Definable (types-as-)closures
in concurrent combinatory algebra

Fritz Obermeyer

Department of Mathematics
Carnegie-Mellon University

2007:03:27

Outline

Motivation: complexity-of-definition

What mathematical objects are definable?

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
$\longrightarrow$ As simple as its shortest description. (Kolmogorov)

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
$\longrightarrow$ As simple as its shortest description. (Kolmogorov)

When is a description simple?

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
$\longrightarrow$ As simple as its shortest description. (Kolmogorov)

When is a description simple?
$\longrightarrow$ Fewer and shorter symbols

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
$\longrightarrow$ As simple as its shortest description. (Kolmogorov)

When is a description simple?
$\longrightarrow$ Fewer and shorter symbols (parametrized)

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
$\longrightarrow$ As simple as its shortest description. (Kolmogorov)

When is a description simple?
$\longrightarrow$ Fewer and shorter symbols (parametrized)

How does complexity depend on language (parameters)?

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
$\longrightarrow$ As simple as its shortest description. (Kolmogorov)

When is a description simple?
$\longrightarrow$ Fewer and shorter symbols (parametrized)

How does complexity depend on language (parameters)?
Hmm... these are more than r.e. sets...

Motivation: complexity-of-definition

What mathematical objects are definable?
$\longrightarrow$ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
$\longrightarrow$ As simple as its shortest description. (Kolmogorov)

When is a description simple?
$\longrightarrow$ Fewer and shorter symbols (parametrized)

How does complexity depend on language (parameters)?
Hmm... these are more than r.e. sets...
these are weighted grammars or weighted presentations.

Motivation: complexity-of-definition

What mathematical objects are definable?
→ First, the constructive objects are definable,
say r.e. sets mod r.e. relations.

How simple is a given object?
→ As simple as its shortest description. (Kolmogorov)

When is a description simple?
→ Fewer and shorter symbols (parametrized)

How does complexity depend on language (parameters)?
Hmm... these are more than r.e. sets...
these are weighted grammars or weighted presentations.
Simpler grammars/signatures are simpler to parametrize.

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient?

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient? Terms need to:

- do nothing at all: λf. f

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient? Terms need to:

- do nothing at all: $\lambda \mathsf{f}.\ \mathsf{f}$
- move terms around: $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\mathsf{x}\ \mathsf{y})$, $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ (\mathsf{f}\ \mathsf{x})\mathsf{y}$

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient? Terms need to:

- do nothing at all: $\lambda \mathsf{f}.\ \mathsf{f}$
- move terms around: $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\mathsf{x}\ \mathsf{y})$, $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ (\mathsf{f}\ \mathsf{x})\mathsf{y}$
- project/ignore terms: $\lambda \mathsf{f}, \mathsf{x}.\ \mathsf{f}$

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient? Terms need to:

- do nothing at all: λf. f
- move terms around: λf, x, y. f(x y), λf, x, y. (f x)y
- project/ignore terms: λf, x. f
- copy terms: λf, x. f x x

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta{+}\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient? Terms need to:

- do nothing at all: $\lambda \mathsf{f}.\ \mathsf{f}$
- move terms around: $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\mathsf{x}\ \mathsf{y})$, $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ (\mathsf{f}\ \mathsf{x})\mathsf{y}$
- project/ignore terms: $\lambda \mathsf{f}, \mathsf{x}.\ \mathsf{f}$
- copy terms: $\lambda \mathsf{f}, \mathsf{x}.\ \mathsf{f}\ \mathsf{x}\ \mathsf{x}$
- (join terms: $\lambda \mathsf{f}, \mathsf{g}.\ \mathsf{f} \mid \mathsf{g}$?)

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient? Terms need to:

- do nothing at all: $\lambda \mathsf{f}.\ \mathsf{f}$
- move terms around: $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\mathsf{x}\ \mathsf{y})$, $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ (\mathsf{f}\ \mathsf{x})\mathsf{y}$
- project/ignore terms: $\lambda \mathsf{f}, \mathsf{x}.\ \mathsf{f}$
- copy terms: $\lambda \mathsf{f}, \mathsf{x}.\ \mathsf{f}\ \mathsf{x}\ \mathsf{x}$
- (join terms: $\lambda \mathsf{f}, \mathsf{g}.\ \mathsf{f} \mid \mathsf{g}$?)

That's enough!

Why untyped λ-calculus?

Church: One language for r.e. sets mod r.e. relations is...
λ-calculus mod an r.e. theory: β, $\beta+\eta$, etc.

Curry: λ-calculus has a complicated syntax, but
a few closed terms generate all others.

What basis is sufficient? Terms need to:

- do nothing at all: $\lambda \mathsf{f}.\ \mathsf{f}$
- move terms around: $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\mathsf{x}\ \mathsf{y})$, $\lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ (\mathsf{f}\ \mathsf{x})\mathsf{y}$
- project/ignore terms: $\lambda \mathsf{f}, \mathsf{x}.\ \mathsf{f}$
- copy terms: $\lambda \mathsf{f}, \mathsf{x}.\ \mathsf{f}\ \mathsf{x}\ \mathsf{x}$
- (join terms: $\lambda \mathsf{f}, \mathsf{g}.\ \mathsf{f} \mid \mathsf{g}$?)

That's enough! Many other bases would work.

Why semantics, extensionality?

How simple is a given object?

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also
$\longrightarrow$ should not depend on any one language

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also
$\longrightarrow$ should not depend on any one language

We want meaning, not syntax.

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also
$\longrightarrow$ should not depend on any one language

We want meaning, not syntax.
$\longrightarrow$ Coarser theories/quotients are better.

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also
$\longrightarrow$ should not depend on any one language

We want meaning, not syntax.
$\longrightarrow$ Coarser theories/quotients are better.
$\longrightarrow$ But "empty" / "undefined" should remain the same.

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also
$\longrightarrow$ should not depend on any one language

We want meaning, not syntax.
$\longrightarrow$ Coarser theories/quotients are better.
$\longrightarrow$ But "empty" / "undefined" should remain the same.

(1) Identify all "empty" / "undefined" terms (the theory $\mathcal{H}$)

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also
$\longrightarrow$ should not depend on any one language

We want meaning, not syntax.
$\longrightarrow$ Coarser theories/quotients are better.
$\longrightarrow$ But "empty" / "undefined" should remain the same.

(1) Identify all "empty" / "undefined" terms (the theory $\mathcal{H}$)

(2) Then identify as much as consistently possible ($\mathcal{H}^*$)

Why semantics, extensionality?

How simple is a given object?
$\longrightarrow$ should not depend on any one description, but also
$\longrightarrow$ should not depend on any one language

We want meaning, not syntax.
$\longrightarrow$ Coarser theories/quotients are better.
$\longrightarrow$ But "empty" / "undefined" should remain the same.

(1) Identify all "empty" / "undefined" terms (the theory $\mathcal{H}$)

(2) Then identify as much as consistently possible ($\mathcal{H}^*$)

From seqential λ-calculus, we get term fragment of $\mathcal{D}_\infty$.

Why semantics, extensionality?

How simple is a given object?
$\rightarrow$ should not depend on any one description, but also
$\rightarrow$ should not depend on any one language

We want meaning, not syntax.
$\rightarrow$ Coarser theories/quotients are better.
$\rightarrow$ But "empty" / "undefined" should remain the same.

(1) Identify all "empty" / "undefined" terms (the theory $\mathcal{H}$)

(2) Then identify as much as consistently possible ($\mathcal{H}^*$)

From seqential λ-calculus, we get term fragment of $\mathcal{D}_\infty$.
From concurrent λ-calculus, we get term+join fragment.

Why semantics, extensionality?

How simple is a given object?
$\rightarrow$ should not depend on any one description, but also
$\rightarrow$ should not depend on any one language

We want meaning, not syntax.
$\rightarrow$ Coarser theories/quotients are better.
$\rightarrow$ But "empty" / "undefined" should remain the same.

(1) Identify all "empty" / "undefined" terms (the theory $\mathcal{H}$)

(2) Then identify as much as consistently possible ($\mathcal{H}^*$)

From seqential λ-calculus, we get term fragment of $\mathcal{D}_\infty$.
From concurrent λ-calculus, we get term+join fragment.

But $\mathcal{H}^*$ is Π^0_2-complete, not r.e.; what about "constructive"?

Why semantics, extensionality?

How simple is a given object?
$\rightarrow$ should not depend on any one description, but also
$\rightarrow$ should not depend on any one language

We want meaning, not syntax.
$\rightarrow$ Coarser theories/quotients are better.
$\rightarrow$ But "empty" / "undefined" should remain the same.

(1) Identify all "empty" / "undefined" terms (the theory $\mathcal{H}$)

(2) Then identify as much as consistently possible ($\mathcal{H}^*$)

From seqential λ-calculus, we get term fragment of $\mathcal{D}_\infty$.
From concurrent λ-calculus, we get term+join fragment.

But $\mathcal{H}^*$ is Π^0_2-complete, not r.e.; what about "constructive"?
$\rightarrow$ Approximate $\mathcal{H}^*$ by r.e. theory, e.g., ZFC.

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?

