Semantics and Strong Normalization

Part I

Semantics

Set-theoretic semantics

We start with a family of sets $\{X_\alpha\}_{\alpha \in \mathbb{A}}$.

- The types $A \in \mathbb{T}$ are interpreted thus:

$$\llbracket \alpha \rrbracket = X_\alpha$$
$$\llbracket A \to B \rrbracket = \{f : \llbracket A \rrbracket \to \llbracket B \rrbracket\} \quad (= \llbracket B \rrbracket^{\llbracket A \rrbracket})$$

- If $\Gamma \vdash t : A$, then $\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \to \llbracket A \rrbracket$, where

$$\llbracket x_1{:}A_1,\ \ldots,\ x_n{:}A_n \rrbracket = \llbracket A_1 \rrbracket \times \cdots \times \llbracket A_n \rrbracket$$

- In particular, if $\vdash t : A$, then $\llbracket t \rrbracket$ is an element of $\llbracket A \rrbracket$.

Set-theoretic semantics

Let $\Gamma = \{x_1{:}A_1,\ \ldots,\ x_n{:}A_n\}$.
The interpretation $[\![t]\!]$ proceeds by induction on t:

- $\Gamma \vdash x_i : A_i$. Then

$$[\![x_i]\!] : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \xrightarrow{\pi_i} [\![A_i]\!]$$
$$[\![x_i]\!]\vec{a} = a_i$$

- $\Gamma \vdash st : B$ by $\Gamma \vdash s : A{\to}B$ and $\Gamma \vdash t : A$. Then, for $(a_1, \ldots, a_n) \in [\![\Gamma]\!]$, $[\![s]\!]\vec{a} \in [\![B]\!]^{[\![A]\!]}$, and $[\![t]\!]\vec{a} \in [\![A]\!]$.

$$[\![st]\!]\vec{a} = [\![s]\!]\vec{a}([\![t]\!]\vec{a})$$

- $\Gamma \vdash t : A \to B$ by $\Gamma, x{:}A \vdash t : B$. Then

$$[\![\lambda x{:}A.t]\!] : [\![\Gamma]\!] \to [\![A \to B]\!]$$
$$[\![\lambda x{:}A.t]\!]\vec{a} = a \mapsto [\![t]\!](\vec{a}; a)$$

example

Example

Suppose $\mathbb{A} = \{o\}$, and let $X_o = \mathbb{N}$. Then

$$\llbracket o \to o \rrbracket = \{f : \mathbb{N} \to \mathbb{N}\} = \mathbb{N}^{\mathbb{N}}$$

$$\llbracket (o \to o) \to (o \to o) \rrbracket = \{\Phi : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}\} = (\mathbb{N}^{\mathbb{N}})^{\mathbb{N}^{\mathbb{N}}}$$

$$\llbracket \lambda x{:}o.x \rrbracket = \llbracket \mathtt{I} \rrbracket = (n \mapsto n) = \mathsf{Id}_{\mathbb{N}} \qquad \in \mathbb{N}^{\mathbb{N}}$$

$$\llbracket \lambda x{:}o{\to}o\,\lambda y{:}o.x(xy) \rrbracket = \llbracket \mathsf{c}_2 \rrbracket = (f \mapsto f \circ f) \qquad \in (\mathbb{N}^{\mathbb{N}})^{\mathbb{N}^{\mathbb{N}}}$$

A typed lambda term of type $A \to B$ should be interpreted as a morphism between mathematical structures A and B.
How can we capture this generality?

Categories

- A category C consists of
  - a collection C_0 of *objects* (representing some mathematical structure), and
  - a collection C_1 of *structure-preserving morphisms* (between objects in C_0)
- The collection of morphisms from object A to object B is denoted as $C(A, B)$. (Also popular: $\mathrm{Hom}(A, B)$, $\mathrm{Mor}(A, B)$)
- One writes $f : A \to B$ if $f \in C(A, B)$.

