Lambda Calculi With Explicit Substitutions

slides: https://github.com/donovancrichton/Talks

Donovan Crichton

February 2025

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

Lambda Calculi With Explicit Substitutions
slides: https://github.com/donovancrichton/Talks

Donovan Crichton

February 2025

The goal of this talk

- How to read (theory) syntax.
- STLC - Simply Typed Lambda Calculus.
- Substituions in anger - De-Bruijn Indicies.
- Explicit Subsitutions.
- Explicit Substitutions in modern type theory.

The goal of this talk

- How to read (theory) syntax.
- STLC - Simply Typed Lambda Calculus.
- Substituions in anger - De-Bruijn Indicies.
- Explicit Subsitutions.
- Explicit Substitutions in modern type theory.

Reading Syntax: Grammar

Natural Numbers (Peano)

$$\mathbb{N} ::= Z \mid S\,\mathbb{N}$$

Untyped Lambda Caluclus

$$\begin{aligned} V &::= x, y, z, \dots \\ M, N &::= V && \text{Variable.} \\ &\mid M\,N && \text{Application.} \\ &\mid \lambda V.M && \text{Abstraction.} \end{aligned}$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Grammar

Reading Syntax: Grammar
Natural Numbers (Peano)
$\mathbb{N} ::= Z \mid S\,\mathbb{N}$
Untyped Lambda Caluclus
$V ::= x, y, z, \dots$
$M, N ::= V$ Variable.
$\mid M\,N$ Application.
$\mid \lambda V.M$ Abstraction.

Reading Syntax: Grammar

Natural Numbers (Peano)

$$\mathbb{N} ::= \ Z \mid S\,\mathbb{N}$$

Reading Syntax: Grammar

Natural Numbers (Peano)

$\mathbb{N} ::= \ Z \mid S\,\mathbb{N}$

Reading Syntax: Grammar

Our <symbol> name...

Natural Numbers (Peano)

$$\boxed{\mathbb{N}} ::= Z \mid S\,\mathbb{N}$$

Reading Syntax: Grammar
Our <symbol> name...
Natural Numbers (Peano)
$\boxed{\mathbb{N}} ::= Z \mid S\,\mathbb{N}$

Reading Syntax: Grammar

...is defined in the following ways:

Natural Numbers (Peano)

$$\mathbb{N} \boxed{::=} Z \mid S\,\mathbb{N}$$

Reading Syntax: Grammar
...is defined in the following ways:
Natural Numbers (Peano)
$\mathbb{N} \boxed{::=} Z \mid S\,\mathbb{N}$

Reading Syntax: Grammar

The letter Z by itself.

Natural Numbers (Peano)

$$\mathbb{N} ::= \boxed{Z} \mid S\,\mathbb{N}$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides:
https://github.com/donovancrichton/Talks

└─Reading Syntax: Grammar

Reading Syntax: Grammar

The letter Z by itself.

Natural Numbers (Peano)

$\mathbb{N} ::= \boxed{Z} \mid S\,\mathbb{N}$

Reading Syntax: Grammar

...or...

Natural Numbers (Peano)

$$\mathbb{N} ::= Z \boxed{|} S\,\mathbb{N}$$

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└─Reading Syntax: Grammar

Reading Syntax: Grammar

...or...

Natural Numbers (Peano)

$\mathbb{N} ::= Z \boxed{|} S\,\mathbb{N}$

Reading Syntax: Grammar

The letter S followed by a space, followed by any $\mathbb{N}$.

Natural Numbers (Peano)

$$\mathbb{N} ::= Z \mid \boxed{S\,\mathbb{N}}$$

Reading Syntax: Grammar

The letter S followed by a space, followed by any $\mathbb{N}$.

Natural Numbers (Peano)

$$\mathbb{N} ::= Z \mid \boxed{S\,\mathbb{N}}$$

Our set of expressions/terms called V...