$\rightarrow$ enumerations : nat $\rightarrow$ a,

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a,   intersection, union

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a, intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$,

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\longrightarrow$ enumerations : nat $\longrightarrow$ a, intersection, union
$\longrightarrow$ semipredicates : a $\longrightarrow \{\bot, \mathbf{I}\}$, intersection, no union

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a,   intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$,   intersection, no union

Disjunction is representable at meta-level (simulation)

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a,   intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$,   intersection, no union

Disjunction is representable at meta-level (simulation)
but this does not work for oracles.

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a, intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$, intersection, no union

Disjunction is representable at meta-level (simulation)
but this does not work for oracles.
Add join as a primitive: a $\rightarrow$ semi becomes a lattice,

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a,   intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$,   intersection, no union

Disjunction is representable at meta-level (simulation)
but this does not work for oracles.
Add join as a primitive: a $\rightarrow$ semi becomes a lattice,

Scott came from opposite direction:
Some top. spaces yield models of λ-calculus

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a,   intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$,   intersection, no union

Disjunction is representable at meta-level (simulation)
but this does not work for oracles.
Add join as a primitive: a $\rightarrow$ semi becomes a lattice,

Scott came from opposite direction:
Some top. spaces yield models of λ-calculus
and in these models join is of course definable.

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a,   intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$,   intersection, no union

Disjunction is representable at meta-level (simulation)
    but this does not work for oracles.
Add join as a primitive: a $\rightarrow$ semi becomes a lattice,

Scott came from opposite direction:
    Some top. spaces yield models of λ-calculus
    and in these models join is of course definable.
But $\mathcal{D}_\infty$,$\mathcal{P}\omega$ introduce extra junk, e.g. step functions.

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a, intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$, intersection, no union

Disjunction is representable at meta-level (simulation)
but this does not work for oracles.
Add join as a primitive: a $\rightarrow$ semi becomes a lattice,

Scott came from opposite direction:
Some top. spaces yield models of λ-calculus
and in these models join is of course definable.
But $\mathcal{D}_\infty$,$\mathcal{P}\omega$ introduce extra junk, e.g. step functions.

So... Consider pure fragement of $\mathcal{D}_\infty$:

Why concurrency (join)?

How are open/r.e. sets represented in λ-calc.?
$\rightarrow$ enumerations : nat $\rightarrow$ a, intersection, union
$\rightarrow$ semipredicates : a $\rightarrow \{\bot, \mathbf{I}\}$, intersection, no union

Disjunction is representable at meta-level (simulation)
but this does not work for oracles.
Add join as a primitive: a $\rightarrow$ semi becomes a lattice,

Scott came from opposite direction:
Some top. spaces yield models of λ-calculus
and in these models join is of course definable.
But $\mathcal{D}_\infty$,$\mathcal{P}\omega$ introduce extra junk, e.g. step functions.

So... Consider pure fragement of $\mathcal{D}_\infty$:
$=$ concurrent combinatory algebra, mod $\mathcal{H}^*$

Combinatory algebra with join

Combinatory algebra: equational, $\mathbf{S}$ and $\mathbf{K}$ for abstraction

$$\mathbf{K}\ \mathsf{x}\ \mathsf{y} = \mathsf{x} \qquad\qquad \mathbf{S}\ \mathsf{x}\ \mathsf{y}\ \mathsf{z} = \mathsf{x}\ \mathsf{z}(\mathsf{y}\ \mathsf{z})$$

Combinatory algebra with join

Combinatory algebra: equational, $\mathbf{S}$ and $\mathbf{K}$ for abstraction

$$\mathbf{K}\ \mathsf{x}\ \mathsf{y} = \mathsf{x} \qquad\qquad \mathbf{S}\ \mathsf{x}\ \mathsf{y}\ \mathsf{z} = \mathsf{x}\ \mathsf{z}(\mathsf{y}\ \mathsf{z})$$

Concurrent CA: partially ordered, also $\mathbf{J}$ for join

$$\mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsupseteq \mathsf{x} \qquad \mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsupseteq \mathsf{y} \qquad \frac{\mathsf{x} \sqsubseteq \mathsf{z} \qquad \mathsf{y} \sqsubseteq \mathsf{z}}{\mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsubseteq \mathsf{z}}$$

Combinatory algebra with join

Combinatory algebra: equational, $\mathbf{S}$ and $\mathbf{K}$ for abstraction

$$\mathbf{K}\ \mathsf{x}\ \mathsf{y} = \mathsf{x} \qquad \mathbf{S}\ \mathsf{x}\ \mathsf{y}\ \mathsf{z} = \mathsf{x}\ \mathsf{z}(\mathsf{y}\ \mathsf{z})$$

Concurrent CA: partially ordered, also $\mathbf{J}$ for join

$$\mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsupseteq \mathsf{x} \qquad \mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsupseteq \mathsf{y} \qquad \frac{\mathsf{x} \sqsubseteq \mathsf{z} \quad \mathsf{y} \sqsubseteq \mathsf{z}}{\mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsubseteq \mathsf{z}}$$

In either case, add $\top$ for error: $\top\ \mathsf{x} = \mathsf{x}$, or $\top \sqsupseteq \mathsf{x}$

Combinatory algebra with join

Combinatory algebra: equational, $\mathbf{S}$ and $\mathbf{K}$ for abstraction

$$\mathbf{K}\ \mathsf{x}\ \mathsf{y} = \mathsf{x} \qquad\qquad \mathbf{S}\ \mathsf{x}\ \mathsf{y}\ \mathsf{z} = \mathsf{x}\ \mathsf{z}(\mathsf{y}\ \mathsf{z})$$

Concurrent CA: partially ordered, also $\mathbf{J}$ for join

$$\mathbf{J}\ \mathsf{x}\ \mathsf{y}\ \sqsupseteq\ \mathsf{x} \qquad \mathbf{J}\ \mathsf{x}\ \mathsf{y}\ \sqsupseteq\ \mathsf{y} \qquad \frac{\mathsf{x}\ \sqsubseteq\ \mathsf{z} \qquad \mathsf{y}\ \sqsubseteq\ \mathsf{z}}{\mathbf{J}\ \mathsf{x}\ \mathsf{y}\ \sqsubseteq\ \mathsf{z}}$$

In either case, add $\top$ for error: $\top\ \mathsf{x} = \mathsf{x}$, or $\top \sqsupseteq \mathsf{x}$

Translation from λ-calculus

$$\begin{aligned} [\![\lambda \mathsf{x}.\mathsf{M}]\!] &= \mathbf{K}\ \mathsf{M} && \mathsf{x}\ \textit{not free in}\ \mathsf{M} \\ [\![\lambda \mathsf{x}.\mathsf{M}\ \mathsf{N}]\!] &= \mathbf{S}[\![\lambda \mathsf{x}.\mathsf{M}]\!][\![\lambda \mathsf{x}.\mathsf{N}]\!] \end{aligned}$$

Combinatory algebra with join

Combinatory algebra: equational, $\mathbf{S}$ and $\mathbf{K}$ for abstraction

$$\mathbf{K}\ \mathsf{x}\ \mathsf{y} = \mathsf{x} \qquad\qquad \mathbf{S}\ \mathsf{x}\ \mathsf{y}\ \mathsf{z} = \mathsf{x}\ \mathsf{z}(\mathsf{y}\ \mathsf{z})$$

Concurrent CA: partially ordered, also $\mathbf{J}$ for join

$$\mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsupseteq \mathsf{x} \qquad \mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsupseteq \mathsf{y} \qquad \frac{\mathsf{x} \sqsubseteq \mathsf{z} \quad \mathsf{y} \sqsubseteq \mathsf{z}}{\mathbf{J}\ \mathsf{x}\ \mathsf{y} \sqsubseteq \mathsf{z}}$$

In either case, add $\top$ for error: $\top\ \mathsf{x} = \mathsf{x}$, or $\top \sqsupseteq \mathsf{x}$

Translation from λ-calculus with join

$$\begin{aligned} [\![\lambda \mathsf{x}.\mathsf{M}]\!] &= \mathbf{K}\ \mathsf{M} && \mathsf{x}\ \textit{not free in}\ \mathsf{M} \\ [\![\lambda \mathsf{x}.\mathsf{M}\ \mathsf{N}]\!] &= \mathbf{S}[\![\lambda \mathsf{x}.\mathsf{M}]\!][\![\lambda \mathsf{x}.\mathsf{N}]\!] \\ [\![\mathsf{M} \mid \mathsf{N}]\!] &= \mathbf{J}[\![\mathsf{M}]\!][\![\mathsf{N}]\!] \end{aligned}$$

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x\ \top^{\sim n} = \top$.

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x \top^{\sim n} = \top$.

Definition

(Scott's information ordering)
$\mathcal{H}^* \vdash x \sqsubseteq y$ iff $\forall M \in \langle \mathbf{S}, \mathbf{K} \rangle.$ M x conv $\implies$ M y conv.

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x\ \top^{\sim n} = \top$.

Definition

(Scott's information ordering)
$\mathcal{H}^* \vdash x \sqsubseteq y$ iff $\forall M \in \langle \mathbf{S}, \mathbf{K} \rangle.\ M\ x\ \text{conv} \implies M\ y\ \text{conv}$.