Categories

- Examples

C_0	$C(A, B)$
Sets	$\{f : A \to B\}$
Groups	$\{h : A \to B \mid h(a_1 a_2) = h(a_1)h(a_2)\}$
Top. spaces	$\{f : A \to B \mid f^{-1}U \text{ open if } U \text{ open}\}$
Vector spaces	$\{L : A \to B \mid L(ax + by) = aL(x) + bL(y)\}$
Posets	$\{m : A \to B \mid a \leq a' \implies m(a) \leq m(a')\}$

Categories

Axioms for a category:

- For every object X, there is a morphism $\mathrm{Id}_X : X \to X$.
- For morphisms $f : X \to Y$ and $g : Y \to Z$, there is the composition morphism

$$g \circ f : X \to Z$$

- For $f : A \to X$, $g : X \to B$, the identity map Id_X satisfies

$$f = \mathrm{Id}_X \circ f \quad \text{and} \quad g = g \circ \mathrm{Id}_X$$

- The composition operation satisfies:

$$(f \circ g) \circ h = f \circ (g \circ h)$$

Universal properties

Many constructions of classical mathematics are characterized by their *mapping properties*:

- The free group on a set
- The Stone-Cech compactification of a space
- The algebraic closure of a field
- The Cauchy-completion of a metric space

These are non-trivial examples.
However, even the most elementary constructs of the language of mathematics can be captured by the "categorical perspective."

A puzzle

What is the universal mapping property of the singleton set $\{*\}$ (a set $\mathbf{1}$ with only one element $* \in \mathbf{1}$)?

A puzzle

What is the universal mapping property of the singleton set $\{*\}$ (a set $\mathbf{1}$ with only one element $* \in \mathbf{1}$)?
Answer: For any set X, there is exactly one map $c_* : X \to \mathbf{1}$.
(The constant map with value $*$.)

Puzzle no. 2

For two sets A, B, what is the universal property of the product $A \times B$?

Puzzle no. 2

For two sets A, B, what is the universal property of the product $A \times B$?
Answer: there are maps $\pi_1 : A \times B \to A$, and $\pi_2 : A \times B \to B$, such that any pair of maps $f : C \to A$ and $g : C \to B$ can be factored through the product:

$$\begin{array}{ccccc} & & C & & \\ & f \swarrow & \downarrow (f,g) & \searrow g & \\ A & \stackrel{\pi_1}{\leftarrow} & A \times B & \stackrel{\pi_2}{\rightarrow} & B \end{array}$$

- In Set Theory, the natural numbers are defined as

$$\mathbb{N} := \bigcap \{X \mid \emptyset \in X, \quad n \in X \Rightarrow n \cup \{n\} \in X\}$$

- Up to isomorphism, the natural numbers are determined by any set N which is freely generated from a distinguished element $0 \in N$ and a function $S : N \to N$.
- The condition of being "freely generated" states the universal property of the natural numbers: for any set X,
  given an element $x_0 \in X$ and a function $x_S : X \to X$,
  there is a canonical map $h : N \to X$ (it is given by recursion).

natural numbers

This leads to the notion of a *natural numbers object*.

Definition

Let $N \in C_0$ come with maps $z : \mathbf{1} \to N$, $s : N \to N$ such that, for any other object X with maps $b : \mathbf{1} \to X$ and $f : X \to X$, there exists a unique map $i : N \to X$ satisfying

$$\begin{aligned} i \circ z &= b \\ i \circ s &= f \circ i \end{aligned}$$

Such an N is called a natural numbers object (nno).

Function space

We have now seen abstract categorical descriptions of

- Singletons (generally, terminal objects)
- Products
- Natural numbers

To interpret the simply typed lambda calculus, we need the notion of a "function space" object. The appropriate categorical setting for this is that of *Cartesian closed categories*.
Idea: we want a category where, for any pair of objects A, B, there is another object "$A \to B$" which in a natural way represents $C(A, B)$, the collection of maps from A to B.

CCCs

Let C be a category with a terminal object $\mathbf{1}$.
C is a *Cartesian category* if, for any two objects A, B,
there is an object $A \times B$ with the following properties:

- there are maps $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$,
- for any pair of maps $f : C \to A$, $g : C \to B$, there exists a unique map $(f, g) : C \to A \times B$ such that

$$\pi_1 \circ (f, g) = f$$
$$\pi_2 \circ (f, g) = g$$

- In other words, the following diagram commutes, for any f, g:

TheStupidDiagramGoesHere

CCC

Let C be a Cartesian category.