Untyped Lambda Caluclus

$$
\begin{aligned}
\boxed{V} &::= x, y, z, \ \dots \\
M, N &::= V && \text{Variable.} \\
&\mid M\ N && \text{Application.} \\
&\mid \lambda V.M && \text{Abstraction.}
\end{aligned}
$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Grammar

Reading Syntax: Grammar

Our set of expressions/terms called V...

Untyped Lambda Caluclus

$\boxed{V} ::= x, y, z, \dots$

$M, N ::= V$ Variable.

$\mid M\ N$ Application.

$\mid \lambda V.M$ Abstraction.

Reading Syntax: Grammar

is given, or defined by:

Untyped Lambda Caluclus

$$
\begin{aligned}
V & \boxed{::=}\; x, y, z, \;\ldots \\
M, N & ::= \; V && \text{Variable.} \\
& \mid \; M\,N && \text{Application.} \\
& \mid \; \lambda V.M && \text{Abstraction.}
\end{aligned}
$$

Reading Syntax: Grammar

is given, or defined by:

Untyped Lambda Caluclus

$V \boxed{::=} x, y, z, \ldots$
$M, N ::= V$ Variable.
$\mid M\,N$ Application.
$\mid \lambda V.M$ Abstraction.

Reading Syntax: Grammar

'x', 'y', 'z', or *any other lower case letter* (lower case words also implied)

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= \boxed{x, y, z, \ \ldots} \\
M, N &::= V && \text{Variable.} \\
&\mid M\ N && \text{Application.} \\
&\mid \lambda V.M && \text{Abstraction.}
\end{aligned}
$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└─Reading Syntax: Grammar

Reading Syntax: Grammar

'x', 'y', 'z', or *any other lower case letter* (lower case words also implied)

Untyped Lambda Caluclus

$V ::= \boxed{x, y, z, \ \ldots}$
$M, N ::= V$ Variable.
$\mid M\ N$ Application.
$\mid \lambda V.M$ Abstraction.

Reading Syntax: Grammar

Our lambda terms, denoted by N or M (other capital letters implied).

Untyped Lambda Caluclus

$$
\begin{array}{rll}
V ::= & x, y, z, \ \ldots & \\
\boxed{M, N} ::= & V & \text{Variable.} \\
& \mid M\ N & \text{Application.} \\
& \mid \lambda V.M & \text{Abstraction.}
\end{array}
$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Grammar

Reading Syntax: Grammar

Our lambda terms, denoted by N or M (other capital letters implied).

Untyped Lambda Caluclus

$$
\begin{array}{rll}
V ::= & x, y, z, \ \ldots & \\
\boxed{M, N} ::= & V & \text{Variable.} \\
& \mid M\ N & \text{Application.} \\
& \mid \lambda V.M & \text{Abstraction.}
\end{array}
$$

are given by:

Untyped Lambda Caluclus

$$
\begin{array}{rll}
V ::= & x, y, z, \ \ldots & \\
M, N \boxed{::=} & V & \text{Variable.} \\
\mid & M\ N & \text{Application.} \\
\mid & \lambda V.M & \text{Abstraction.}
\end{array}
$$

Reading Syntax: Grammar
are given by:
Untyped Lambda Caluclus
$V ::= x, y, z, \ \ldots$
$M, N \boxed{::=} V$ Variable.
$\mid M\ N$ Application.
$\mid \lambda V.M$ Abstraction.

Reading Syntax: Grammar

A V...

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= \; x, y, z, \; \ldots \\
M, N &::= \; \boxed{V} && \text{Variable.} \\
&\mid \; M\,N && \text{Application.} \\
&\mid \; \lambda V.M && \text{Abstraction.}
\end{aligned}
$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Grammar

Reading Syntax: Grammar

A V...

Untyped Lambda Caluclus

$V ::= x, y, z, \ldots$

$M, N ::= \boxed{V}$ Variable.

$\mid M\,N$ Application.

