Substitution, Normalization, and Formalization

Marcelo Fiore & Yulong Huang

University of Cambridge, UK

S-REPLS 16, Imperial College London

April 2025

M, the warm-up language

```
Syntax(M)

      M := c | M + M
                {- where c ∈ Σ, some alphabet of characters -}
```

M, the warm-up language

```
Syntax(M)
      M := c | M + M
                {- where c ∈ Σ, some alphabet of characters -}

Equivalence(≡)
      (m + m') + m'' ≡ m + (m' + m'')
                {- _≡_ is reflexive, transitive, and symmetric -}
```

M, the warm-up language

```
Syntax(M)
      M := c | M + M
                {- where c ∈ Σ, some alphabet of characters -}

Equivalence(≡)
      (m + m') + m'' ≡ m + (m' + m'')
                {- _≡_ is reflexive, transitive, and symmetric -}

How to check if m ≡ m'?
```

N, the normal form of M

```
Syntax(N)
      N := c | N + c
```

N, the normal form of M

Syntax(N)

```
N := c | N + c
```

Observation: all expressions can be rearranged to left-associative form.

Decide m ≡ m': Re-arrange m and m' into left-associative form, then compare them syntactically.

Normalization of M

Normalization of M

```
nf : M → N
```

Normalization of M

```
nf : M → N
nf c = c
```

Normalization of M

```
nf : M → N
nf c = c
nf (m + m') =
```

Normalization of M

```
nf : M → N
nf c = c
nf (m + m') = (nf m) ⊞ (nf m')

  where
    _⊞_ : N → N → N
```

Normalization of M

```
nf : M → N
nf c = c
nf (m + m') = (nf m) ⊞ (nf m')

  where
    _⊞_ : N → N → N
    n ⊞ c = n + c
```

Normalization of M

```
nf : M → N
nf c = c
nf (m + m') = (nf m) ⊞ (nf m')

  where
    _⊞_ : N → N → N
    n ⊞ c = n + c
    n ⊞ (n' + c) = (n ⊞ n') + c
```

Simply typed lambda calculus (STLC)

```
Syntax(STLC)
      τ := θ | τ ⇒ τ
      t := x | λx:τ.t | t t
      Γ := · | Γ, x:τ
                        {- where θ is some base type -}
```

STLC typing

Typing(⊢)

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \text{Var} \qquad \frac{\Gamma,\ x{:}\tau \vdash t : \tau'}{\Gamma \vdash \lambda x{:}\tau.t : \tau \Rightarrow \tau'}\ \text{Lam}$$

$$\frac{\Gamma \vdash t : \tau \Rightarrow \tau' \qquad \Gamma \vdash t' : \tau}{\Gamma \vdash t\ t' : \tau'}\ \text{App}$$

STLC equivalence

```
Equivalence(≡)
```

$$\frac{\Gamma,\ x{:}\tau \vdash t : \tau' \qquad \Gamma \vdash t' : \tau}{\Gamma \vdash (\lambda x{:}\tau.t)\ t' \equiv t\ [\ t'\ /\ x\ ] : \tau'}\ \beta$$

```
          {- t [ t' / x ] is the capture-avoiding substitution -}
```

$$\frac{\Gamma \vdash t : \tau \Rightarrow \tau'}{\Gamma \vdash\ \ t \equiv (\lambda x{:}\tau.t\ x) : \tau \Rightarrow \tau'}\ \eta$$

```
          {- _≡_ also has congruence under lambda and application -}
```

Normal forms for STLC

Decide t ≡ t': Normalize t and t' into a suitable normal form, then compare them syntactically.

But which normal form to use?

Normal forms for STLC

Decide $t \equiv t'$: Normalize t and t' into a suitable normal form, then compare them syntactically.

But which normal form to use?

β-short form: terms without any β-redex like $(\lambda x{:}\tau.t)\ t'$.

Problem: $f \equiv (\lambda x{:}\tau.f\ x) : \tau \Rightarrow \tau'$, η-equivalence for open terms.

