \( \ast, x, \lambda x.t, t_a.t_b \), then there exist a \( t' \) such that \( \vdash t \rightsquigarrow t' \). If \( t'' \equiv \Pi x : t_a.t_b, \text{\space} \forall t, \text{\space} \text{\space} x, \lambda x.t \), then it is already a value.

Case:

\[ \vdash \Pi x : t_1.t_2 \quad \vdash t : t_1 \quad \text{App} \]

\[ \vdash tt' : [t'/x]t_2 \]

Since \( \vdash \Pi x : t_1.t_2 \) and \( \vdash t : t_1 \), by IH, \( t \) either steps or is a value, likewise for \( t' \). If \( t \) can take a step, then \( tt' \) can also take a step. If \( t \) is a value, then \( tt' \) also takes a step. If both \( t \) is a value, then \( tt' \) can take a step.

\[ \Box \]

### C. Proofs of Section 3.1

Let \( \hat{\mu} \) denote \( 0 \) or more closures.

**Lemma 41.** Let \( \Phi \) denote the set of \( \mu \) normal form. For any term \( t, \mathcal{M}(t) \in \Phi \).

**Proof.** One way to prove this is first identify \( t \) as \( \hat{\mu} t' \), here \( \hat{\mu} \) means there are zero or more closures and \( t' \) does not contain any closure at head position. Then we can proceed by induction on the structure of \( t' \).

**Base Cases:** \( t' = x, t' = \ast \), obvious.

**Step Cases:** If \( t' = \lambda x.t'' \), then \( \mathcal{M} (\hat{\mu} (\lambda x.t'')) \equiv \lambda x.m (\hat{\mu} t'') \).

Now we can again identify \( t'' \) as \( \hat{\mu} t'' \), where \( t'' \) does not have any closure at head position. Since \( t'' \) is structurally smaller than \( \lambda x.t'' \), by IH, \( m (\hat{\mu} t'') \in \Phi \), thus \( \mathcal{M} (\hat{\mu} (\lambda x.t'')) \equiv \lambda x.m (\hat{\mu} t'') \in \Phi \).

For \( t' = t_a.t_b, t' = \lambda x.t'', t' = t_a.t_b, t' = \lambda x.t, t \), we can argue similarly as above.

In order to prove lemma \( \ref{lemma} \) we prove the following more general lemma instead.

**Lemma 42.** If \( \Gamma, \mu \vdash a \rightarrow_{\beta} b \), then \( \Gamma \vdash m (\hat{\mu} a) \rightarrow_{\beta} m (\hat{\mu} b) \).

**Proof.** By induction on derivation of \( \Gamma, \mu \vdash a \rightarrow_{\beta} b \).

**Base Case:**

\[ (x \rightarrow t) \in \Gamma, \mu \]

\[ \mu \vdash x \rightarrow_{\beta} t \]

If \( x \rightarrow t \in \mu \), then \( \Gamma \vdash m (\hat{\mu} x) \equiv \mu x \rightarrow_{\beta} m (\mu t) \equiv m (\hat{\mu} t) \).

Technically, the last equality need to be justified, informally we can justify that by locality of \( \mu \). If \( x \rightarrow t \in \Gamma \), then \( \Gamma \vdash m (\hat{\mu} x) \equiv x \rightarrow_{\beta} m (\mu t) \equiv m (\hat{\mu} t) \).

**Base Case:**

\[ \mu \vdash (\lambda x.t) \rightarrow_{\beta} [t'/x]t \]

We have \( \Gamma \vdash (\hat{\mu} (\lambda x.t')) \equiv (\lambda x.m (\hat{\mu} t')) \rightarrow_{\beta} m (\hat{\mu} t') \equiv m (\hat{\mu} t') \equiv m (\hat{\mu} t') \equiv m (\hat{\mu} (\lambda x.t')). \]

The last two equalities are by lemma \( \ref{lemma} \), \( \ref{lemma} \).

**Step Case:**

\[ \Gamma, \mu \vdash t \rightarrow_{\beta} t' \]

\[ \Gamma \vdash \lambda x.t \rightarrow_{\beta} \lambda x.t' \]

\[ \Gamma \vdash m (\hat{\mu} (\lambda x.t)) \equiv \lambda x.m (\hat{\mu} t') \rightarrow_{\beta} m (\hat{\mu} t') \equiv m (\hat{\mu} (\lambda x.t')) \]

**Step Case:**

\[ \Gamma, \mu \vdash t \rightarrow_{\beta} t' \]

\[ \Gamma \vdash \mu t \rightarrow_{\beta} \mu t' \]

We want to show \( \Gamma \vdash m (\hat{\mu} t) \rightarrow_{\beta} m (\hat{\mu} t') \). This is directly by IH.