$\mid \lambda V.M$ Abstraction.

Reading Syntax: Grammar

…or,

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= \; x, y, z, \; \ldots \\
M, N &::= \; V && \text{Variable.} \\
& \quad \boxed{\,|\,}\; M\ N && \text{Application.} \\
& \quad |\ \lambda V.M && \text{Abstraction.}
\end{aligned}
$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Grammar

Reading Syntax: Grammar
…or,
Untyped Lambda Caluclus
$V ::= x, y, z, \ldots$
$M, N ::= V$ Variable.
$\boxed{|}\ M\ N$ Application.
$|\ \lambda V.M$ Abstraction.

Reading Syntax: Grammar

A lambda term (M), followed by a space, followed by another lambda term (N).

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= x, y, z, \ \ldots & & \\
M, N &::= V & & \text{Variable.} \\
&\mid \boxed{M\ N} & & \text{Application.} \\
&\mid \lambda V.M & & \text{Abstraction.}
\end{aligned}
$$

Reading Syntax: Grammar

A lambda term (M), followed by a space, followed by another lambda term (N).

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= x, y, z, \ \ldots & & \\
M, N &::= V & & \text{Variable.} \\
&\mid \boxed{M\ N} & & \text{Application.} \\
&\mid \lambda V.M & & \text{Abstraction.}
\end{aligned}
$$

Or,

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= \; x, y, z, \; \ldots \\
M, N &::= \; V && \text{Variable.} \\
&\quad\; \mid M \; N && \text{Application.} \\
&\quad\; \boxed{\mid} \; \lambda V.M && \text{Abstraction.}
\end{aligned}
$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Grammar

Reading Syntax: Grammar
Or,
Untyped Lambda Caluclus
$V ::= x, y, z, \ldots$
$M, N ::= V$ Variable.
$\mid M\ N$ Application.
$\boxed{\mid}\ \lambda V.M$ Abstraction.

The λ symbol, followed by a V element, followed by a ".", followed by a lambda term (M)

Untyped Lambda Caluclus

$$
\begin{array}{rll}
V ::= & x, y, z, \ \ldots & \\
M, N ::= & V & \text{Variable.} \\
| & M\ N & \text{Application.} \\
| & \boxed{\lambda V.M} & \text{Abstraction.}
\end{array}
$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Grammar

Reading Syntax: Grammar

The λ symbol, followed by a V element, followed by a ".", followed by a lambda term (M)

Untyped Lambda Caluclus

$V ::= x, y, z, \ \ldots$
$M, N ::= V$ Variable.
$| \ M\ N$ Application.
$| \ \boxed{\lambda V.M}$ Abstraction.

Why does Grammar look like this?

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= x, y, z, \ \dots \\
M, N &::= V && \text{Variable.} \\
&\mid M\ N && \text{Application.} \\
&\mid \lambda V.M && \text{Abstraction.}
\end{aligned}
$$

Why does Grammar look like this?
Untyped Lambda Caluclus
$V ::= x, y, z, \ \dots$
$M, N ::= V$ Variable.
$\mid M\ N$ Application.
$\mid \lambda V.M$ Abstraction.

Untyped Lambda Caluclus

$$\begin{aligned} V &::= x, y, z, \ \dots \\ M, N &::= V && \text{Variable.} \\ &\mid M\,N && \text{Application.} \\ &\mid \lambda V.M && \text{Abstraction.} \end{aligned}$$

- The smallest possible definition.

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Why does Grammar look like this?

Why does Grammar look like this?

Untyped Lambda Caluclus

$V ::= x, y, z, \dots$
$M, N ::= V$ Variable.
$\mid M\,N$ Application.
$\mid \lambda V.M$ Abstraction.

- The smallest possible definition.

Why does Grammar look like this?