Consider the term model $\mathcal{B}$ mod $\mathcal{H}^*$,

$$\frac{}{\mathbf{S}, \mathbf{K}, \mathbf{J} \in \mathcal{B}} \qquad \frac{x \in \mathcal{B} \qquad y \in \mathcal{B}}{(x\ y) \in \mathcal{B}}$$

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x\ \top^{\sim n} = \top$.

Definition

(Scott's information ordering)
$\mathcal{H}^* \vdash x \sqsubseteq y$ iff $\forall M \in \langle \mathbf{S}, \mathbf{K} \rangle.\ M\ x \text{ conv} \implies M\ y \text{ conv}.$

Consider the term model $\mathcal{B}$ mod $\mathcal{H}^*$, with arbirary joins

$$\frac{}{\mathbf{S}, \mathbf{K}, \mathbf{J} \in \mathcal{B}} \qquad \frac{x \in \mathcal{B} \qquad y \in \mathcal{B}}{(x\ y) \in \mathcal{B}} \qquad \frac{X \subseteq \mathcal{B}}{\bigsqcup X \in \mathcal{B}}$$

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x\ \top^{\sim n} = \top$.

Definition

(Scott's information ordering)
$\mathcal{H}^* \vdash x \sqsubseteq y$ iff $\forall M \in \langle \mathbf{S}, \mathbf{K} \rangle.$ M x conv $\implies$ M y conv.

Consider the term model $\mathcal{B}$ mod $\mathcal{H}^*$, with arbirary joins

$$\frac{}{\mathbf{S}, \mathbf{K}, \mathbf{J} \in \mathcal{B}} \qquad \frac{x \in \mathcal{B} \qquad y \in \mathcal{B}}{(x\ y) \in \mathcal{B}} \qquad \frac{X \subseteq \mathcal{B}}{\bigsqcup X \in \mathcal{B}}$$

Theorem

$\mathcal{B}$ is an algebraic lattice

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x\ \top^{\sim n} = \top$.

Definition

(Scott's information ordering)
$\mathcal{H}^* \vdash x \sqsubseteq y$ iff $\forall M \in \langle \mathbf{S}, \mathbf{K} \rangle.\ M\ x\ \text{conv} \implies M\ y\ \text{conv}$.

Consider the term model $\mathcal{B}$ mod $\mathcal{H}^*$, with arbirary joins

$$\frac{}{\mathbf{S}, \mathbf{K}, \mathbf{J} \in \mathcal{B}} \qquad \frac{x \in \mathcal{B} \qquad y \in \mathcal{B}}{(x\ y) \in \mathcal{B}} \qquad \frac{X \subseteq \mathcal{B}}{\bigsqcup X \in \mathcal{B}}$$

Theorem

$\mathcal{B}$ is an algebraic lattice with join $\mathbf{J} = \mathbf{K} \mid \mathbf{S}\ \mathbf{K}$,

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x\ \top^{\sim n} = \top$.

Definition

(Scott's information ordering)

$\mathcal{H}^* \vdash x \sqsubseteq y$ iff $\forall M \in \langle \mathbf{S}, \mathbf{K} \rangle.$ M x conv $\implies$ M y conv.

Consider the term model $\mathcal{B}$ mod $\mathcal{H}^*$, with arbirary joins

$$\frac{}{\mathbf{S}, \mathbf{K}, \mathbf{J} \in \mathcal{B}} \qquad \frac{x \in \mathcal{B} \qquad y \in \mathcal{B}}{(x\ y) \in \mathcal{B}} \qquad \frac{X \subseteq \mathcal{B}}{\bigsqcup X \in \mathcal{B}}$$

Theorem

$\mathcal{B}$ is an algebraic lattice with join $\mathbf{J} = \mathbf{K} \mid \mathbf{S}\ \mathbf{K}$,

bottom $\bot = \mathbf{Y}\ \mathbf{K}$,

The completed term model

Definition

A term x converges iff $\exists n \in \mathbb{N}.\ x\ \top^{\sim n} = \top$.

Definition

(Scott's information ordering)
$\mathcal{H}^* \vdash x \sqsubseteq y$ iff $\forall M \in \langle \mathbf{S}, \mathbf{K} \rangle.$ M x conv $\implies$ M y conv.

Consider the term model $\mathcal{B}$ mod $\mathcal{H}^*$, with arbirary joins

$$\frac{}{\mathbf{S}, \mathbf{K}, \mathbf{J} \in \mathcal{B}} \qquad \frac{x \in \mathcal{B} \qquad y \in \mathcal{B}}{(x\ y) \in \mathcal{B}} \qquad \frac{X \subseteq \mathcal{B}}{\bigsqcup X \in \mathcal{B}}$$

Theorem

$\mathcal{B}$ is an algebraic lattice with join $\mathbf{J} = \mathbf{K} \mid \mathbf{S}\ \mathbf{K}$, bottom $\bot = \mathbf{Y}\ \mathbf{K}$, and top $\top = \mathbf{Y}\ \mathbf{J}$.

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\text{a type} \iff \text{a}\circ\text{a} = \text{a}$$

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$
\begin{aligned}
\mathsf{a}\ \text{type} &\iff \mathsf{a}\circ\mathsf{a} = \mathsf{a} \\
\mathsf{x}:\mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x}
\end{aligned}
$$

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a}\circ\mathsf{a} = \mathsf{a} \\ \mathsf{x}:\mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f}:\mathsf{a}\to\mathsf{b} &\iff \mathsf{b}\circ\mathsf{f} = \mathsf{f} = \mathsf{f}\circ\mathsf{a} \end{aligned}$$

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \\ \mathsf{x} : \mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f} : \mathsf{a} \to \mathsf{b} &\iff \mathsf{b} \circ \mathsf{f} = \mathsf{f} = \mathsf{f} \circ \mathsf{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \\ \mathsf{x} : \mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f} : \mathsf{a} \to \mathsf{b} &\iff \mathsf{b} \circ \mathsf{f} = \mathsf{f} = \mathsf{f} \circ \mathsf{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$
\begin{aligned}
\mathsf{a}\ \text{type} &\iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \\
\mathsf{x} : \mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\
\mathsf{f} : \mathsf{a} \to \mathsf{b} &\iff \mathsf{b} \circ \mathsf{f} = \mathsf{f} = \mathsf{f} \circ \mathsf{a}
\end{aligned}
$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\Longrightarrow$ No booleans,

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \\ \mathsf{x} : \mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f} : \mathsf{a} \to \mathsf{b} &\iff \mathsf{b} \circ \mathsf{f} = \mathsf{f} = \mathsf{f} \circ \mathsf{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\Longrightarrow$ No booleans, $\Longrightarrow$ no numerals.

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \\ \mathsf{x} : \mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f} : \mathsf{a} \to \mathsf{b} &\iff \mathsf{b} \circ \mathsf{f} = \mathsf{f} = \mathsf{f} \circ \mathsf{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\implies$ No booleans, $\implies$ no numerals.

How many denotations of not : $\mathbf{K}, \mathbf{F} \mapsto \mathbf{F}, \mathbf{K}$?

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \\ \mathsf{x} : \mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f} : \mathsf{a} \to \mathsf{b} &\iff \mathsf{b} \circ \mathsf{f} = \mathsf{f} = \mathsf{f} \circ \mathsf{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\implies$ No booleans, $\implies$ no numerals.

How many denotations of not : $\mathbf{K}, \mathbf{F} \mapsto \mathbf{F}, \mathbf{K}$?
Infinitely many.

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{array}{rcl} \mathsf{a}\ \text{type} & \iff & \mathsf{a} \circ \mathsf{a} = \mathsf{a} \\ \mathsf{x} : \mathsf{a} & \iff & \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f} : \mathsf{a} \to \mathsf{b} & \iff & \mathsf{b} \circ \mathsf{f} = \mathsf{f} = \mathsf{f} \circ \mathsf{a} \end{array}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\implies$ No booleans, $\implies$ no numerals.

How many denotations of not : $\mathbf{K}, \mathbf{F} \mapsto \mathbf{F}, \mathbf{K}$?
Infinitely many.

Is there a solution to and x y = and y x ?

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a}\circ\mathsf{a} = \mathsf{a} \\ \mathsf{x}:\mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f}:\mathsf{a}\to\mathsf{b} &\iff \mathsf{b}\circ\mathsf{f} = \mathsf{f} = \mathsf{f}\circ\mathsf{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\implies$ No booleans, $\implies$ no numerals.

How many denotations of not : $\mathbf{K}, \mathbf{F} \mapsto \mathbf{F}, \mathbf{K}$?
Infinitely many.

Is there a solution to and x y = and y x ? No.

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \mathsf{a}\ \text{type} &\iff \mathsf{a}\circ\mathsf{a} = \mathsf{a} \\ \mathsf{x}:\mathsf{a} &\iff \mathsf{a}\ \mathsf{x} = \mathsf{x} \\ \mathsf{f}:\mathsf{a}\to\mathsf{b} &\iff \mathsf{b}\circ\mathsf{f} = \mathsf{f} = \mathsf{f}\circ\mathsf{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\Longrightarrow$ No booleans, $\Longrightarrow$ no numerals.

How many denotations of $\mathsf{not} : \mathbf{K}, \mathbf{F} \mapsto \mathbf{F}, \mathbf{K}$?
Infinitely many.