All the other cases are similar.

\[ \Box \]

**Lemma 43.**

\[ m (\hat{\mu} t) \equiv m (\hat{\mu} t) \]
Proof. We can prove this using the same method as lemma 45, namely, identify $t$ and then proceed by induction.

Lemma 44. $m(m(t)) \equiv m(t)$ and $m([m(t_1)/y]m(t_2)) \equiv m([t_1]/y)t_2$.

Proof. The first equality is by lemma 45 and lemma 41. For the second equality, we prove it using similar method as lemma 41. We identify $t_2$ as $\tilde{\mu}t_2$, where $t_2$ does not contains any closure at head position. We proceed by induction on the structure of $t_2^*$:

Base Cases: $t_2^* = \ast$, obvious. For $t_2 = x$, we use $m(t) \equiv m(t)$.

Step Cases: If $t_2 = \lambda x.t_2'$, then $m(\tilde{\mu}^1(t_2')t_2') \equiv m(\tilde{\mu}^1([t_1]/y)t_2') \equiv \lambda x.m(\tilde{\mu}^1([t_1]/y)t_2')$, and $t_2'$ is identified as $\tilde{\mu}^2t_2''$, and $t_2''$ does not have any closure at head position. Since $t_2''$ is structurally smaller than $\lambda x.t_2'$, by IH, $m(\tilde{\mu}^2(t_1')t_2'') \equiv m([t_1]/y)\tilde{\mu}^2(t_1't_2'')$.

Thus $\lambda x.m(\tilde{\mu}^2([t_1]/y)t_2'') \equiv \lambda x.m([m(t_1)/y]m(\tilde{\mu}^2(t_1't_2''))) \equiv m(m([t_1]/y)m(\tilde{\mu}^2(t_1't_2''))) \equiv m([t_1]/y)m(\tilde{\mu}^2(t_1't_2''))$.

Lemma 45. If $n \in \Phi$, then $m(n) \equiv n$.

Proof. By induction on the structure of $n$.

Definition 46 ($\beta$ Reduction on $\mu$-normal Forms).

$\Gamma \vdash n \rightarrow \beta \mu \ m(t)$

Note: From this definition we can conclude:

$\Gamma \vdash n \rightarrow \beta \mu \ n'$

$\Gamma \vdash \lambda x. n \rightarrow \beta \mu \ \lambda x. n'$

$\Gamma \vdash n' \rightarrow \beta \mu \ n''$

$\Gamma \vdash \Pi x : n.n' \rightarrow \beta \mu \ \Pi x : n.n''$

$\Gamma \vdash n \rightarrow \beta \mu \ n''$

$\Gamma \vdash \lambda x.n \rightarrow \beta \mu \ \lambda x.n'$

$\Gamma \vdash \Pi x : n.n' \rightarrow \beta \mu \ \Pi x : n''n''$

$\Gamma \vdash n \rightarrow \beta \mu \ n''$

$\Gamma \vdash \lambda x.n \rightarrow \beta \mu \ \lambda x.n'$

$\Gamma \vdash \Pi x : n.n' \rightarrow \beta \mu \ \Pi x : n''n''$

The first rule follows because: Assume $\Gamma \vdash n \rightarrow \beta \mu \ n'$, say $m(t) \equiv n'$ and $\Gamma \vdash n \rightarrow \beta \mu \ t$. Then $\Gamma \vdash \lambda x.n \rightarrow \beta \mu \ \lambda x.t$ and $m(\lambda x.t) \equiv \lambda x.m(t) \equiv \lambda x.n'$. The others follow similarly.

Lemma 47. If $\Gamma \vdash n_1 \rightarrow \beta \mu \ n_1'$, then $\Gamma \vdash m([n_2/x]n_1) \rightarrow \beta \mu \ m([n_2/x]n_1')$.

Proof. By induction on derivation of $\Gamma \vdash n_1 \rightarrow \beta \mu \ t_1$, where $m(t_1) \equiv n_1'$. We will list a few nontrivial cases. Note that the we use lemma 45 implicitly.