Untyped Lambda Caluclus

$$
\begin{aligned}
V &::= x, y, z, \dots & \\
M, N &::= V & \text{Variable.} \\
&\mid M\, N & \text{Application.} \\
&\mid \lambda V.M & \text{Abstraction.}
\end{aligned}
$$

- The smallest possible definition.
- Can be used to generate arbitrary elements.

Why does Grammar look like this?
Untyped Lambda Caluclus
$V ::= x, y, z, \dots$
$M, N ::= V$ Variable.
$\mid M\, N$ Application.
$\mid \lambda V.M$ Abstraction.
- The smallest possible definition.
- Can be used to generate arbitrary elements.

What about in programming?

Natural Numbers (Peano)

```
data Nat = Z | S Nat

zero :: Nat
zero = Z

one :: Nat
one = S Z

two :: Nat
two = S (S Z)

three :: Nat
three = S two
```


What about in programming?

Natural Numbers (Peano)

```
data Nat = Z | S Nat

zero :: Nat
zero = Z

one :: Nat
one = S Z

two :: Nat
two = S (S Z)

three :: Nat
three = S two
```

What about in programming?

Untyped Lambda Caluclus (Idris)

```
V : Type
V = String

data Λ = Var V
       | App Λ Λ
       | Abs V Λ

id : Λ
id = Abs "x" (Var "x")

const : Λ
const = Abs "a" (Abs "b" (Var "a"))
```


What about in programming?

Untyped Lambda Caluclus (Idris)

```
V : Type
V = String

data Λ = Var V
       | App Λ Λ
       | Abs V Λ

id : Λ
id = Abs "x" (Var "x")

const : Λ
const = Abs "a" (Abs "b" (Var "a"))
```

Church Encoding - Naturals and Booleans

Church Booleans

$$\text{True} = \lambda a.\lambda b.a$$
$$\text{False} = \lambda a.\lambda b.b$$

Church Encoding - Naturals and Booleans
Church Booleans
True = $\lambda a.\lambda b.a$
False = $\lambda a.\lambda b.b$

Church Encoding - Naturals and Booleans

Church Booleans

$$\text{True} = \lambda a.\lambda b.a$$
$$\text{False} = \lambda a.\lambda b.b$$

Church Naturals

$$0 = \lambda f.\lambda x.x$$
$$1 = \lambda f.\lambda x.f\,x$$
$$2 = \lambda f.\lambda x.f\,(f\,x)$$
$$\vdots$$

Church Encoding - Naturals and Booleans

Church Booleans

True = λa.λb.a
False = λa.λb.b

Church Naturals

0 = λf.λx.x
1 = λf.λx.f x
2 = λf.λx.f (f x)
⋮

Church Encoding - Functions and Predicates

Functions and Predicates

$$\mathrm{Succ}(n, f, x) = \lambda n.\lambda f.\lambda x.\lambda f\,(n\; f\; x)$$

$$\mathrm{Add}(m, n, f, x) = \lambda m.\lambda n.\lambda f.\lambda x.m\; f\,(n\; f\; x)$$

$$\mathrm{IsZero}(n) = \lambda n.n\;(\lambda x.\mathrm{False})\;\mathrm{True}$$

Church Encoding - Functions and Predicates
Functions and Predicates
$\mathrm{Succ}(n, f, x) = \lambda n.\lambda f.\lambda x.\lambda f\,(n\; f\; x)$
$\mathrm{Add}(m, n, f, x) = \lambda m.\lambda n.\lambda f.\lambda x.m\; f\,(n\; f\; x)$
$\mathrm{IsZero}(n) = \lambda n.n\;(\lambda x.\mathrm{False})\;\mathrm{True}$

Church Encoding - Functions and Predicates

Functions and Predicates

$$
\begin{aligned}
\mathrm{Succ}(n, f, x) &= \lambda n.\lambda f.\lambda x.\lambda f\,(n\ f\ x)\\
\mathrm{Add}(m, n, f, x) &= \lambda m.\lambda n.\lambda f.\lambda x.m\ f\,(n\ f\ x)\\
\mathrm{IsZero}(n) &= \lambda n.n\ (\lambda x.\lambda a.\lambda b.b)\ \lambda a.\lambda b.a
\end{aligned}
$$