- C is a *Cartesian closed category* if, for any two objects Y, Z, there is an object Z^Y and a canonical isomorphism

$$\begin{array}{rcl} \{f : X \times Y \to Z\} & \Longleftrightarrow & \{g : X \to Z^Y\} \\ f(x,y) & \longleftrightarrow & (x \mapsto f_x(y)) \end{array}$$

- Taking $X = Z^Y$ and $g = \mathrm{Id}_{Z^Y}$ yields the *evaluation map*

$$\mathrm{ev}_{Y,Z} : Z^Y \times Y \to Z$$

Examples

$\mathcal{C}_0$	$A \times B$	B^A
Sets	$A \times B$	$\{f : A \to B\}$
Top. Spaces	$X \times Y$, product topology	$C(X, Y)$, compact-open topolo
Posets	$X \times Y$, product order	$C(X, Y)$, ordered pointwise

Categorical Semantics

Let C be a ccc.
The simply typed lambda calculus is interpreted as follows.
Types are interpreted by objects of C:

- Choose $X_\alpha \in C_0$ for each $\alpha \in \mathbb{A}$
- $A \in \mathbb{T}$ is interpreted by induction
  - $[\![\alpha]\!] = X_\alpha$
  - $[\![A \to B]\!] = [\![B]\!]^{[\![A]\!]}$

A context $\Gamma = \{x_1{:}A_1, \ldots, x_n{:}A_n\}$ is interpreted by the product

$$[\![\Gamma]\!] = [\![A_1]\!] \times \cdots \times [\![A_n]\!]$$

A term t of type A in the context Γ is interpreted by an arrow

$$[\![t]\!] : [\![\Gamma]\!] \to [\![A]\!]$$

In particular, a closed term is interpreted by an arrow $\mathbf{1} \to [\![A]\!]$.

Categorical Semantics

Let $\Gamma = \{x_1{:}A_1,\ \ldots,\ x_n{:}A_n\}$.
The interpretation $[\![t]\!]$ proceeds by induction on t:

- $\Gamma \vdash x_i : A_i$. Then

$$[\![x_i]\!] : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \xrightarrow{\pi_i} [\![A_i]\!]$$

- $\Gamma \vdash st : B$ by $\Gamma \vdash s : A{\to}B$ and $\Gamma \vdash t : A$. Then

$$[\![st]\!] : [\![\Gamma]\!] \xrightarrow{([\![s]\!],[\![t]\!])} [\![B]\!]^{[\![A]\!]} \times [\![A]\!] \xrightarrow{\mathsf{ev}_{[\![A]\!],[\![B]\!]}} [\![B]\!]$$

- $\Gamma \vdash t : A{\to}B$ by $\Gamma, x{:}A \vdash t : B$. Then

$$[\![\lambda x{:}A.t]\!] : [\![\Gamma]\!] \longrightarrow [\![B]\!]^{[\![A]\!]}$$

  is the transpose of

$$[\![\Gamma]\!] \times [\![A]\!] \xrightarrow{[\![t]\!]} [\![B]\!]$$

Relationship between syntax and semantics.

- "Cartesian closed categories are the canonical semantic universe of the simply typed lambda calculus. Conversely, the internal language of Cartesian closed categories is that of simple type theory."
- There is a subtlety: CCCs have more structure than the function space between two types: they also have products and a terminal object (which acts as the unit element for the product).
- Questsion: what should we add to the system of types and terms in order to have the full language of cccs?

Simple type theory

Any type theory consists of three components

- Syntax:

$$\mathbb{T} ::= \alpha \mid \mathbb{T} \to \mathbb{T}$$

$$t ::= x \mid \lambda x{:}\mathbb{T}.t \mid t\,t$$

- Typing rules:

$$\frac{(x{:}A) \in \Gamma}{\Gamma \vdash x{:}A}\,(\mathrm{Ax})$$

$$\frac{\Gamma, x{:}A \vdash r{:}B}{\Gamma \vdash \lambda x{:}A.r : A{\to}B}\,(\to\mathrm{I}) \qquad \frac{\Gamma \vdash s{:}A{\to}B \qquad \Gamma \vdash t{:}A}{\Gamma \vdash st{:}B}\,(\to\mathrm{E})$$

- Computation rules:

$$\beta : \qquad (\lambda x{:}A.r)a = r[a/x]$$

Simple type theory

Any type theory consists of three components

- Syntax:

$$\mathbb{T} ::= \alpha \mid \mathbb{T} \to \mathbb{T} \mid \mathbb{T} \times \mathbb{T} \mid$$

$$t ::= x \mid \lambda x{:}\mathbb{T}.t \mid t\,t \mid (t,t) \mid \pi_1 t \mid \pi_2 t \mid$$