Normal forms for STLC

β-short η-long form: terms without any β-redex, and functions are fully η-expanded.

Normal forms for STLC

β-short η-long form: terms without any β-redex, and functions are fully η-expanded.

For example, $g : (\theta \Rightarrow \theta) \Rightarrow \theta \Rightarrow \theta$ expands to:

$$\lambda f{:}\theta \Rightarrow \theta.\ \lambda x{:}\theta.\ g\ (\lambda y{:}\theta.f\ y)\ x$$

Note that we also expanded $f : \theta \Rightarrow \theta$.

Normal forms for STLC

β-short η-long form: terms without any β-redex, and functions are fully η-expanded.

For example, g : (θ ⇒ θ) ⇒ θ ⇒ θ expands to:

```
λf:θ ⇒ θ. λx:θ. g (λy:θ.f y) x
```

Note that we also expanded f : θ ⇒ θ.

Property: normal forms of type τ ⇒ τ' is always in the form of λx:τ.n, where n is a normal form of type τ'.

Normalization of STLC

Normalization of STLC

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
```

```
Notation: t : Tm Γ τ means that Γ ⊢ t : τ.
```

Normalization of STLC

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
nf x = η-exp x
```

Notation: t : Tm Γ τ means that Γ ⊢ t : τ.

Normalization of STLC

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
nf x = η-exp x
nf (λx:τ.t) = λx:τ. (nf t)
```

Notation: `t : Tm Γ τ` means that `Γ ⊢ t : τ`.

Normalization of STLC

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
nf x = η-exp x
nf (λx:τ.t) = λx:τ. (nf t)
nf (t t') = (nf t) $ (nf t')
```

```
Notation: t : Tm Γ τ means that Γ ⊢ t : τ.
```

Normalization of STLC

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
nf x = η-exp x
nf (λx:τ.t) = λx:τ. (nf t)
nf (t t') = (nf t) $ (nf t')

  where
    _$_ : ∀{Γ τ τ'} → Nf Γ (τ ⇒ τ') → Nf Γ τ → Nf Γ τ'
    (λx:τ.t) $ t' = t [ t' / x ]
```

Notation: `t : Tm Γ τ` means that `Γ ⊢ t : τ`.

Normalization of STLC

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
nf x = η-exp x
nf (λx:τ.t) = λx:τ. (nf t)
nf (t t') = (nf t) $ (nf t')

  where
    _$_ : ∀{Γ τ τ'} → Nf Γ (τ ⇒ τ') → Nf Γ τ → Nf Γ τ'
    (λx:τ.t) $ t' = !! t [ t' / x ] !!
```

But substitution does not preserve normal forms!

Hereditary substitution

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
nf x = η-exp x
nf (λx:τ.t) = λx:τ. (nf t)
nf (t t') = (nf t) $ (nf t')

  where
    _$_ : ∀{Γ τ τ'} → Nf Γ (τ ⇒ τ') → Nf Γ τ → Nf Γ τ'
    (λx:τ.t) $ t' = t [ t' / x ]he
```

Solution: define a hereditary substitution that keeps reducing the occurred β-redexes.

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h) = λh. nf(g h)
```

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h) = λh. nf(g h)
                 = λh. (nf g) $ (nf h)
```

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h) = λh. nf(g h)
                 = λh. (nf g) $ (nf h)
                 = λh. (η-exp g) $ (η-exp h)
```

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h) = λh. nf(g h)
                 = λh. (nf g) $ (nf h)
                 = λh. (η-exp g) $ (η-exp h)
                 = λh. (λf x. g (λz. f z) x) $ (λy. h y)
```

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h) = λh. nf(g h)
                 = λh. (nf g) $ (nf h)
                 = λh. (η-exp g) $ (η-exp h)
                 = λh. (λf x. g (λz. f z) x) $ (λy. h y)
                 = λh. (λx. g (λz. f z) x) [ (λy. h y) / f ]he
```