Is there a solution to $\mathsf{and}\ \mathsf{x}\ \mathsf{y} = \mathsf{and}\ \mathsf{y}\ \mathsf{x}$? No.

Semantically typed λ-calculus (sequential)

Types as idempotents: unityped $\longrightarrow$ typed

$$\begin{aligned} \text{a type} &\iff \text{a}\circ\text{a} = \text{a} \\ \text{x}:\text{a} &\iff \text{a x} = \text{x} \\ \text{f}:\text{a}\to\text{b} &\iff \text{b}\circ\text{f} = \text{f} = \text{f}\circ\text{a} \end{aligned}$$

Is this really typed, fully abstract? Not usefully.

Range property: every range is a singleton or infinite.
$\implies$ No booleans, $\implies$ no numerals.

How many denotations of not : $\mathbf{K}, \mathbf{F} \mapsto \mathbf{F}, \mathbf{K}$?
Infinitely many.

Is there a solution to and x y = and y x ? No.

Semantically typed λ-calculus (concurrent)

Types as closures

Semantically typed λ-calculus (concurrent)

Types as closures

$$\mathsf{a}\ \text{type} \iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \sqsupseteq \mathbf{I}$$

Semantically typed λ-calculus (concurrent)

Types as closures

$$\mathsf{a} \text{ type} \iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Semantically typed λ-calculus (concurrent)

Types as closures

$$\mathsf{a}\ \text{type} \iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails,

Semantically typed λ-calculus (concurrent)

Types as closures

$$\mathsf{a} \text{ type} \iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\mathsf{rng}(\mathbf{Y}(\mathbf{I} \mid \langle \top \rangle)) = \{\bot, \top\}$.

Semantically typed λ-calculus (concurrent)

Types as closures

$$\mathsf{a}\ \text{type} \iff \mathsf{a} \circ \mathsf{a} = \mathsf{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\mathsf{rng}(\mathbf{Y}(\mathbf{I} \mid \langle \top \rangle)) = \{\bot, \top\}$.
We will define bool, nat, ...

Semantically typed λ-calculus (concurrent)

Types as closures

$$\mathsf{a}\ \text{type} \iff \mathsf{a}{\circ}\mathsf{a} = \mathsf{a} \ \sqsupseteq \ \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\mathsf{rng}(\mathbf{Y}(\mathbf{I} \mid \langle\top\rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

Semantically typed λ-calculus (concurrent)

Types as closures

$$\text{a type} \iff \text{a} \circ \text{a} = \text{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\text{rng}(\mathbf{Y}(\mathbf{I} \mid \langle\top\rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

How many denotations of 'not' ?

Semantically typed λ-calculus (concurrent)

Types as closures

$$\text{a type} \iff \text{a}\circ\text{a} = \text{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract?   Yes!

Range property fails, e.g. $\text{rng}(\mathbf{Y}(\mathbf{I} \mid \langle\top\rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

How many denotations of ‘not’ ?
Still infinitely many solutions,
but unique maximum denotation.

Semantically typed λ-calculus (concurrent)

Types as closures

$$\text{a type} \iff \text{a}\circ\text{a} = \text{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\text{rng}(\mathbf{Y}(\mathbf{I} \mid \langle\top\rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

How many denotations of 'not' ?
Still infinitely many solutions,
but unique maximum denotation.

Is there a solution to and x y = and y x ?

Semantically typed λ-calculus (concurrent)

Types as closures

$$\text{a type} \iff \text{a}\circ\text{a} = \text{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\text{rng}(\mathbf{Y}(\mathbf{I} \mid \langle\top\rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

How many denotations of 'not' ?
Still infinitely many solutions,
but unique maximum denotation.

Is there a solution to and x y = and y x ? Yes.

Semantically typed λ-calculus (concurrent)

Types as closures

$$\text{a type} \iff \text{a}\circ\text{a} = \text{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\text{rng}(\mathbf{Y}(\mathbf{I} \mid \langle\top\rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

How many denotations of 'not' ?
Still infinitely many solutions,
but unique maximum denotation.

Is there a solution to and x y = and y x ? Yes.

Semantically typed λ-calculus (concurrent)

Types as closures

$$\text{a type} \iff \text{a} \circ \text{a} = \text{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\text{rng}(\mathbf{Y}(\mathbf{I} \mid \langle \top \rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

How many denotations of 'not' ?
Still infinitely many solutions,
but unique maximum denotation.

Is there a solution to and x y = and y x ? Yes.

Semantically typed λ-calculus (concurrent)

Types as closures

$$\text{a type} \iff \text{a}\circ\text{a} = \text{a} \sqsupseteq \mathbf{I}$$

Is this really typed, fully abstract? Yes!

Range property fails, e.g. $\text{rng}(\mathbf{Y}(\mathbf{I} \mid \langle\top\rangle)) = \{\bot, \top\}$.
We will define bool, nat, ... from only $\mathbf{S}, \mathbf{K}, \mathbf{J}$.

How many denotations of 'not' ?
Still infinitely many solutions,
but unique maximum denotation.

Is there a solution to and x y = and y x ? Yes.

Main difference: coproducts (dropped, lifted)

What types are definable?

$$\begin{array}{rcll} x : a & \iff & a\ x = x & \textit{(inhabitation)} \\ \lambda x : a.M & = & (\lambda x.M) \circ a & \textit{(typed abstraction)} \end{array}$$

What types are definable?

$$
\begin{array}{rcll}
x : a & \Longleftrightarrow & a\ x = x & \textit{(inhabitation)} \\
\lambda x : a.M & = & (\lambda x.M) \circ a & \textit{(typed abstraction)} \\
 & & & \\
a \rightarrow b & = & \lambda f. b \circ f \circ a & \textit{(function = exponential)} \\
\forall y : a.M & = & \lambda x, y : a.M(x\ y) & \textit{(dependent, polymorphic)}
\end{array}
$$

What types are definable?

$$
\begin{array}{rcll}
x\!:\!a & \iff & a\ x = x & \textit{(inhabitation)} \\
\lambda x\!:\!a.M & = & (\lambda x.M)\circ a & \textit{(typed abstraction)} \\
\\
a \rightarrow b & = & \lambda f.b\circ f\circ a & \textit{(function = exponential)} \\
\forall y\!:\!a.M & = & \lambda x, y\!:\!a.M(x\ y) & \textit{(dependent, polymorphic)} \\
\\
a <: b & \iff & b\circ a\circ b = a & \textit{(subtyping)} \\
a \wedge b & = & \mathrm{Sub}\ a\ b & \textit{(type intersection)}
\end{array}
$$

What types are definable?

$$
\begin{array}{rcll}
x\,{:}\,a & \Longleftrightarrow & a\ x = x & \textit{(inhabitation)} \\
\lambda x\,{:}\,a.M & = & (\lambda x.M)\circ a & \textit{(typed abstraction)} \\
\\
a \rightarrow b & = & \lambda f.b\circ f\circ a & \textit{(function = exponential)} \\
\forall y\,{:}\,a.M & = & \lambda x, y\,{:}\,a.M(x\ y) & \textit{(dependent, polymorphic)} \\
\\
a <: b & \Longleftrightarrow & b\circ a\circ b = a & \textit{(subtyping)} \\
a \wedge b & = & \mathrm{Sub}\ a\ b & \textit{(type intersection)} \\
\\
a \times b & = & \mathrm{Prod}\ a\ b & \textit{(dropped products)} \\
a + b & = & \mathrm{Sum}\ a\ b & \textit{(dropped lifted sums)} \\
\exists y\,{:}\,a.M & = & \mathrm{Exists}\ \lambda y\,{:}\,a.M & \textit{(d'd l'd indexed sums)}
\end{array}
$$

What types are definable?

$$
\begin{array}{rcll}
x : a & \Longleftrightarrow & a\ x = x & \textit{(inhabitation)} \\
\lambda x : a.M & = & (\lambda x.M) \circ a & \textit{(typed abstraction)} \\
\\
a \to b & = & \lambda f.b \circ f \circ a & \textit{(function = exponential)} \\
\forall y : a.M & = & \lambda x, y : a.M(x\ y) & \textit{(dependent, polymorphic)} \\
\\
a <: b & \Longleftrightarrow & b \circ a \circ b = a & \textit{(subtyping)} \\
a \wedge b & = & \mathrm{Sub}\ a\ b & \textit{(type intersection)} \\
\\
a \times b & = & \mathrm{Prod}\ a\ b & \textit{(dropped products)} \\
a + b & = & \mathrm{Sum}\ a\ b & \textit{(dropped lifted sums)} \\
\exists y : a.M & = & \mathrm{Exists}\ \lambda y : a.M & \textit{(d'd l'd indexed sums)}
\end{array}
$$

Also atoms: type,

What types are definable?

$$
\begin{array}{rcll}
x : a & \iff & a\ x = x & \textit{(inhabitation)} \\
\lambda x : a.M & = & (\lambda x.M) \circ a & \textit{(typed abstraction)} \\
\\
a \to b & = & \lambda f.b \circ f \circ a & \textit{(function = exponential)} \\
\forall y : a.M & = & \lambda x, y : a.M(x\ y) & \textit{(dependent, polymorphic)} \\
\\
a <: b & \iff & b \circ a \circ b = a & \textit{(subtyping)} \\
a \wedge b & = & \mathsf{Sub}\ a\ b & \textit{(type intersection)} \\
\\
a \times b & = & \mathsf{Prod}\ a\ b & \textit{(dropped products)} \\
a + b & = & \mathsf{Sum}\ a\ b & \textit{(dropped lifted sums)} \\
\exists y : a.M & = & \mathsf{Exists}\ \lambda y : a.M & \textit{(d'd l'd indexed sums)}
\end{array}
$$