Base Case:

$\ast \rightarrow t_1 \in \Gamma$

$\Gamma \vdash \ast \rightarrow \beta \ t_1$

In this case $n_1 = \ast$. By locality, we have $\Gamma \vdash m([n_2/x]y) \equiv y \rightarrow \beta \mu \ m(t_1) \equiv m([n_2/x]t_1)$.

Base Case:

$\Gamma \vdash \lambda y.n \rightarrow \beta \ n'$

$n_1 = \lambda y.n$. So $\Gamma \vdash m([n_2/x]n_1) \equiv m([\lambda y.[n_2/x]n_1]n_1) \equiv ([\lambda y.m([n_2/x]n_1)m([n_2/x]n_1'] \rightarrow \beta \mu \ m([n_2/x]n_1')/y)m([n_2/x]n_1) \equiv m([n_2/x]n_1'/y)m([n_2/x]n_1) \equiv m([n_2/x]n_1'/y/m(n_1))$.

Base Case:

$\Gamma \vdash \mu x \rightarrow \beta \ t_1$

$\Gamma \vdash \mu x \rightarrow \beta \ t_1$

This case will not arise since $n_1$ is already in $\mu$ normal form.

The other cases are similar.

Lemma 48. If $\Gamma \vdash n_2 \rightarrow \beta \mu \ n_2'$, then $\Gamma \vdash m([n_2/x]n_1) \rightarrow \beta \mu \ m([n_2/x]n_1')$.

Proof. By induction on $n_1$.

Definition 49 (Parallel Reductions).

$\Gamma \vdash (x \rightarrow t) \in \Gamma$

$\Gamma \vdash x \rightarrow \beta \mu \ n$

$\Gamma \vdash n_1 \rightarrow \beta \mu \ n_1'$

$\Gamma \vdash n_2 \rightarrow \beta \mu \ n_2'$

$\Gamma \vdash (\lambda x.n_1)n_2 \rightarrow \beta \mu \ m([n_2/x]n_1')$

$\Gamma \vdash n \rightarrow \beta \mu \ n''$

$\Gamma \vdash n \rightarrow \beta \mu \ n''$

$\Gamma \vdash m' \rightarrow \beta \mu \ n''$n''

$\Gamma \vdash \Pi x : n.n' \rightarrow \beta \mu \ \Pi x : n''n''$

Lemma 50. $\rightarrow \beta \mu \subseteq \rightarrow \beta \mu \subseteq \rightarrow \beta \mu$.

Proof. For $\rightarrow \beta \mu \subseteq \rightarrow \beta \mu$, by induction on the derivation of $\Gamma \vdash n \rightarrow \beta \mu \ t$, where $\Gamma \vdash n \rightarrow \beta \mu \ t$. For $\rightarrow \beta \mu \subseteq \rightarrow \beta \mu$, by induction on the derivation of $\Gamma \vdash n \rightarrow \beta \mu \ n'$. We show the case where (the other cases are obvious):

$\Gamma \vdash n_1 \rightarrow \beta \mu \ n_1'$

$\Gamma \vdash n_2 \rightarrow \beta \mu \ n_2'$

$\Gamma \vdash (\lambda x.n_1)n_2 \rightarrow \beta \mu \ m([n_2/x]n_1')$

By lemma 52, we know that $\Gamma \vdash m([n_2/x]n_1) \rightarrow \beta \mu \ m([n_2/x]n_1)$, given $\Gamma \vdash n_1 \rightarrow \beta \mu \ n_1'$, $\Gamma \vdash n_2 \rightarrow \beta \mu \ n_2'$. Since $\rightarrow \beta \mu \subseteq \rightarrow \beta \mu$,
we have: if \( \Gamma \vdash n_1 \to_{\beta} n_1' \), then \( \Gamma \vdash m(n_2/x) \to_{\beta} m(n_2/x)n_1' \). By IH, we have \( \Gamma \vdash n_1 \to_{\beta} n_1' \) and \( \Gamma \vdash n_2 \to_{\beta} n_2' \). By lemma [47], lemma [48] and (f), we have \( \Gamma \vdash (\lambda x. n_1)n_2 \to_{\beta} m(n_2/x)n_1 \to_{\beta} m(n_2/x)n_1' \).

\[ \Gamma \vdash n \to_{\beta} n' \]
\[ \Gamma \vdash \lambda x. n \to_{\beta} \lambda x. n' \]

We have \( \Gamma \vdash m(\lambda x. [n_2/y]n) \equiv \lambda x. m([n_2/y]n) \to_{\beta} \lambda x. m([n_2/y]n') \equiv m(\lambda x. [n_2/y]n') \)