- Typing rules:

$$\frac{(x{:}A) \in \Gamma}{\Gamma \vdash x{:}A}\,(\mathrm{Ax})$$

$$\frac{\Gamma, x{:}A \vdash r{:}B}{\Gamma \vdash \lambda x{:}A.r : A \to B}\,(\to\mathrm{I}) \qquad \frac{\Gamma \vdash s{:}A \to B \qquad \Gamma \vdash t{:}A}{\Gamma \vdash st{:}B}\,(\to\mathrm{E})$$

$$\frac{\Gamma \vdash s{:}A \qquad \Gamma \vdash t{:}B}{\Gamma \vdash (s,t) : A \times B}\,(\times\mathrm{I}) \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_1 p : A}\,(\times_1\mathrm{E}) \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \pi_2 p : B}\,(\times_2\mathrm{E})$$

- Computation rules:

$$\beta : \quad (\lambda x{:}A.r)a = r[a/x]$$

$$\pi : \quad \pi_i(a_1, a_2) = a_i \qquad i \in \{1, 2\}$$

Simple type theory

Any type theory consists of three components

- Syntax:

$$\mathbb{T} ::= \alpha \mid \mathbb{T} \to \mathbb{T} \mid \mathbb{T} \times \mathbb{T} \mid \mathbb{N}$$

$$t ::= x \mid \lambda x{:}\mathbb{T}.t \mid t\,t \mid (t,t) \mid \pi_1 t \mid \pi_2 t \mid 0 \mid St \mid I_{\mathbb{T}} tt$$

- Typing rules:

$$\frac{(x{:}A) \in \Gamma}{\Gamma \vdash x{:}A}(\mathrm{Ax})$$

$$\frac{\Gamma, x{:}A \vdash r{:}B}{\Gamma \vdash \lambda x{:}A.r : A{\to}B}(\to\mathrm{I}) \qquad \frac{\Gamma \vdash s{:}A{\to}B \qquad \Gamma \vdash t{:}A}{\Gamma \vdash st{:}B}(\to\mathrm{E})$$

$$\frac{\Gamma \vdash s{:}A \qquad \Gamma \vdash t{:}B}{\Gamma \vdash (s,t) : A{\times}B}(\times\mathrm{I}) \qquad \frac{\Gamma \vdash p : A{\times}B}{\Gamma \vdash \pi_1 p : A}(\times_1\mathrm{E}) \qquad \frac{\Gamma \vdash p : A{\times}B}{\Gamma \vdash \pi_2 p : B}(\times_2\mathrm{E})$$

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}}(\mathbb{N}_0\mathrm{I}) \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash Sn : \mathbb{N}}(\mathbb{N}_S\mathrm{I}) \qquad \frac{\Gamma \vdash b : X \qquad \Gamma \vdash f : X{\to}X}{\Gamma \vdash I_X bf : \mathbb{N} \to X}(\mathbb{N}\mathrm{E})$$

- Computation rules:

$$\beta : \quad (\lambda x{:}A.r)a = r[a/x]$$

$$\pi : \quad \pi_i(a_1,a_2) = a_i \qquad i \in \{1,2\}$$

$$\iota : \quad I_X bf0 = b$$

$$\iota : \quad I_X bf(Sn) = f(I_X bfn)$$

Type theory provides a uniform term language for concepts defined abstractly by their universal properties.

- The map appearing in the universal property of the natural numbers, technically called the *iteration functional*

$$I_X bf 0 = b$$
$$I_X bf (Sn) = f(I_X bfn)$$

  is not sufficient to define the predecessor function.
- But one can use the product (present in any ccc) to define it using Kleene's laughing gas trick:

$$\mathrm{Pred}\, n = \pi_2(I_{\mathbb{N}\times\mathbb{N}}\ (0,0)\ (\lambda p{:}\mathbb{N}\times\mathbb{N}.(S(\pi_1 p), \pi_1 p))\ n)$$