Also atoms: type, any,

What types are definable?

$$
\begin{array}{rcll}
x\!:\!a & \iff & a\ x = x & \textit{(inhabitation)} \\
\lambda x\!:\!a.M & = & (\lambda x.M)\circ a & \textit{(typed abstraction)} \\
\\
a \to b & = & \lambda f.b\circ f\circ a & \textit{(function = exponential)} \\
\forall y\!:\!a.M & = & \lambda x, y\!:\!a.M(x\ y) & \textit{(dependent, polymorphic)} \\
\\
a <: b & \iff & b\circ a\circ b = a & \textit{(subtyping)} \\
a \wedge b & = & \mathsf{Sub}\ a\ b & \textit{(type intersection)} \\
\\
a \times b & = & \mathsf{Prod}\ a\ b & \textit{(dropped products)} \\
a + b & = & \mathsf{Sum}\ a\ b & \textit{(dropped lifted sums)} \\
\exists y\!:\!a.M & = & \mathsf{Exists}\ \lambda y\!:\!a.M & \textit{(d'd l'd indexed sums)}
\end{array}
$$

Also atoms: type, any, nil, unit, bool,

What types are definable?

$$
\begin{array}{rcll}
x\!:\!a & \iff & a\ x = x & \textit{(inhabitation)} \\
\lambda x\!:\!a.M & = & (\lambda x.M)\circ a & \textit{(typed abstraction)} \\
\\
a \to b & = & \lambda f.b\circ f\circ a & \textit{(function = exponential)} \\
\forall y\!:\!a.M & = & \lambda x, y\!:\!a.M(x\ y) & \textit{(dependent, polymorphic)} \\
\\
a <: b & \iff & b\circ a\circ b = a & \textit{(subtyping)} \\
a \wedge b & = & \mathrm{Sub}\ a\ b & \textit{(type intersection)} \\
\\
a\times b & = & \mathrm{Prod}\ a\ b & \textit{(dropped products)} \\
a + b & = & \mathrm{Sum}\ a\ b & \textit{(dropped lifted sums)} \\
\exists y\!:\!a.M & = & \mathrm{Exists}\ \lambda y\!:\!a.M & \textit{(d'd l'd indexed sums)}
\end{array}
$$

Also atoms: type, any, nil, unit, bool, nat

The universal type of types

$$\mathsf{type} \;:=\; \lambda \mathsf{a}.\ \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}{\circ}\mathsf{a} \mid \mathsf{a}{\circ}\mathsf{a}{\circ}\mathsf{a} \mid \ldots$$

The universal type of types

$$\text{type} := \lambda\text{a}.\ \mathbf{I} \mid \text{a} \mid \text{a}\circ\text{a} \mid \text{a}\circ\text{a}\circ\text{a} \mid \ldots \quad = \lambda\text{a}.\mathbf{Y}\lambda\text{b}.\ \mathbf{I} \mid \text{a}\circ\text{b}$$

The universal type of types

$$\mathsf{type} \;:=\; \lambda \mathsf{a}.\ \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}{\circ}\mathsf{a} \mid \mathsf{a}{\circ}\mathsf{a}{\circ}\mathsf{a} \mid \ldots \qquad = \; \lambda \mathsf{a}.\mathbf{Y}\lambda \mathsf{b}.\ \mathbf{I} \mid \mathsf{a}{\circ}\mathsf{b}$$

Theorem

type *is a closure,*

The universal type of types

$$\mathsf{type} := \lambda \mathsf{a}.\ \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \mathsf{a}\circ\mathsf{a}\circ\mathsf{a} \mid \dots \quad = \lambda \mathsf{a}.\mathbf{Y}\lambda \mathsf{b}.\ \mathbf{I} \mid \mathsf{a}\circ\mathsf{b}$$

Theorem

type *is a closure, and* a : type $\iff$ a *is a closure.*

The universal type of types

$$\mathsf{type} \;:=\; \lambda \mathsf{a}.\, \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}{\circ}\mathsf{a} \mid \mathsf{a}{\circ}\mathsf{a}{\circ}\mathsf{a} \mid \ldots \qquad = \; \lambda \mathsf{a}.\mathbf{Y}\lambda \mathsf{b}.\, \mathbf{I} \mid \mathsf{a}{\circ}\mathsf{b}$$

Theorem

type *is a closure, and* a : type $\iff$ a *is a closure.*

Proof.

(closure) type $\sqsupseteq \lambda \mathsf{a}.\mathsf{a} = \mathbf{I}$,

The universal type of types

$$\mathsf{type} \;:=\; \lambda \mathsf{a}.\ \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \mathsf{a}\circ\mathsf{a}\circ\mathsf{a} \mid \ldots \qquad = \; \lambda \mathsf{a}.\mathbf{Y}\lambda \mathsf{b}.\ \mathbf{I} \mid \mathsf{a}\circ\mathsf{b}$$

Theorem

type *is a closure, and* a : type $\iff$ a *is a closure.*

Proof.

(closure) type $\sqsupseteq \lambda \mathsf{a}.\mathsf{a} = \mathbf{I}$, and type(type a) = type a.

The universal type of types

$$\mathsf{type} \;:=\; \lambda \mathsf{a}.\ \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \mathsf{a}\circ\mathsf{a}\circ\mathsf{a} \mid \ldots \qquad = \; \lambda \mathsf{a}.\mathbf{Y}\lambda \mathsf{b}.\ \mathbf{I} \mid \mathsf{a}\circ\mathsf{b}$$

Theorem

type *is a closure, and* a : type $\iff$ a *is a closure.*

Proof.

(closure) type $\sqsupseteq \lambda$a.a $= \mathbf{I}$, and type(type a) $=$ type a.

$(\implies)$ Suppose a : type, i.e., a $= \mathbf{I} \mid$ a $\mid$ a$\circ$a $\mid \ldots$.

The universal type of types

$$\text{type} := \lambda a.\ \mathbf{I} \mid a \mid a\circ a \mid a\circ a\circ a \mid \ldots \quad = \lambda a.\mathbf{Y}\lambda b.\ \mathbf{I} \mid a\circ b$$

Theorem

type *is a closure, and* a : type $\iff$ a *is a closure.*

Proof.

(closure) type $\sqsupseteq \lambda a.a = \mathbf{I}$, and type(type a) = type a.

($\Longrightarrow$) Suppose a : type, i.e., $a = \mathbf{I} \mid a \mid a\circ a \mid \ldots$.

Then $a \sqsupseteq \mathbf{I}$,

The universal type of types

$$\mathsf{type} \;:=\; \lambda \mathsf{a}.\, \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \mathsf{a}\circ\mathsf{a}\circ\mathsf{a} \mid \ldots \qquad = \; \lambda \mathsf{a}.\mathbf{Y}\lambda \mathsf{b}.\, \mathbf{I} \mid \mathsf{a}\circ\mathsf{b}$$

Theorem

type *is a closure, and* $\mathsf{a}:\mathsf{type} \iff \mathsf{a}$ *is a closure.*

Proof.

(closure) $\mathsf{type} \sqsupseteq \lambda \mathsf{a}.\mathsf{a} = \mathbf{I}$, and $\mathsf{type}(\mathsf{type}\ \mathsf{a}) = \mathsf{type}\ \mathsf{a}$.

$(\implies)$ Suppose $\mathsf{a}:\mathsf{type}$, i.e., $\mathsf{a} = \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \ldots$.
Then $\mathsf{a} \sqsupseteq \mathbf{I}$, and $\mathsf{a}\circ\mathsf{a} = \mathsf{a}$.

The universal type of types

$$\mathsf{type} \;:=\; \lambda \mathsf{a}.\ \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \mathsf{a}\circ\mathsf{a}\circ\mathsf{a} \mid \ldots \qquad = \; \lambda \mathsf{a}.\mathbf{Y}\lambda \mathsf{b}.\ \mathbf{I} \mid \mathsf{a}\circ\mathsf{b}$$

Theorem

type *is a closure, and* a : type $\iff$ a *is a closure.*

Proof.

(closure) type $\sqsupseteq \lambda \mathsf{a}.\mathsf{a} = \mathbf{I}$, and type(type a) = type a.

($\Longrightarrow$) Suppose a : type, i.e., $\mathsf{a} = \mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \ldots$.
Then $\mathsf{a} \sqsupseteq \mathbf{I}$, and $\mathsf{a}\circ\mathsf{a} = \mathsf{a}$.

($\Longleftarrow$) If $\mathsf{a} \sqsupseteq \mathbf{I}$ then $(\mathbf{I} \mid \mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \ldots) = (\mathsf{a} \mid \mathsf{a}\circ\mathsf{a} \mid \ldots)$.