**Step Case:**

\[ \Gamma \vdash n_a \to_{\beta} n'_a \quad \Gamma \vdash n_b \to_{\beta} n'_b \]

\[ \Gamma \vdash n_a n_b \to_{\beta} n'_a n'_b \]

We have \( \Gamma \vdash m([n_2/y]n_a)[n_2/y]n_b \equiv m([n_2/y]n_a)(\lambda x. m([n_2/y]n) \to_{\beta} m([n_2/y]n')(\lambda x. m([n_2/y]n'))) \equiv m(([n_2/y]n')(\lambda x. m([n_2/y]n'))) \)

**Theorem 54.** \( \to_{\beta} \cup \to_{\beta} \) is confluent.

**Proof.** We know by diamond property of \( \to_{\beta} \cup \to_{\beta} \) is confluent. Since \( \to_{\beta} \) is strongly normalizing and confluent, and by lemma [44].
and Hardin’s interpretation lemma [17], we conclude $\rightarrow_{\beta} \cup \rightarrow_{\mu}$ is confluent.

D. Proofs of Section 3.2

Lemma 55. Let $\rightarrow$ denote $\rightarrow_{\beta} \cup \rightarrow_{\mu}$, if $\Gamma \vdash t \rightarrow t'$, then $\Gamma \vdash [t/x]t \rightarrow [t/x]t'$ for any $t_1$.

Proof. Obvious.

Lemma 56. Let $\rightarrow$ denote $\rightarrow_{\beta} \cup \rightarrow_{\mu}$, then $\rightarrow$ commutes with $\rightarrow_{\beta}$, i.e. if $\Gamma \vdash t_1 \rightarrow t_2$ and $\Gamma \vdash t_3 \rightarrow t_4$, then there exist $t_4$ such that $\Gamma \vdash t_2 \rightarrow t_4$ and $\Gamma \vdash t_1 \rightarrow t_4$.

Proof. Since $\Gamma \vdash t_1 \rightarrow t_3$, we know that $t_1 \equiv \lambda x.t' \rightarrow t_3 \equiv [t/x]t''$. We also have $\Gamma \vdash t_2 \equiv \lambda x.t'' \rightarrow t_2$. By inversion, we know that $t_2 \equiv \lambda x.t'' \rightarrow t'$ and $\Gamma \vdash t_1 \rightarrow t''$. By lemma 55, we know that $\Gamma \vdash [t/x]t'' \rightarrow [t/x]t''$. Thus $t_4 \equiv [t/x]t''$ and $\Gamma \vdash \lambda x.t'' \rightarrow [t/x]t''$.

Theorem 57. $\rightarrow_{\beta} \cup \rightarrow_{\mu}$ is confluent.

Lemma 58. If $\Gamma \vdash t_1 \rightarrow_o t_2$, then $\Gamma \vdash [t/x]t_1 \rightarrow_o [t/x]t_2$.

Proof. By induction on derivaton.

Lemma 59. If $\Gamma \vdash t_1 \rightarrow_o t_2$, then $\Gamma \vdash [t_1/x]t \rightarrow_o [t_2/x]t$.

Proof. By induction on the structure of $t$.

Lemma 60. $\rightarrow_o$ has diamond property, thus is confluent.

Proof. Straightforward induction.

Lemma 61. $\rightarrow_o$ commutes with $\rightarrow_{\beta}$.

Proof. Suppose $\Gamma \vdash \lambda x.t \rightarrow [t/x]t'$ and $\Gamma \vdash \lambda x.t \rightarrow [t/x]t''$ with $\Gamma \vdash t' \rightarrow t''$. Then by lemma 58, we have $\Gamma \vdash [t/x]t' \rightarrow_o [t/x]t''$, i.e. we also have $\Gamma \vdash [t/x]t' \rightarrow_o [t/x]t''$.

Lemma 62. $\rightarrow_o$ weak commutes with $\rightarrow_{\beta}$.

Proof. By induction on $\rightarrow_o$.

Case: $\Gamma \vdash \mu t \rightarrow_o t$, where $\mu \in \Gamma$.

If $\Gamma \vdash \mu \xi \rightarrow_\beta \mu t$, where $\xi \rightarrow_\beta \mu t$.