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h) = λh. nf(g h)
                 = λh. (nf g) $ (nf h)
                 = λh. (η-exp g) $ (η-exp h)
                 = λh. (λf x. g (λz. f z) x) $ (λy. h y)
                 = λh. (λx. g (λz. f z) x) [ (λy. h y) / f ]he
                 = λh. λx. g (λz. (λy. h y) z) x
                                    {- hereditary substitution triggered -}
```

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h) = λh. nf(g h)
                 = λh. (nf g) $ (nf h)
                 = λh. (η-exp g) $ (η-exp h)
                 = λh. (λf x. g (λz. f z) x) $ (λy. h y)
                 = λh. (λx. g (λz. f z) x) [ (λy. h y) / f ]he
                 = λh. λx. g (λz. (λy. h y) z) x
                 = λh. λx. g (λz. h z) x
```

Example

```
Have g : (θ ⇒ θ) ⇒ θ ⇒ θ.

    nf (λh. g h)   = λh. λx. g (λz. h z) x
```

Example

Have $g : (\theta \Rightarrow \theta) \Rightarrow \theta \Rightarrow \theta$.

$\mathsf{nf}\ (\lambda h.\ g\ h) \quad = \lambda h.\ \lambda x.\ g\ (\lambda z.\ h\ z)\ x$

$\mathsf{nf}\ g = \eta\text{-}\mathsf{exp}\ g = \lambda f.\ \lambda x.\ g\ (\lambda z.\ f\ z)\ x$

Indeed, $\lambda h.\ g\ h \equiv g$ by η.

Correctness?

```
Soundness:  λh. g h ≡ g ⟹ nf (λh. g h) = nf g

Completeness: g ≡ nf g = λf. λx. g (λz. f z) x

Uniqueness: emb n ≡ emb n' ⟹ n = n'
```

Correctness?

```
Soundness: t ≡ t' ⟹ nf t = nf t'

Completeness: t ≡ emb (nf t)

Uniqueness: emb n ≡ emb n' ⟹ n = n'
```

Algebraic story of M

```
Syntax(M)
      M := c | M + M
                {- where c ∈ Σ, some alphabet of characters -}

Equivalence(≡)
      (m + m') + m'' ≡ m + (m' + m'')
                {- _≡_ is reflexive, transitive, and symmetric -}
```

An algebra of M

Def. An M-Algebra $(X, \mathrm{char}, +, \equiv)$ is a set X with operators

$$\mathrm{char} : \Sigma \to X$$
$$+ \quad : X \to X \to X$$

and an equivalence relation $\equiv$ over X such that

$$\forall \{x\ x'\ x''\}.\ (x + x') + x'' \equiv x + (x' + x'').$$

M is an M-Algebra.

Initial algebra

Def. A homomorphism between M-Algebras $(X, \mathrm{char}, +, \equiv)$ and $(Y, \mathrm{char}', +', \equiv')$ is a function $f : X \to Y$ such that

$$
\begin{aligned}
f\ (\mathrm{char}\ c) &= \mathrm{char}'\ c \\
f\ (x + x') &= (f\ x) +' (f\ x') \\
x \equiv x' &\implies f\ x \equiv' f\ x'
\end{aligned}
$$

Initial algebra

Def. A homomorphism between M-Algebras (X, char, +, ≡) and (Y, char', +', ≡') is a function f : X → Y such that

```
f (char c) =  char' c
f (x + x') =  (f x) +' (f x')
x ≡ x'     ⟹  f x ≡' f x'
```

Def. (X, char, +, ≡) is initial if there exists a unique homomorphism from X to any other M-Algebra.

Initial algebra

Def. A homomorphism between M-Algebras (X, char, +, ≡) and (Y, char', +', ≡') is a function f : X → Y such that

$$
\begin{aligned}
f\ (\mathrm{char}\ c) &= \mathrm{char}'\ c \\
f\ (x + x') &= (f\ x) +' (f\ x') \\
x \equiv x' &\implies f\ x \equiv' f\ x'
\end{aligned}
$$

Def. (X, char, +, ≡) is initial if there exists a unique homomorphism from X to any other M-Algebra.