The universal type of types

$$\text{type} \;:=\; \lambda \mathsf{a}.\ \mathbf{I} \mid \mathsf{a} \mid \mathsf{a} \circ \mathsf{a} \mid \mathsf{a} \circ \mathsf{a} \circ \mathsf{a} \mid \ldots \qquad = \; \lambda \mathsf{a}. \mathbf{Y} \lambda \mathsf{b}.\ \mathbf{I} \mid \mathsf{a} \circ \mathsf{b}$$

Theorem

type *is a closure, and* a : type $\iff$ a *is a closure.*

Proof.

(closure) type $\sqsupseteq \lambda$a.a $= \mathbf{I}$, and type(type a) $=$ type a.

($\Longrightarrow$) Suppose a : type, i.e., a $= \mathbf{I} \mid$ a $\mid$ a$\circ$a $\mid \ldots$.
Then a $\sqsupseteq \mathbf{I}$, and a$\circ$a $=$ a.

($\Longleftarrow$) If a $\sqsupseteq \mathbf{I}$ then $(\mathbf{I} \mid$ a $\mid$ a$\circ$a $\mid \ldots) =$ (a $\mid$ a$\circ$a $\mid \ldots$).
If also a $=$ a$\circ$a, the chain collapses to a.

$\square$

Maximal and minimal types

Everything is fixed by the identity,

Maximal and minimal types

Everything is fixed by the identity, so the largest type is

$$\text{any} := \mathbf{I} \quad = \text{type } \mathbf{I}.$$

Maximal and minimal types

Everything is fixed by the identity, so the largest type is

$$\begin{aligned} \mathsf{any} &:= \mathbf{I} \quad = \mathsf{type}\ \mathbf{I}. \\ \mathsf{inhab}(\mathsf{any}) &= \mathcal{B} \end{aligned}$$

Maximal and minimal types

Everything is fixed by the identity, so the largest type is

$$\begin{aligned} \text{any} \;&:=\; \mathbf{I} \quad = \text{type } \mathbf{I}. \\ \text{inhab}(\text{any}) &= \mathcal{B} \end{aligned}$$

Every type is inhabited by $\top$,

Maximal and minimal types

Everything is fixed by the identity, so the largest type is

$$\text{any} := \mathbf{I} = \text{type } \mathbf{I}.$$
$$\text{inhab(any)} = \mathcal{B}$$

Every type is inhabited by $\top$, so the smallest type is

$$\text{nil} := \top = \text{type } \top.$$

Maximal and minimal types

Everything is fixed by the identity, so the largest type is

$$\begin{aligned} \mathsf{any} \;&:=\; \mathbf{I} \quad = \mathsf{type}\ \mathbf{I}. \\ \mathsf{inhab}(\mathsf{any}) &= \mathcal{B} \end{aligned}$$

Every type is inhabited by $\top$, so the smallest type is

$$\begin{aligned} \mathsf{nil} \;&:=\; \top \quad = \mathsf{type}\ \top. \\ \mathsf{inhab}(\mathsf{nil}) &= \{\top\} \end{aligned}$$

Maximal and minimal types

Everything is fixed by the identity, so the largest type is

$$\begin{aligned} \text{any} &:= \mathbf{I} \quad = \text{type } \mathbf{I}. \\ \text{inhab(any)} &= \mathcal{B} \end{aligned}$$

Every type is inhabited by $\top$, so the smallest type is

$$\begin{aligned} \text{nil} &:= \top \quad = \text{type } \top. \\ \text{inhab(nil)} &= \{\top\} \end{aligned}$$

nil is: terminal object,

Maximal and minimal types

Everything is fixed by the identity, so the largest type is

$$\begin{aligned} \text{any} &:= \mathbf{I} = \text{type } \mathbf{I}. \\ \text{inhab}(\text{any}) &= \mathcal{B} \end{aligned}$$

Every type is inhabited by $\top$, so the smallest type is

$$\begin{aligned} \text{nil} &:= \top = \text{type } \top. \\ \text{inhab}(\text{nil}) &= \{\top\} \end{aligned}$$

nil is: terminal object, dropped initial object.
($\mathcal{B}$ has no initial object)

Function types (exponentials)

Definition

For any terms a, b, define the conjugation operator

$$a \to b \;:=\; \lambda f.\, b \circ f \circ a \qquad = \lambda f, x.\; b(f(a\; x))$$

Function types (exponentials)

Definition

For any terms a, b, define the conjugation operator

$$a \to b \ := \ \lambda f. b \circ f \circ a \qquad = \lambda f, x.\ b(f(a\ x))$$

(associates to the right)

Function types (exponentials)

Definition

For any terms a, b, define the conjugation operator

$$a \to b \;:=\; \lambda f. b \circ f \circ a \qquad = \lambda f, x.\ b(f(a\ x))$$

(associates to the right)

Now define a binary operation on types

$$\mathsf{Exp} \;:=\; \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\lambda a, b.\ a \to b).$$

Function types (exponentials)

Definition

For any terms a, b, define the conjugation operator

$$a \to b \;:=\; \lambda f. b \circ f \circ a \qquad = \lambda f, x.\; b(f(a\ x))$$

(associates to the right)

Now define a binary operation on types

$$\begin{aligned} \mathrm{Exp} \;:=&\; \mathrm{type} \to \mathrm{type} \to \mathrm{type}\ (\lambda a, b.\ a \to b). \\ =&\; \lambda a : \mathrm{type}.\ \mathrm{type} \to \mathrm{type}\ \lambda b.\ a \to b \end{aligned}$$

Function types (exponentials)

Definition

For any terms a, b, define the conjugation operator

$$a \to b \;:=\; \lambda f. b \circ f \circ a \qquad = \lambda f, x.\; b(f(a\;x))$$

(associates to the right)

Now define a binary operation on types

$$\begin{aligned} \mathrm{Exp} \;&:=\; \mathrm{type} \to \mathrm{type} \to \mathrm{type}\; (\lambda a, b.\; a \to b). \\ &=\; \lambda a : \mathrm{type}.\; \mathrm{type} \to \mathrm{type}\; \lambda b.\; a \to b \\ &=\; \lambda a : \mathrm{type},\;\; b : \mathrm{type}.\; \mathrm{type}\; (a \to b) \end{aligned}$$

Function types (exponentials)

Definition

For any terms a, b, define the conjugation operator

$$a \to b \ := \ \lambda f. b \circ f \circ a \qquad = \lambda f, x.\ b(f(a\ x))$$

(associates to the right)

Now define a binary operation on types

$$\begin{aligned} \mathsf{Exp} \ &:= \ \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\lambda a, b.\ a \to b). \\ &= \ \lambda a : \mathsf{type}.\ \mathsf{type} \to \mathsf{type}\ \lambda b.\ a \to b \\ &= \ \lambda a : \mathsf{type},\ b : \mathsf{type}.\ \mathsf{type}\ (a \to b) \end{aligned}$$

We'll use this form often:

$$\mathsf{some\_term} \ := \ \mathsf{its\_type} \ \ \mathsf{untyped\_definition}.$$

Navigating a minefield.

Consider the section-retract pair

Navigating a minefield.

Consider the section-retract pair

$$R_{mn} \;:=\; \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x$$

Navigating a minefield.

Consider the section-retract pair

$$\begin{aligned} R_{mn} &:= \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x \\ L_{mn} &:= \lambda g, x.\ g\ \ \top^{\sim m}\ \ x\ \ \top^{\sim n} \end{aligned}$$

Navigating a minefield.

Consider the section-retract pair

$$\begin{aligned} R_{mn} &:= \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x \\ L_{mn} &:= \lambda g, x.\ g\ \ \top^{\sim m}\ \ x\ \ \top^{\sim n} \end{aligned}$$

so that

$$L_{mn} \circ R_{mn} = \mathbf{I}$$

Navigating a minefield.

Consider the section-retract pair

$$\begin{array}{lll} \mathsf{R}_{mn} & := & \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x \\ \mathsf{L}_{mn} & := & \lambda g, x.\ g\ \ \top^{\sim m}\ \ x\ \ \top^{\sim n} \end{array}$$

so that

$$\begin{array}{l} \mathsf{L}_{mn} \circ \mathsf{R}_{mn} = \mathbf{I} \\ \mathsf{R}_{mn} \circ \mathsf{L}_{mn} = \lambda f, \underline{w}, x, \underline{y}.\ f\ \top^{\sim m}\ x\ \top^{\sim n} \end{array}$$

Navigating a minefield.

Consider the section-retract pair

$$\begin{aligned} \mathsf{R}_{mn} &:= \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x \\ \mathsf{L}_{mn} &:= \lambda g, x.\ g\ \top^{\sim m}\ x\ \top^{\sim n} \end{aligned}$$

so that

$$\begin{aligned} &\mathsf{L}_{mn} \circ \mathsf{R}_{mn} = \mathbf{I} \\ &\mathsf{R}_{mn} \circ \mathsf{L}_{mn} = \lambda f, \underline{w}, x, \underline{y}.\ f\ \top^{\sim m}\ x\ \top^{\sim n} \sqsupseteq \mathbf{I} \end{aligned}$$

Navigating a minefield.