So we have $\Gamma \vdash \mu \xi \rightarrow_\beta \mu t$ and $\Gamma \vdash \xi \rightarrow_\beta t$. By lemma 59, we know that $\Gamma \vdash [t_2/x]t_1 \rightarrow_o [t_2/x]t_1$. And we also have $\Gamma \vdash [\lambda x.\xi]t_1 \rightarrow_\beta [\lambda x.t_1]t_1$.

Case: $\Gamma \vdash (\lambda x.\xi)t_2 \rightarrow_o (\lambda x.\xi)t_2$, where $\Gamma \vdash t_2 \rightarrow_o t_2$.

Suppose $\Gamma \vdash (\lambda x.\xi)t_2 \rightarrow_\beta (\lambda x.\xi)t_2$. By lemma 58, we know that $\Gamma \vdash [t_2/x]t_1 \rightarrow_o [t_2/x]t_1$. And we also have $\Gamma \vdash (\lambda x.\xi)t_1 \rightarrow_\beta [\lambda x.t_1]t_1$.

The other cases are by induction.

Lemma 63. $\rightarrow_o$ weak commutes with $\rightarrow_{\mu}$, i.e. if $\Gamma \vdash t \rightarrow_o t'$ and $\Gamma \vdash t \rightarrow_{\mu} t'$, then there exist a $t_2$ such that $\Gamma \vdash t' \rightarrow_\beta t_2$.

Proof. By induction on $\rightarrow_{\beta}$.

Case: $\Gamma \vdash \mu t \rightarrow_{\mu} t$, where $\mu \in \Gamma$.

Suppose $\Gamma \vdash \mu t \rightarrow_{\mu} t$ with dom(\mu)\#VF(t). This case is obvious.

Suppose $t \equiv \lambda x.t_2$ and $\Gamma \vdash \mu (\lambda x.t_2) \rightarrow_{\mu} \lambda x.\mu t_2$. Then $\Gamma \vdash \lambda x.t_2 \rightarrow_{\mu} \lambda x.\mu t_2$ and $\Gamma \vdash \lambda x.t_2 \rightarrow_\beta \lambda x.t_2$.

Suppose $t \equiv t_2t_3$ and $\Gamma \vdash (t_2t_3) \rightarrow_{\mu} (t_2t_3)(t_3t_3)$. Then $\Gamma \vdash t_2t_3 \rightarrow_\beta t_2t_3$ and $\Gamma \vdash (t_2t_3)(t_3t_3) \rightarrow_\beta t_2t_3$.

We then argue similarly.

The other cases are by induction.

Theorem 64. $\rightarrow_o \cup \rightarrow_{\beta} \cup \rightarrow_{\mu}$ is confluent.

E. Proofs of Section 3.3

Note: In this section we use $\equiv_{\beta,\mu,i,o}$ to mean the same thing as $\equiv_{\beta,\mu,i,o}$, but with an emphasis on the subject $t$.

Lemma 65. If $\Gamma \vdash t_1 \equiv_{\beta,\mu,i,o} t_2$ and $\Gamma \vdash t \rightarrow : s$, then $\Gamma \vdash t \rightarrow : t_2$.

Proof. By induction on length of $\Gamma \vdash t_1 \equiv_{\beta,\mu,i,o} t_2$.

Lemma 66. If $\Gamma \vdash t_1 \equiv_{\beta,\mu,i,o} t_2$ and $\Gamma \vdash t \rightarrow : t'$, then $\Gamma \vdash t_1 \equiv_{\beta,\mu,i,o} t_2$.

Proof. By induction on length of $\Gamma \vdash t_1 \equiv_{\beta,\mu,i,o} t_2$.

Lemma 67. $m(\mu_1\mu_2t) \equiv m(\mu_2\mu_1t)$, thus $\Gamma \vdash \mu_1\mu_2t \equiv \mu_2\mu_1t$.

Proof. Identify $t$ as $\tilde{t}'$, where $t'$ does not have any closure at heap position. By induction on the structure of $t'$. Also $\Gamma \vdash \mu_1\mu_2t = m(\mu_2\mu_1t) = m(\mu_2\mu_1t)$.

Lemma 68. $\Gamma \vdash \mu([t/x]t') = [\mu[t/x]t']t'$.

Proof. By induction on length of $\Gamma \vdash \mu([t/x]t') = [\mu[t/x]t']t'$.

Lemma 69. If $\tilde{t_1} \rightarrow : t_2$, then $\Gamma \vdash t_1 \rightarrow : t_2$. We list a few cases.

Case: $\Gamma \vdash : t_1 \rightarrow : t_2$.