It is more natural to simply allow f to know the iteration step n. This results in the principle of *primitive recursion*:

$$\frac{\Gamma \vdash b : X \qquad \Gamma \vdash f : \mathbb{N} \to X \to X}{\Gamma \vdash R_X bf : \mathbb{N} \to X} \text{(}\mathbb{N}\text{E)}$$

with reduction rules:

$$\begin{aligned} R_X bf 0 &= b \\ R_X bf (Sn) &= fn(R_X bfn) \end{aligned}$$

With a recursor, the predecessor can be defined directly

$$\mathrm{Pred}\, n = R_X\, 0\, (\lambda nx.n)$$

Induction

When $X : \mathbb{N} \to *$ is a type which depends on the recursed variable, the dependence of f on n cannot be avoided:

$$\frac{\Gamma \vdash b : X0 \qquad \Gamma \vdash f : \forall n{:}\mathbb{N}.Xn \to X(Sn)}{\Gamma \vdash \mathrm{Ind}_X bf : \forall n{:}\mathbb{N}.Xn}\,(\mathbb{N}\mathrm{E})$$

This is the *induction principle*. It is much more powerful than the (non-dependent) recursion principle.
(Roughly, recursion says that $\mathbb{N}$ is a *weakly* initial $\{0, S\}$-algebra, i.e. that the universal morphism exists; induction also says that it is unique — that $\mathbb{N}$ is the “standard model” of Peano axioms.)

Dependent eliminators

```
Check nat_rect.
> nat_rect
     : forall P : nat -> Type,
       P 0 -> (forall n : nat, P n -> P (S n))
           -> forall n : nat, P n
```

Any inductive definition has a corresponding induction principle, giving the dependent version of its universal mapping property.

Puzzle no.3

What is the dependent elimination rule for the product type $A \times B$?

Puzzle no.3

What is the dependent elimination rule for the product type $A \times B$?
Hint. The projection operators π_i can be replaced by a "product recursor"

$$R_X^{A\times B} : (A \to B \to X) \to A \times B \to X$$
$$R_X^{A\times B}\, h\,(a, b) = hab$$

This presents $A \times B$ as an inductive type.

Puzzle no.3

What is the dependent elimination rule for the product type $A \times B$?
Hint. The projection operators π_i can be replaced by a "product recursor"

$$R_X^{A \times B} : (A \to B \to X) \to A \times B \to X$$
$$R_X^{A \times B}\, h\,(a, b) = hab$$

This presents $A \times B$ as an inductive type.
Answer: Allowing X to be a dependent type over $A \times B$ (so that $X : A \times B \to *$) yields

```
prod_rect
     : forall (A B : Type) (P : A * B -> Type),
       (forall (a : A) (b : B), P (a, b))
       -> forall p : A * B, P p
```

The function type $A \to B$ is not inductive.
The map on types $(A, B \mapsto A \to B)$ is not monotone in the first argument: as A gets bigger, $A \to B$ will generally get smaller.
So the universal property instead goes via the adjunction

$$\mathrm{Mor}(C \times A, B) \cong \mathrm{Mor}(C, A \to B)$$

(The negatively-occurring C encodes the context.)

The function type $A \to B$ is not inductive.
The map on types $(A,B \mapsto A \to B)$ is not monotone in the first argument: as A gets bigger, $A \to B$ will generally get smaller.
So the universal property instead goes via the adjunction

$$\mathrm{Mor}(C \times A, B) \cong \mathrm{Mor}(C, A \to B)$$

(The negatively-occurring C encodes the context.)
Puzzle for the break:
How could a dependent version of this property be formulated?
(IOW, what is the "dependent elimination principle" for the simply typed lambda calculus?)

Part II
Strong Normalization

Logical Relations

Solution to the puzzle:

For every type $A \in \mathbb{T}$, we are given a predicate $PA : A \to *$ on terms of type A.

That is, we are given $P : \forall A{:}\mathbb{T}.A{\to}*$

The "elimination principle" specifies which conditions guarantee that PA will hold of all terms of type A.