Church Encoding - Functions and Predicates

Functions and Predicates

Succ(n, f, x) = λn.λf.λx.λf (n f x)
Add(m, n, f, x) = λm.λn.λf.λx.m f (n f x)
IsZero(n) = λn.n (λx.λa.λb.b) λa.λb.a

Typing Rules

$$\frac{\Gamma \vdash A \ \ \text{Type} \qquad \Gamma, x : A \vdash B \ \ \text{Type}}{\Gamma \vdash A \to B \ \ \text{Type}}(\mathit{Ty\text{-}Arrow})$$

$$\frac{\Gamma \vdash A \to B \ \text{Type} \qquad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda(x : A).M : A \to B}(\mathit{Func})$$

$$\frac{\Gamma \vdash A \to B \ \text{Type} \qquad \Gamma \vdash f : A \to B \qquad \Gamma \vdash x : A}{\Gamma \vdash f\, x : B}(\mathit{App})$$

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└Reading Syntax: Typing Rules

Reading Syntax: Typing Rules

Typing Rules

$$\frac{\Gamma \vdash A \ \ \text{Type} \qquad \Gamma, x : A \vdash B \ \ \text{Type}}{\Gamma \vdash A \to B \ \ \text{Type}}(\mathit{Ty\text{-}Arrow})$$

$$\frac{\Gamma \vdash A \to B \ \text{Type} \qquad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda(x : A).M : A \to B}(\mathit{Func})$$

$$\frac{\Gamma \vdash A \to B \ \text{Type} \qquad \Gamma \vdash f : A \to B \qquad \Gamma \vdash x : A}{\Gamma \vdash f\, x : B}(\mathit{App})$$

The current scope of our program.

The Program Context

$$\Gamma, \Delta ::= \diamond \mid \Gamma, x : A$$

The Program Context
The current scope of our program.
The Program Context
$\Gamma, \Delta ::= \diamond \mid \Gamma, x : A$

The Program Context

Our context called Γ or Δ, etc...

The Program Context

$$\boxed{\Gamma, \Delta} ::= \diamond \mid \Gamma, x : A$$

The Program Context

Our context called Γ or Δ, etc...

The Program Context

$\boxed{\Gamma, \Delta} ::= \diamond \mid \Gamma, x : A$

The Program Context

Is defined by:

The Program Context

$$\Gamma, \Delta \boxed{::=} \diamond \mid \Gamma, x : A$$

The Program Context
Is defined by:
The Program Context
$\Gamma, \Delta \boxed{::=} \diamond \mid \Gamma, x : A$

The Program Context

Empty (like the empty list) represented by $\diamond$.

The Program Context

$$\Gamma, \Delta ::= \boxed{\diamond} \mid \Gamma, x : A$$

The Program Context

Empty (like the empty list) represented by $\diamond$.

The Program Context

$\Gamma, \Delta ::= \boxed{\diamond} \mid \Gamma, x : A$

The Program Context

Or,

The Program Context

$$\Gamma, \Delta ::= \diamond \boxed{\mid} \Gamma, x : A$$

The Program Context

Or,

The Program Context

$\Gamma, \Delta ::= \diamond \boxed{\mid} \Gamma, x : A$

The Program Context

Our program scope, extended with a variable 'x' of type 'A'.

The Program Context

$$\Gamma, \Delta ::= \diamond \mid \boxed{\Gamma, x : A}$$

The Program Context

Our program scope, extended with a variable 'x' of type 'A'.

The Program Context

$\Gamma, \Delta ::= \diamond \mid \boxed{\Gamma, x : A}$

Example Contexts

What is the context of this program?

```
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```


Example Contexts

What is the context of this program?

-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True = False
not False = True

Example Contexts

What is the context of this program?