We have $\Gamma \vdash : t_1 \rightarrow : t_2$.

Case: $\Gamma \vdash : t_1 \rightarrow : t_2$.

We know $\Gamma \vdash \mu[t/x]t_1 \rightarrow : \mu[t/x]t_1 \equiv \mu[t/x]t_1$ (the last equality is by lemma 68).

Lemma 70 (Inversion I). $\Gamma \vdash \lambda x.t \rightarrow : t_2$, then $\Gamma \vdash \lambda x.t : t_2$ and $\Gamma \vdash \Pi t : t_1.t_2 \equiv_{\beta,\mu,i,o} t_2$.

Proof. By induction on the derivation of $\Gamma \vdash \lambda x.t \rightarrow : t'$.

Lemma 71 (Inversion II). $\Gamma \vdash t_1.t_2 : t_2$, then $\Gamma \vdash : t_1.t_2 : t_2$ and $\Gamma \vdash : t_1.t_2 : t_2 \equiv_{\beta,\mu,i,o} t_2$.

Proof. By induction on the derivation of $\Gamma \vdash : t_1.t_2 : t'$.
Lemma 72 (Inversion III). If $\Gamma \vdash x : a$, then $\Gamma \vdash * \equiv_{\beta,\mu,\iota,o} t$.

Lemma 73 (Inversion IV). If $\Gamma \vdash x : t'$, then $x : t \in \Gamma$ and $\Gamma \vdash t = \equiv_{\beta,\mu,\iota,o} t'$.

Lemma 74 (Inversion V). If $\Gamma,\mu \vdash x_j : t'$ and $x_j \in \text{dom}(\mu)$, then $x_j : a_j \in \mu$ and $\Gamma,\mu \vdash a_j = \equiv_{\beta,\mu,\iota,o} t'$.

Lemma 75 (Inversion VI). If $\Gamma \vdash \mu t : t$ and $t$ does not have a closure at head position, then $\Gamma,\mu \vdash t : t'$ and $\Gamma \vdash \mu t' = \equiv_{\beta,\mu,\iota,o} t'$.

Lemma 76 (Inversion VII). If $\Gamma \vdash \xi x.t : t'$, then $x : \xi x.t \vdash t : *$ and $\Gamma \vdash * = \equiv_{\beta,\mu,\iota,o} t'$.

Lemma 77 (Inversion VIII). If $\Gamma \vdash \Pi x : t_1.t_2 : t'$, then $\Gamma,\mu \vdash t_1 : *$ and $\Gamma \vdash t_1 : t_2 = \equiv_{\beta,\mu,\iota,o} t'$.

Lemma 78. If $\Gamma,\mu,\iota,\mu \vdash b : a$, then $\Gamma,\iota,\mu \vdash t : a$.

Proof. By induction on the derivation of $\Gamma,\mu,\iota,\mu \vdash b : a$.

Lemma 79 (Substitution). If $\Gamma, x : t_1,\Gamma_2 \vdash t : t_2$ and $\Gamma \vdash t' : t_1$, then $\Gamma_1, [t'/x]t_2 \vdash [t'/x]t' : [t'/x]t_2$.

Proof. By induction on the derivation of $\Gamma_1, x : t_1,\Gamma_2 \vdash t : t_2$.

We will show a few nontrivial cases.

Case:

$\Gamma, y : \iota y.t \vdash t : *$,

$\Gamma \vdash \iota y.t : *$.

Let $\Gamma = \Gamma_1, x : t_1,\Gamma_2$. We want to show $\Gamma_1, [t'/x]t_2 \vdash \iota y.[t'/x]t' : *$. By IH, we have $\Gamma_1, [t'/x]t_2, y : \iota y.[t'/x]t' \vdash [t'/x]t' : *$. So it is the case.

Case:

$\Gamma \vdash t : [t/y]t''$,

$\Gamma \vdash \iota y.t'' : *$.

Let $\Gamma = \Gamma_1, x : t_1,\Gamma_2$. We want to show $\Gamma_1, [t'/x]t_2 \vdash [t'/x]t' : \iota y.[t'/x]t''$. By IH, we have $\Gamma_1, [t'/x]t_2 \vdash [t'/x]t' : \iota y.[t'/x]t''$. So it is the case.

Case:

$\Gamma \vdash t : \iota y.t''$.

Let $\Gamma = \Gamma_1, x : t_1,\Gamma_2$. We want to show $\Gamma_1, [t'/x]t_2 \vdash [t'/x]t : \iota y.[t'/x]t''$. By IH, we have $\Gamma_1, [t'/x]t_2 \vdash [t'/x]t : \iota y.[t'/x]t''$. So it is the case.