Thm. M is the initial M-Algebra.

Normal form as M-Alg

Since M is initial, for any (X, char', +', ≡') we have

```
ind : M → X
ind c = char' c
ind (m + m') = (ind m) +' (ind m')
```

such that m ≡ m' ⟹ (ind m) ≡' (ind m').

Normal form as *M*-Alg

Let N be the set of normal forms. If we have char' and +' such that (N, char', +', =) is an M-Algebra, then we have

```
nf : M → N
nf c = char' c
nf (m + m') = (nf m) +' (nf m')
```

such that $m \equiv m' \implies (nf\ m) = (nf\ m')$!

(Recall) Normalization of M

```
nf : M → N
nf c = c
nf (m + m') = (nf m) ⊞ (nf m')

  where
    _⊞_ : N → N → N
    n ⊞ c = n + c
    n ⊞ (n' + c) = (n ⊞ n') + c
```

Normal form as M-Alg

Let N be the set of normal forms. If we have ⊞ such that (x ⊞ x') + x'' = x ⊞ (x' ⊞ x''), then we have

```
nf : M → N
nf c = c
nf (m + m') = (nf m) ⊞ (nf m')
```

such that m ≡ m' ⟹ (nf m) = (nf m').

I.e. nf is sound if ⊞ is associative.

Isomorphism up to ≡

Def. M-Algebras $(X, \text{char}, +, \equiv)$ and $(Y, \text{char}', +', \equiv')$ are isomorphic up to their equivalences if there exist functions $f : X \to Y$ and $g : Y \to X$ such that

$$\forall x.\ g\ (f\ x) \equiv x$$
$$\forall y.\ f\ (g\ y) \equiv' y.$$

Note: This definition works for all sets with an equivalence relation.

Completeness and uniqueness

```
If (N, ⊞, =) is isomorphic to M up to equivalence, then
we have

   nf : M → N
   emb : N → M

such that  emb (nf m) ≡ m          {- completeness -}
           nf (emb n) = n.         {- implies uniqueness -}
```

Completeness and uniqueness

If $(N, \boxplus, =)$ is isomorphic to M up to equivalence, then we have

```
nf : M → N
emb : N → M
```

such that

```
emb (nf m) ≡ m          {- completeness -}
nf (emb n) = n.         {- implies uniqueness -}
```

$$\mathtt{emb\ n} \equiv \mathtt{emb\ n'} \implies \mathtt{nf\ (emb\ n)} = \mathtt{nf\ (emb\ n')} \implies \mathtt{n} = \mathtt{n'}$$

Embedding of N

```
emb : N → M
emb c = c
emb (n + c) = emb n + c
```

Note: emb (n ⊞ n') ≠ (emb n) + (emb n'), they are only equivalent, so emb is not a homomorphism!

Algebraic story of M

1. Define M-Algebra (X, char, +, ≡) where M is the initial algebra.

Algebraic story of M

1. Define M-Algebra (X, char, +, ≡) where M is the initial algebra.

2. Find N, char', +' such that they form an M-Algebra (N, char', +', =).

$\Longrightarrow$ have sound normalization algorithm nf.

Algebraic story of M

1. Define M-Algebra (X, char, +, ≡) where M is the initial algebra.

2. Find N, char', +' such that they form an M-Algebra (N, char', +', =).
   ⟹ have sound normalization algorithm nf.

3. Show that N is isomorphic to M up to equivalence.
   ⟹ nf is complete.
   ⟹ N has a unique normal forms.

STLC-Algebra

1. Define STLC-Algebra (X, var, lam, app, sub, ≡) where Tm is the initial algebra, following (Fiore 2002).

STLC-Algebra

1. Define STLC-Algebra (X, var, lam, app, sub, ≡) where Tm is the initial algebra, following (Fiore 2002).

2. Find Nf, var', lam', app', sub' such that they form an STLC-Algebra (Nf, var', lam', app', sub' =).

$\Longrightarrow$ have sound normalization algorithm nf.