```
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```

Answer.

$$\Gamma = ???$$

Example Contexts

What is the context of this program?

```
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```

Answer.

Γ =???

Example Contexts

What is the context of this program?

```
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```

Answer.

$$\Gamma = \diamond, \ldots$$

Example Contexts

What is the context of this program?

-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True = False
not False = True

Answer.

Γ = ◇, …

What is the context of this program?

```
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```

Answer.

$$\Gamma = \diamond, A \to B : \mathrm{Type}, \ldots$$

Example Contexts

What is the context of this program?

```
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```

Answer.

$\Gamma = \diamond, A \to B : \mathrm{Type}, \ldots$

Example Contexts

What is the context of this program?

```
-- assuming ⋄ empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```

Answer.

$$\Gamma = \diamond, A \to B : \text{Type}, \text{Bool} : \text{Type}, \text{True} : \text{Bool}, \text{False} : \text{Bool}, \ldots$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└─Example Contexts

Example Contexts
What is the context of this program?
Answer.
Γ = ⋄, A → B : Type, Bool : Type, True : Bool, False : Bool, ...

Example Contexts

What is the context of this program?

```
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False

not : Bool -> Bool
not True  = False
not False = True
```

Answer.

$$\Gamma = \diamond, \mathsf{A} \to \mathsf{B} : \mathsf{Type}, \mathsf{Bool} : \mathsf{Type}, \mathsf{True} : \mathsf{Bool}, \mathsf{False} : \mathsf{Bool}, \mathsf{not} : \mathsf{Bool} \to \mathsf{Bool}$$

Example Contexts
What is the context of this program?
-- assuming   empty context here.
-- assuming A → B : Type here.
data Bool = True | False
not : Bool -> Bool
not True = False
not False = True
Answer.
Γ = ◇, A → B : Type, Bool : Type, True : Bool, False : Bool, not : Bool → Bool

The Function Type Formation Rule

If the derivations (true statements) hold above the line (premise), then the derivations hold below the line (conclusion).

The Type Formation Rule

$$\frac{}{}\ (\textit{Ty-Arrow})$$

The Function Type Formation Rule
If the derivations (true statements) hold above the line (premise), then the derivations hold below the line (conclusion).
The Type Formation Rule
(Ty-Arrow)

The Function Type Formation Rule

...from an arbitrary context, Γ

The Type Formation Rule

$$\frac{\boxed{\Gamma}\qquad\qquad\qquad}{\qquad\qquad\qquad\qquad}\,(\textit{Ty-Arrow})$$

The Function Type Formation Rule

...from an arbitrary context, Γ

The Type Formation Rule

$\boxed{\Gamma}$ (Ty-Arrow)

The Function Type Formation Rule

...we may derive (produce, obtain...)

The Type Formation Rule

$$\frac{\Gamma \boxed{\vdash}}{\qquad\qquad}\,(\textit{Ty-Arrow})$$

The Function Type Formation Rule
...we may derive (produce, obtain...)
The Type Formation Rule
Γ⊢ (Ty-Arrow)

The Function Type Formation Rule

...an arbitrary type, called A

The Type Formation Rule

$$\frac{\Gamma \vdash \boxed{A \text{ Type}}}{\qquad\qquad\qquad}\,(\mathit{Ty\text{-}Arrow})$$