Consider the section-retract pair

$$\begin{aligned} R_{mn} &:= \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x \\ L_{mn} &:= \lambda g, x.\ g\ \ \top^{\sim m}\ \ x\ \ \top^{\sim n} \end{aligned}$$

so that

$$\begin{aligned} L_{mn} \circ R_{mn} &= \mathbf{I} \\ R_{mn} \circ L_{mn} &= \lambda f, \underline{w}, x, \underline{y}.\ f\ \top^{\sim m}\ x\ \top^{\sim n}\ \ \sqsupseteq\ \ \mathbf{I} \end{aligned}$$

Hence $R_{mn} \circ L_{mn}$ is a closure.

Navigating a minefield.

Consider the section-retract pair

$$\begin{aligned} \mathsf{R}_{mn} &:= \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x \\ \mathsf{L}_{mn} &:= \lambda g, x.\ g\ \ \top^{\sim m}\ \ x\ \ \top^{\sim n} \end{aligned}$$

so that

$$\begin{aligned} &\mathsf{L}_{mn} \circ \mathsf{R}_{mn} = \mathbf{I} \\ &\mathsf{R}_{mn} \circ \mathsf{L}_{mn} = \lambda f, \underline{w}, x, \underline{y}.\ f\ \top^{\sim m}\ x\ \top^{\sim n}\ \sqsupseteq\ \mathbf{I} \end{aligned}$$

Hence $\mathsf{R}_{mn} \circ \mathsf{L}_{mn}$ is a closure.

(often omit indices: $\mathsf{L} \circ \mathsf{R} = \mathbf{I}$)

Navigating a minefield.

Consider the section-retract pair

$$\begin{aligned} R_{mn} &:= \lambda f, w_1, \ldots, w_m, x, y_1, \ldots, y_n.\ f\ x \\ L_{mn} &:= \lambda g, x.\ g\ \ \top^{\sim m}\ \ x\ \ \top^{\sim n} \end{aligned}$$

so that

$$\begin{aligned} L_{mn} \circ R_{mn} &= \mathbf{I} \\ R_{mn} \circ L_{mn} &= \lambda f, \underline{w}, x, \underline{y}.\ f\ \top^{\sim m}\ x\ \top^{\sim n}\ \ \sqsupseteq\ \ \mathbf{I} \end{aligned}$$

Hence $R_{mn} \circ L_{mn}$ is a closure.

(often omit indices: $L \circ R = \mathbf{I}$)

Think of L as a minefield of errors
and R as a map through the minefield.

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$$(a \rightarrow a) \rightarrow a \rightarrow a$$

Avoiding errors.

The Church numeral $1 = \lambda \mathsf{f}, \mathsf{x}.\mathsf{f}\ \mathsf{x}$ has simple type

$(\mathsf{a} \rightarrow \mathsf{a}) \rightarrow \mathsf{a} \rightarrow \mathsf{a}$ *note variance of each* a

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$(a \to a) \to a \to a$ *note variance of each* a

Consider the action of $(L \to R) \to R \to L$ on 1:

Avoiding errors.

The Church numeral $1 = \lambda \mathsf{f}, \mathsf{x}.\mathsf{f}\ \mathsf{x}$ has simple type

$(\mathsf{a} \to \mathsf{a}) \to \mathsf{a} \to \mathsf{a}$ *note variance of each* a

Consider the action of $(\mathsf{L} \to \mathsf{R}) \to \mathsf{R} \to \mathsf{L}$ on 1:

$(\mathsf{L} \to \mathsf{R}) \to \mathsf{R} \to \mathsf{L}\ (\lambda \mathsf{f}, \mathsf{x}.\mathsf{f}\ \mathsf{x})\ \mathsf{f}\ \mathsf{x}$

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$$(a \to a) \to a \to a \qquad \textit{note variance of each } a$$

Consider the action of $(L \to R) \to R \to L$ on 1:

$$\begin{aligned} &(L \to R) \to R \to L\ (\lambda f, x.f\ x)\ f\ x \\ &\quad = \ L\ (\ (\lambda f, x.f\ x)\ (R \circ f \circ L)\ (R\ x)\ ) \end{aligned}$$

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$$(a \to a) \to a \to a \qquad \textit{note variance of each } a$$

Consider the action of $(L \to R) \to R \to L$ on 1:

$$\begin{aligned} &(L \to R) \to R \to L\ (\lambda f, x.f\ x)\ f\ x \\ &\quad = L\ (\ (\lambda f, x.f\ x)\ (R \circ f \circ L)\ (R\ x)\ ) \\ &\quad = L\ (R \circ f \circ L\ (R\ x)) \end{aligned}$$

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$$(a \to a) \to a \to a \qquad \textit{note variance of each } a$$

Consider the action of $(L \to R) \to R \to L$ on 1:

$$\begin{aligned}
&(L \to R) \to R \to L\ (\lambda f, x.f\ x)\ f\ x \\
&\quad = \ L\ (\ (\lambda f, x.f\ x)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&\quad = \ L\ (R \circ f \circ L\ (R\ x)) \\
&\quad = \ (L \circ R) \circ f \circ (L \circ R)\ x
\end{aligned}$$

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$$(a \to a) \to a \to a \qquad \textit{note variance of each } a$$

Consider the action of $(L \to R) \to R \to L$ on 1:

$$\begin{aligned} &(L \to R) \to R \to L\ (\lambda f, x.f\ x)\ f\ x \\ &\quad = L\ (\ (\lambda f, x.f\ x)\ (R \circ f \circ L)\ (R\ x)\ ) \\ &\quad = L\ (R \circ f \circ L\ (R\ x)) \\ &\quad = (L \circ R) \circ f \circ (L \circ R)\ x \\ &\quad = \mathbf{I} \circ f \circ \mathbf{I}\ x \end{aligned}$$

Avoiding errors.

The Church numeral $1 = \lambda \mathsf{f}, \mathsf{x}.\mathsf{f}\ \mathsf{x}$ has simple type

$(a \to a) \to a \to a$ *note variance of each* a

Consider the action of $(L \to R) \to R \to L$ on 1:

$$
\begin{aligned}
&(L \to R) \to R \to L\ (\lambda f, x.f\ x)\ f\ x \\
&\quad = L\ (\ (\lambda f, x.f\ x)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&\quad = L\ (R \circ f \circ L\ (R\ x)) \\
&\quad = (L \circ R) \circ f \circ (L \circ R)\ x \\
&\quad = \mathbf{I} \circ f \circ \mathbf{I}\ x \\
&\quad = f\ x
\end{aligned}
$$

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$$(a \to a) \to a \to a \qquad \textit{note variance of each } a$$

Consider the action of $(L \to R) \to R \to L$ on 1:

$$\begin{aligned}
&(L \to R) \to R \to L\ (\lambda f, x.f\ x)\ f\ x \\
&\quad = \ L\ (\ (\lambda f, x.f\ x)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&\quad = \ L\ (R \circ f \circ L\ (R\ x)) \\
&\quad = \ (L \circ R) \circ f \circ (L \circ R)\ x \\
&\quad = \ \mathbf{I} \circ f \circ \mathbf{I}\ x \\
&\quad = \ f\ x \quad = 1\ f\ x
\end{aligned}$$

Hence $1 : (L \to R) \to R \to L$.

Avoiding errors.

The Church numeral $1 = \lambda f, x.f\ x$ has simple type

$$(a \rightarrow a) \rightarrow a \rightarrow a \qquad \textit{note variance of each a}$$

Consider the action of $(L \rightarrow R) \rightarrow R \rightarrow L$ on 1:

$$\begin{aligned}
&(L \rightarrow R) \rightarrow R \rightarrow L\ (\lambda f, x.f\ x)\ f\ x \\
&\quad = \ L\ (\ (\lambda f, x.f\ x)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&\quad = \ L\ (R \circ f \circ L\ (R\ x)) \\
&\quad = \ (L \circ R) \circ f \circ (L \circ R)\ x \\
&\quad = \ \mathbf{I} \circ f \circ \mathbf{I}\ x \\
&\quad = \ f\ x \quad = 1\ f\ x
\end{aligned}$$

Hence $1 : (L \rightarrow R) \rightarrow R \rightarrow L$.
actually works for any section-retract pair

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda \mathsf{f}, \mathsf{x}.\mathsf{x}\ \mathsf{f}$?

$$(\mathsf{L} \rightarrow \mathsf{R}) \rightarrow \mathsf{R} \rightarrow \mathsf{L}\ (\lambda \mathsf{f}, \mathsf{x}.\mathsf{x}\ \mathsf{f})\ \mathsf{f}\ \mathsf{x}$$

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$\begin{aligned} &(L \rightarrow R) \rightarrow R \rightarrow L\ (\lambda f, x.x\ f)\ f\ x \\ &\quad = L\ (\ (\lambda f, x.x\ f)\ (R \circ f \circ L)\ (R\ x)\ ) \end{aligned}$$

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$
\begin{aligned}
&(L \rightarrow R) \rightarrow R \rightarrow L\ (\lambda f, x.x\ f)\ f\ x \\
&\quad = \ L\ (\ (\lambda f, x.x\ f)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&\quad = \ L\ (R\ x\ (R \circ f \circ L))
\end{aligned}
$$

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$
\begin{aligned}
(L \to R) \to R \to L\ (\lambda f, x.x\ f)\ f\ x \\
&= L\ (\ (\lambda f, x.x\ f)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&= L\ (R\ x\ (R \circ f \circ L)) \\
&= L\ x
\end{aligned}
$$