STLC-Algebra

1. Define STLC-Algebra (X, var, lam, app, sub, ≡) where Tm is the initial algebra, following (Fiore 2002).

2. Find Nf, var', lam', app', sub' such that they form an STLC-Algebra (Nf, var', lam', app', sub' =).
   ⟹ have sound normalization algorithm nf.

3. Show that Nf is isomorphic to Tm up to equivalence.
   ⟹ nf is complete.
   ⟹ Tm has a unique normal forms.

(Recall) Hereditary substitution

```
nf : ∀{Γ τ} → Tm Γ τ → Nf Γ τ
nf x = η-exp x
nf (λx:τ.t) = λx:τ. (nf t)
nf (t t') = (nf t) $ (nf t')

  where
    _$_ : ∀{Γ τ τ'} → Nf Γ (τ ⇒ τ') → Nf Γ τ → Nf Γ τ'
    (λx:τ.t) $ t' = t [ t' / x ]he
```

Algebraic perspective at formalization

- Simpler, more principled proofs than previous work (Keller and Altenkirch 2010), only need induction.

Algebraic perspective at formalization

- Simpler, more principled proofs than previous work (Keller and Altenkirch 2010), only need induction.

- Termination: hereditary substitution in structural recursive form

Algebraic perspective at formalization

- Simpler, more principled proofs than previous work (Keller and Altenkirch 2010), only need induction.

- Termination: hereditary substitution in structural recursive form

- Proof goals: clear indication from the algebraic signature.
  E.g. t [t' / x]he = t when x is not free in t

Algebraic perspective at formalization

- Simpler, more principled proofs than previous work (Keller and Altenkirch 2010), only need induction.

- Termination: hereditary substitution in structural recursive form.

- Proof goals: clear indication from the algebraic signature.

```
  E.g. (t t') [ t'' / x ]he =
            (t [ t'' / x ]he) $ (t' [ t'' / x ]he)
```

Thank you!

…and questions?

Speaker email: yh419@cam.ac.uk

References

Fiore, Marcelo. "Semantic analysis of normalisation by evaluation for typed lambda calculus." Proceedings of the 4th ACM SIGPLAN international conference on Principles and practice of declarative programming. 2002.

Keller, Chantal, and Thorsten Altenkirch. "Hereditary substitutions for simple types, formalized." Proceedings of the third ACM SIGPLAN workshop on Mathematically structured functional programming. 2010.

Intrinsic syntax of terms

```
data Var : Con → Ty → Set where
  vz : ∀{Γ τ} → Var (Γ , τ) τ
  vs : ∀{Γ τ τ'} → Var Γ τ → Var (Γ , τ') τ

data Tm : Con → Ty → Set where
  var : ∀{Γ τ} → Var Γ τ → Tm Γ τ
  lam : ∀{Γ τ τ'} → Tm (Γ , τ) τ' → Tm Γ (τ ⇒ τ')
  app : ∀{Γ τ τ'} → Tm Γ (τ ⇒ τ') → Tm Γ τ → Tm Γ τ'
```

Intrinsic syntax of normal forms

```
mutual
  data Nf : Con → Ty → Set where
    lam : ∀{Γ τ τ'} → Nf (Γ , τ) τ' → Nf Γ (τ ⇒ τ')
    ↓   : ∀{Γ} → Ne Γ ι → Nf Γ ι

  data Ne : Con → Ty → Set where
    var : ∀{Γ τ} → Var Γ τ → Ne Γ τ
    app : ∀{Γ τ τ'} → Ne Γ (τ ⇒ τ') → Nf Γ τ → Ne Γ τ'
```

Colour scheme

Keywords
Comments
Parenthesis
Constructors
Functions
Sets and types
Highlights
Text
Expressions