└─The Function Type Formation Rule

The Function Type Formation Rule
...an arbitrary type, called A
The Type Formation Rule
$\Gamma \vdash \boxed{A \text{ Type}}$ $(\mathit{Ty\text{-}Arrow})$

The Function Type Formation Rule

AND

The Type Formation Rule

$$\frac{\Gamma \vdash A \ \text{Type} \quad \boxed{}}{} \ (\textit{Ty-Arrow})$$

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└─The Function Type Formation Rule

The Function Type Formation Rule

AND

The Type Formation Rule

Γ ⊢ A Type (Ty-Arrow)

The Function Type Formation Rule

...From the same arbitrary context Γ, extended with a variable 'x' of type A

The Type Formation Rule

$$\frac{\Gamma \vdash A \text{ Type} \qquad \boxed{\Gamma, x : A}}{}\,(\mathit{Ty\text{-}Arrow})$$

The Function Type Formation Rule

...From the same arbitrary context Γ, extended with a variable 'x' of type A

The Type Formation Rule

$\Gamma \vdash A$ Type $\boxed{\Gamma, x : A}$ (Ty-Arrow)

...we may derive

The Type Formation Rule

$$\frac{\Gamma \vdash A \ \mathrm{Type} \qquad \Gamma, x : A \boxed{\vdash}}{} \,(\mathit{Ty\text{-}Arrow})$$

The Function Type Formation Rule

...we may derive

The Type Formation Rule

$$\frac{\Gamma \vdash A \ \mathrm{Type} \qquad \Gamma, x : A \boxed{\vdash}}{} \,(\mathit{Ty\text{-}Arrow})$$

The Function Type Formation Rule

some arbitrary type B.

The Type Formation Rule

$$\frac{\Gamma \vdash A \text{ Type} \qquad \Gamma, x : A \vdash \boxed{B \text{ Type}}}{}(\textit{Ty-Arrow})$$

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

2025-02-11

└─The Function Type Formation Rule

The Function Type Formation Rule

some arbitrary type B.

The Type Formation Rule

$\Gamma \vdash A$ Type $\quad \Gamma, x : A \vdash \boxed{B \text{ Type}}$ (Ty-Arrow)

The Function Type Formation Rule

Then, from that arbitrary Γ (program scope)

The Type Formation Rule

$$\frac{\Gamma \vdash A \text{ Type} \qquad \Gamma, x : A \vdash B \text{ Type}}{\boxed{\Gamma}}(\mathit{Ty\text{-}Arrow})$$

The Function Type Formation Rule
Then, from that arbitrary Γ (program scope)
The Type Formation Rule
$$\frac{\Gamma \vdash A \text{ Type} \qquad \Gamma, x : A \vdash B \text{ Type}}{\boxed{\Gamma}}(\mathit{Ty\text{-}Arrow})$$

The Function Type Formation Rule

...we may derive

The Type Formation Rule

$$\frac{\Gamma \vdash A \text{ Type} \qquad \Gamma, x : A \vdash B \text{ Type}}{\Gamma \boxed{\vdash}}(\textit{Ty-Arrow})$$

The Function Type Formation Rule

...we may derive

The Type Formation Rule

$$\frac{\Gamma \vdash A \text{ Type} \qquad \Gamma, x : A \vdash B \text{ Type}}{\Gamma \boxed{\vdash}}(\textit{Ty-Arrow})$$

The Function Type Formation Rule

...an arrow type between them

The Type Formation Rule

$$\frac{\Gamma \vdash A\ \text{Type} \qquad \Gamma, x : A \vdash B\ \text{Type}}{\Gamma \vdash \boxed{A \to B}}(\mathit{Ty\text{-}Arrow})$$

The Function Type Formation Rule

...an arrow type between them

The Type Formation Rule

$$\frac{\Gamma \vdash A\ \text{Type} \qquad \Gamma, x : A \vdash B\ \text{Type}}{\Gamma \vdash \boxed{A \to B}}(\mathit{Ty\text{-}Arrow})$$

The Function Type Formation Rule

...that is also a type.

The Type Formation Rule