Case:

$\Gamma, \mu \vdash t : t''$,

$\{, \mu \vdash t_j : a_j \}_{(t_j, a_j) \in \mu}$.

$\Gamma \vdash \mu t : \mu t''$.

Let $\Gamma = \Gamma_1, x : t_1,\Gamma_2$. We want to show $\Gamma_1, [t'/x]t_2 \vdash \mu [t'/x]t'$. By IH, we have $\Gamma_1, [t'/x]t_2 \vdash [t'/x]t' \vdash [t'/x]t''$ and $\{, [t'/x]t_2 \mu \vdash t_j : [t'/x]a_j \}_{(t_j, a_j) \in \mu}$. So it is the case.

Theorem 80 (Type Preservation). If $\Gamma \vdash \iota x.t$ and $\Gamma \vdash t : \iota x.t'$ and $\Gamma \vdash t : a$, then $\Gamma \vdash t' : a$.

Proof. By induction on the derivation of $\Gamma \vdash t : a$. We list a few nontrivial cases.

Case:

$\Gamma \vdash * : *$.

This case will not arise.

Case:

$x : a \in \Gamma$,

$\Gamma \vdash x : a$.

If $\Gamma \vdash x \sim t'$, this means $(x : a) \sim t' \in \Gamma$ and $\Gamma \vdash t' : a$ since $\Gamma \vdash \iota x.t$.

Case:

$\Gamma \vdash t : t_1$,

$\Gamma \vdash t_1 : t_2$,

$\Gamma \vdash t : *$.

In this case $\Gamma \vdash t \sim t'$. By IH, $\Gamma \vdash t : t_1$. Since $\Gamma \vdash t_1 : t_2$, we have $\Gamma \vdash t' : t_2$.

Case:

$\Gamma \vdash t : [t/x]t''$.

$\Gamma \vdash t : [t/x]t''$.

In this case $\Gamma \vdash t \sim t'$. By IH, $\Gamma \vdash t' : [t/x]t''$. Thus we have $\Gamma \vdash t' : [t/x]t''$. Since $\Gamma \vdash t' : t$, we have $\Gamma \vdash t' : [t/x]t''$ by Conv rule.

Case:

$\Gamma \vdash t : [t/x]t''$,

$\Gamma \vdash \iota x.t' : *$.

$\Gamma \vdash t : [t/x]t''$.

In this case $\Gamma \vdash t \sim t'$. By IH, $\Gamma \vdash t' : [t/x]t''$. Since $\Gamma \vdash t' : [t/x]t''$, we have $\Gamma \vdash t' : [t/x]t''$. Thus we have $\Gamma \vdash t' : [t/x]t''$.

Case:

$\Gamma \vdash \Pi x \vdash t_1, t_2 : t'_1$, $t'_2 : t''_1$.

$\Gamma \vdash t'_1, t'_2 : t'_2[t_2/x]t''_2$.

Suppose $\Gamma \vdash (\lambda x.t_1) v \sim [v/x]t_1$. Then we know $\Gamma \vdash (\lambda x.t_1) v : [v/x]t''_2$ and $\Gamma \vdash \lambda x.t_1 : \Pi x : t'_1, t'_2 \vdash t''_2$. By inversion on $\Gamma \vdash \lambda x.t_1 : \Pi x : t'_1, t'_2$, we have $\Gamma, x : a \vdash t_1 : t'_1$ and $\Gamma \vdash \Pi x : a, b =_{\beta,\mu,\iota,o} \Pi x : t'_1, t'_2$. By theorem $[34]$ we have $\Gamma, x : a \vdash t_1 : t'_1$ and $\Gamma \vdash \Pi x : a, b =_{\beta,\mu,\iota,o} \Pi x : t'_1, t'_2$. So we have $\Gamma, x : a \vdash t_1 : t'_1$ and $\Gamma \vdash v : a$. So by lemma $[79]$ we have $\Gamma \vdash [v/x]t_1 : [v/x]t''_2$, as required.

Suppose $\Gamma \vdash t_1 t_2 \sim t'_1 t'_2$, where $\Gamma \vdash t_1 \sim t'_1$. We know $\Gamma \vdash t_1 t_2 : [t_2/x]t''_2$ and $\Gamma \vdash t_1 : t'_1, t'_2 \vdash t''_2$. By IH, we know $\Gamma \vdash t'_1 : t_2 : [t_2/x]t''_2$. So we have $\Gamma \vdash t'_1 t'_2 : [t_2/x]t''_2$.