Its conclusion has type $\forall A : \mathbb{T}\, \forall a : A.\, PAa$

$P(A{\to}B)f$ holds for a term $f : A \to B$ if

$$\forall a{:}A.PAa \to PB(fa)$$

Logical Relations

Definition

A predicate $P : \forall A{:}\mathbb{T}.A{\to}*$ is a *logical relation* if

$$P(A{\to}B)f \iff \forall a{:}A.PAa \to PB(fa)$$

If P is a logical relation, then

- P holds of an application st provided $P(A{\to}B)s$ and PAt
- P holds of an abstraction $\lambda x{:}A.t$ provided $PAa \Rightarrow PBt[x{:=}a]$

Logical Relations

The variables of the context Γ now add hypotheses $P\Gamma$:
If $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$, the typing $\Gamma \vdash t : A$ eliminates into

$$\forall(a_1{:}A_1, \ldots, a_n{:}A_n)\;\; PA_1 a_1 \wedge \cdots \wedge PA_n a_n \Rightarrow PAt[\vec{x}{:=}\vec{a}]$$

Theorem

Let P be a logical relation. Let $\Gamma \vdash t : A$.
If $PA_i x_i$ holds for each $(x_i{:}A_i) \in \Gamma$, then PAt holds.

Strong Normalization

Three fundamental pillars of type theory:

- Church–Rosser: $M = N \iff M \twoheadrightarrow Z \twoheadleftarrow N$
  ("Consistency of computation")
- Subject Reduction: $\Gamma \vdash M : A,\ M \twoheadrightarrow N \Longrightarrow \Gamma \vdash N : A$
  ("Consistency between computation and logic")
- Strong Normalization: $\Gamma \vdash M : A \Longrightarrow SN(M)$
  ("Consistency of logic")

Strong Normalization

Assume $\Gamma \vdash M : A$.
To show: there is no infinite reduction sequence starting from M

$$M \to M' \to M'' \to \cdots$$

Clearly, if t is strongly normalizing, then $\lambda x{:}A.t$ is too.
But why should application preserve SN?
($\omega = \lambda x.xx$ is a normal form, but $\Omega = \omega\omega$ is unsolvable.)
The induction has to be higher-order: logical relations.

By induction on A, define the predicate $\mathsf{Comp}_A : A \to *$:

- $\mathsf{Comp}_\alpha t \;:=\; SN(t)$
- $\mathsf{Comp}_{A\to B} f \;:=\; \forall a{:}A.\mathsf{Comp}_A a \Rightarrow \mathsf{Comp}_B (fa)$

By definition, Comp is a logical relation.

Lemma

1. $\mathrm{Comp}_A t \implies SN(t)$.
2. $\vec{t} \in SN \implies \mathrm{Comp}_A x\vec{t}$

Proof.

For atomic types α, both points are trivial.
For function types $A \to B$, we suppose given $f : A \to B$ such that, if Comp_A holds at $a{:}A$, then $\mathrm{Comp}_B(fa)$. In particular, $\mathrm{Comp}_B(fx)$. By inductive hypothesis, $SN(fx)$. Hence $SN(f)$.
Now let $\vec{t} \in SN$, and suppose that $\mathrm{Comp}_A a$. By inductive hypothesis, we have $SN(a)$, and thus also $\mathrm{Comp}_B(x\vec{t}a)$. Hence

$$\mathrm{Comp}_{A\to B}(x\vec{t})$$

□

SN

Theorem

$\Gamma \vdash t : A \Longrightarrow SN(t)$

Proof.

By 2, every variable of any type is computable. Hence, the hypotheses CompΓ is satisfied for every $\Gamma \vdash t : A$. By the logical-relations theorem, we have $\text{Comp}_A t$. By 1, SN(t). $\square$

Lemma

$\mathrm{Comp}_A(r[x{:=}s]\vec{t}),\ r, s, \vec{t} \in SN \Longrightarrow \mathrm{Comp}_A((\lambda x.r)s\vec{t}).$

Proof.

When α is atomic, this is just

$$r, s, \vec{t}, r[x{:=}s]\vec{t} \in SN \Longrightarrow (\lambda x.r)s\vec{t} \in SN$$

Given $r, s, \vec{t} \in SN$, suppose that $\mathrm{Comp}_{A\to B}(r[x{:=}s]\vec{t})$.
For $a{:}A$, we want to show that $\mathrm{Comp}_A a \Longrightarrow \mathrm{Comp}_B((\lambda x.r)s\vec{t}a)$.
Indeed, $\mathrm{Comp}_A a$ gives $\mathrm{Comp}_B(r[x{:=}s]\vec{t}a)$. By inductive hypothesis, $\mathrm{Comp}_B((\lambda x.r)s\vec{t}a)$ as desired. $\square$

Extensions

This proof method easily extends to the full Gödel's system T on the slides before.
Girard extended it to second-order and higher-order logic.