$$\frac{\Gamma \vdash A\ \text{Type} \qquad \Gamma, x : A \vdash B\ \text{Type}}{\Gamma \vdash A \to B\ \boxed{\text{Type}}}(\textit{Ty-Arrow})$$

The Function Type Formation Rule

...that is also a type.

The Type Formation Rule

$$\frac{\Gamma \vdash A\ \text{Type} \qquad \Gamma, x : A \vdash B\ \text{Type}}{\Gamma \vdash A \to B\ \boxed{\text{Type}}}(\textit{Ty-Arrow})$$

The Function Term Formation Rule

If the premises hold above, so the conclusion holds below.

The Function Term Former

$$\frac{\qquad\qquad}{\qquad\qquad}\,(\mathit{Func})$$

The Function Term Formation Rule

If the premises hold above, so the conclusion holds below.

The Function Term Former

(Func)

The Function Term Formation Rule

From our program scope Γ we can obtain an arbitrary function type $A \to B$.

The Function Term Former

$$\frac{\boxed{\Gamma \vdash A \to B \text{ Type}}}{}\,(\mathit{Func})$$

The Function Term Formation Rule
From our program scope Γ we can obtain an arbitrary function type $A \to B$.
The Function Term Former
$\Gamma \vdash A \to B$ Type (Func)

The Function Term Formation Rule

The Function Term Former

$$\frac{\Gamma \vdash A \to B \ \text{Type} \ \boxed{}}{} \, (\mathit{Func})$$

The Function Term Formation Rule

The Function Term Former

$$\frac{\Gamma \vdash A \to B \ \text{Type} \ \boxed{}}{} \, (\mathit{Func})$$

The Function Term Formation Rule

The Function Term Former

$$\frac{\Gamma \vdash A \to B \; \text{Type} \qquad \boxed{\Gamma, x : A \vdash M : B}}{}\,(\mathit{Func})$$

The Function Term Formation Rule
The Function Term Former
$\Gamma \vdash A \to B$ Type $\boxed{\Gamma, x : A \vdash M : B}$ (Func)

The Function Term Formation Rule

The Function Term Former

$$\frac{\Gamma \vdash A \to B \text{ Type} \qquad \Gamma, x : A \vdash M : B}{\boxed{\Gamma \vdash \lambda(x : A).M : A \to B}}(\mathit{Func})$$

The Function Term Formation Rule
The Function Term Former
$$\frac{\Gamma \vdash A \to B \text{ Type} \qquad \Gamma, x : A \vdash M : B}{\boxed{\Gamma \vdash \lambda(x : A).M : A \to B}}(\mathit{Func})$$

The Function Term Elimination Rule

The Function Term Elimination Rule

STLC: The Simply Typed Lambda Calculus

STLC: The Simply Typed Lambda Calculus

STLC 2

STLC 2

α-Equivalence/Conversion.

α-Equivalence/Conversion.

Substitution and Computation

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

Substitution and Computation

Alpha Equivalence, Free Variables, Beta Reduction

Alpha Equivalence, Free Variables, Beta Reduction

De-Bruijn Indices

De-Bruijn Indices

Lift-and-shift

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

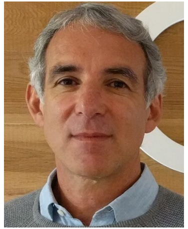
└─Lift-and-shift

A lot of work at the meta-level

└─A lot of work at the meta-level

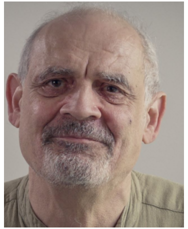
A lot of work at the meta-level

Explicit Substitutions (Paper)



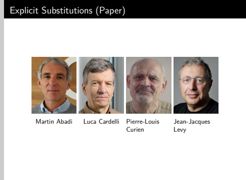
Martin Abadi

Luca Cardelli



Pierre-Louis Curien

Jean-Jacques Levy



Extending the STLC with Explicit Substitutions

Extending the STLC with Explicit Substitutions

Syntax cont.

Syntax cont.

ES rules.

ES rules.

Evaluating ES

Evaluating ES

Evaluating ES 2

Evaluating ES 2

References

2025-02-11

Lambda Calculi With Explicit Substitutions slides: https://github.com/donovancrichton/Talks

└─References

References