Suppose $\Gamma \vdash (\lambda x.t_1) t_2 \sim (\lambda x.t_1) t'_2$, where $\Gamma \vdash t_2 \sim t'_2$. We know $\Gamma \vdash (\lambda x.t_1) t_2 : [t_2/x]t''_2$ and $\Gamma \vdash \lambda x.t_1 : \Pi x : t'_1, t'_2 \vdash t''_2$. By IH, we know $\Gamma \vdash t_1 : t'_1, t'_2 \vdash t''_2$. So we have $\Gamma \vdash (\lambda x.t_1) t_2 : [t_2/x]t''_2$.
Case: 

\[ \Gamma, \bar{\mu} \vdash t : t' \quad \{ \Gamma, \bar{\mu} \vdash t_j : a_j \}_{(t_j, a_j) \in \bar{\mu}} \]

\[ \Gamma \vdash \mu t : \mu t' \]

Suppose \( \Gamma \vdash \mu x_j \rightarrow \mu t_j \), where \( x_j \mapsto t_j \in \mu \). We have \( \Gamma, \bar{\mu} \vdash x_j : t_j \). By inversion, \( \Gamma, \bar{\mu} \vdash x_j : a_j \) and \( \Gamma, \bar{\mu} \vdash a_j \triangleq_{\beta, \mu, i, o} t_j \). Since \( \Gamma, \bar{\mu} \vdash x_j : t_j \) and by lemma 68, we get \( \Gamma, \bar{\mu} \vdash a_j \triangleq_{\beta, \mu, i, o} t_j \). Since \( \Gamma, \bar{\mu} \vdash t_j : a_j \), by lemma 65, \( \Gamma, \bar{\mu} \vdash t_j : t_j \). Thus we have \( \Gamma \vdash \mu t_j : \mu t' \).

Suppose \( \Gamma \vdash \mu \bar{\mu} x_j \rightarrow \mu \bar{\mu} t_j \), where \( x_j \mapsto t_j \in \mu \). By inversion on \( \Gamma, \bar{\mu} \vdash \bar{\mu} x_j : t_j \), we have \( \Gamma, \bar{\mu} \vdash \bar{\mu} x_j : t_j \) and \( \Gamma, \bar{\mu} \vdash \bar{\mu} x_j : \tilde{t}_j \). By inversion on \( \Gamma, \bar{\mu} \vdash \bar{\mu} x_j : \tilde{t}_j \), we have \( \Gamma, \bar{\mu} \vdash \bar{\mu} x_j : b \), where \( (x_j : b) \in \mu \cup \bar{\mu} \) and \( \Gamma, \bar{\mu} \vdash b \triangleq_{\beta, \mu, i, o} t_a \). So \( \Gamma \vdash \mu \bar{\mu} b \rightarrow_{\beta, \mu, i, o} \mu \bar{\mu} b \rightarrow_{\beta, \mu, i, o} \mu \bar{\mu} t \). Since \( \Gamma \vdash \mu \bar{\mu} t_j : \mu \bar{\mu} t_j \) and \( \Gamma \vdash \mu \bar{\mu} \bar{\mu} t_j : \mu \bar{\mu} t' \), we have \( \Gamma \vdash \mu \bar{\mu} \bar{\mu} t_j : \mu \bar{\mu} t' \).

Suppose \( \Gamma \vdash \mu \bar{\mu} t \rightarrow *, \) we have \( \Gamma, \bar{\mu} \vdash * : t' \). We have \( \Gamma, \bar{\mu} \vdash * : * \). By inversion, we get \( \Gamma, \bar{\mu} \vdash * : \tilde{t} \). Then \( \Gamma, \bar{\mu} \vdash \tilde{t} : \tilde{t} \). We have \( \Gamma \vdash \mu \bar{\mu} \tilde{t} : \mu \bar{\mu} \tilde{t} \). By inversion on \( \Gamma \vdash \mu \bar{\mu} \tilde{t} : \mu \bar{\mu} \tilde{t} \), we get \( \Gamma \vdash \mu \bar{\mu} \tilde{t} : \mu \bar{\mu} \tilde{t} \). Thus we have \( \Gamma \vdash \mu \bar{\mu} \tilde{t} : \mu \bar{\mu} \tilde{t} \).