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$
\begin{aligned}
(L \to R) \to R \to L\ (\lambda f, x.x\ f)\ f\ x \\
&= L\ (\ (\lambda f, x.x\ f)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&= L\ (R\ x\ (R \circ f \circ L)) \\
&= L\ x \\
&= x\ \top
\end{aligned}
$$

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$
\begin{aligned}
(\mathsf{L}\!\rightarrow\!\mathsf{R})\!\rightarrow\!\mathsf{R}\!\rightarrow\!\mathsf{L}\ (\lambda f, x.x\ f)\ f\ x \\
&= \mathsf{L}\ (\ (\lambda f, x.x\ f)\ (\mathsf{R}\circ f\circ \mathsf{L})\ (\mathsf{R}\ x)\ ) \\
&= \mathsf{L}\ (\mathsf{R}\ x\ (\mathsf{R}\circ f\circ \mathsf{L})) \\
&= \mathsf{L}\ x \\
&= x\ \top \qquad \neq\ x\ f
\end{aligned}
$$

oops: $\lambda f, x.x\ f\ \div\ (\mathsf{L}\!\rightarrow\!\mathsf{R})\!\rightarrow\!\mathsf{R}\!\rightarrow\!\mathsf{L}.$

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$
\begin{aligned}
(L \to R) \to R \to L\ (\lambda f, x.x\ f)\ f\ x \\
&= L\ (\ (\lambda f, x.x\ f)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&= L\ (R\ x\ (R \circ f \circ L)) \\
&= L\ x \\
&= x\ \top \qquad \neq\ x\ f
\end{aligned}
$$

oops: $\lambda f, x.x\ f\ \div\ (L \to R) \to R \to L$.

Can we force any incorrect term up to $\top =$ error?

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$
\begin{aligned}
(L \to R) \to R \to L\ (\lambda f, x.x\ f)\ f\ x \\
&= L\ (\ (\lambda f, x.x\ f)\ (R \circ f \circ L)\ (R\ x)\ ) \\
&= L\ (R\ x\ (R \circ f \circ L)) \\
&= L\ x \\
&= x\ \top \qquad \neq\ x\ f
\end{aligned}
$$

oops: $\lambda f, x.x\ f\ \div\ (L \to R) \to R \to L$.

Can we force any incorrect term up to $\top =$ error?
Can we raise any partial term up to a fixedpoint?

Failing to avoid errors.

What about non-Church numerals, e.g., $\lambda f, x.x\ f$?

$$\begin{aligned}(L \to R) \to R \to L\ (\lambda f, x.x\ f)\ f\ x \\ &= L\ (\ (\lambda f, x.x\ f)\ (R \circ f \circ L)\ (R\ x)\ ) \\ &= L\ (R\ x\ (R \circ f \circ L)) \\ &= L\ x \\ &= x\ \top \qquad \neq\ x\ f\end{aligned}$$

oops: $\lambda f, x.x\ f\ \div\ (L \to R) \to R \to L$.

Can we force any incorrect term up to $\top$ = error?
Can we raise any partial term up to a fixedpoint?
...sometimes...

The type of divergent computations

$$\mathsf{div} \;:= \bigsqcup \mathsf{m} \geq 0.\; \mathsf{L}_{\mathsf{m}0} \;= \bigsqcup \mathsf{m} \geq 0.\; \mathsf{m}\; \langle \top \rangle$$

The type of divergent computations

$$\mathsf{div} \;:= \bigsqcup m \geq 0.\; \mathsf{L}_{m0} \;= \bigsqcup m \geq 0.\; m\,\langle\top\rangle \;= \mathsf{type}\,\langle\top\rangle$$

The type of divergent computations

$$\mathsf{div} \;:= \bigsqcup \mathsf{m} \geq 0.\ \mathsf{L_{m0}} \;= \bigsqcup \mathsf{m} \geq 0.\ \mathsf{m}\ \langle \top \rangle \quad = \mathsf{type}\ \langle \top \rangle$$

Theorem

$\mathsf{inhab}(\mathsf{div}) = \{\bot, \top\}$.

The type of divergent computations

$$\text{div} \;:= \bigsqcup m \geq 0.\; L_{m0} \;= \bigsqcup m \geq 0.\; m\ \langle \top \rangle \;= \text{type}\ \langle \top \rangle$$

Theorem

$\text{inhab}(\text{div}) = \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot$, $\quad \bot : \text{div}$.

The type of divergent computations

$$\text{div} := \bigsqcup \text{m} \geq 0.\ \text{L}_{\text{m}0} = \bigsqcup \text{m} \geq 0.\ \text{m}\ \langle\top\rangle = \text{type}\ \langle\top\rangle$$

Theorem

$\text{inhab}(\text{div}) = \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot$, $\quad \bot : \text{div}$.

Any other term $\text{q} : \text{div}$ in question must converge

The type of divergent computations

$$\mathsf{div} \;:= \bigsqcup \mathsf{m} \geq 0.\ \mathsf{L}_{\mathsf{m0}} \;= \bigsqcup \mathsf{m} \geq 0.\ \mathsf{m}\ \langle\top\rangle \;= \mathsf{type}\ \langle\top\rangle$$

Theorem

inhab(div) $= \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot$, $\quad \bot$: div.

Any other term q : div in question must converge

(recall q converges iff for some m, q $\top^{\sim m} \;\equiv\; \top$).

The type of divergent computations

$$\mathsf{div} \;:= \bigsqcup \mathsf{m} \geq 0.\ \mathsf{L_{m0}} \;= \bigsqcup \mathsf{m} \geq 0.\ \mathsf{m}\ \langle \top \rangle \;\; = \mathsf{type}\ \langle \top \rangle$$

Theorem

$\mathsf{inhab(div)} = \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot$, $\quad \bot : \mathsf{div}$.

Any other term $\mathsf{q} : \mathsf{div}$ in question must converge

(recall q converges iff for some m, $\mathsf{q}\ \top^{\sim \mathsf{m}} \equiv \top$).

Then

$$\mathsf{q} = \mathsf{div}\ \mathsf{q}$$

The type of divergent computations

$$\text{div} \;:= \bigsqcup m \geq 0.\ L_{m0} \;= \bigsqcup m \geq 0.\ m\ \langle \top \rangle \;\; = \text{type}\ \langle \top \rangle$$

Theorem

$\text{inhab}(\text{div}) = \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot, \quad \bot : \text{div}$.

Any other term $q : \text{div}$ in question must converge
(recall q converges iff for some m, $q\ \top^{\sim m}\ \equiv\ \top$).

Then

$$\begin{aligned} q &= \text{div}\ q \\ &= \bigsqcup m \geq 0.\ m\ \langle \top \rangle\ q \end{aligned}$$

The type of divergent computations

$$\text{div} \;:= \bigsqcup m \geq 0.\ L_{m0} \;= \bigsqcup m \geq 0.\ m\ \langle\top\rangle \;= \text{type}\ \langle\top\rangle$$

Theorem

$\text{inhab(div)} = \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot$, $\quad \bot : \text{div}$.

Any other term $q : \text{div}$ in question must converge

(recall q converges iff for some m, $q\ \top^{\sim m} \equiv \top$).

Then

$$\begin{aligned} q &= \text{div}\ q \\ &= \bigsqcup m \geq 0.\ m\ \langle\top\rangle\ q \\ &= \bigsqcup m \geq 0.\ q\ \top^{\sim m} \end{aligned}$$

The type of divergent computations

$$\text{div} := \bigsqcup \text{m} \geq 0.\ \text{L}_{\text{m}0} = \bigsqcup \text{m} \geq 0.\ \text{m}\ \langle \top \rangle = \text{type}\ \langle \top \rangle$$

Theorem

$\text{inhab}(\text{div}) = \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot$, $\bot : \text{div}$.

Any other term $\text{q} : \text{div}$ in question must converge (recall q converges iff for some m, $\text{q}\ \top^{\sim\text{m}} \equiv \top$).

Then

$$\begin{aligned} \text{q} &= \text{div}\ \text{q} \\ &= \bigsqcup \text{m} \geq 0.\ \text{m}\ \langle \top \rangle\ \text{q} \\ &= \bigsqcup \text{m} \geq 0.\ \text{q}\ \top^{\sim\text{m}} = \top \end{aligned}$$

□

The type of divergent computations

$$\text{div} := \bigsqcup m \geq 0.\ L_{m0} = \bigsqcup m \geq 0.\ m\ \langle \top \rangle = \text{type}\ \langle \top \rangle$$

Theorem

inhab(div) $= \{\bot, \top\}$.

Proof.

Since $\bot\ \top = \bot$, $\bot$: div.
Any other term q : div in question must converge
(recall q converges iff for some m, $q\ \top^{\sim m} \equiv \top$).
Then

$$\begin{aligned} q &= \text{div}\ q \\ &= \bigsqcup m \geq 0.\ m\ \langle \top \rangle\ q \\ &= \bigsqcup m \geq 0.\ q\ \top^{\sim m} \quad = \top \end{aligned}$$

□

Moral: every candidate q stepped on a mine somewhere.

Protecting terms from divergence

We'll also need to make terms temporarily inert

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\mathsf{curry} \;:=\; \lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}\langle \mathsf{x}, \mathsf{y}\rangle$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\text{curry} \;:=\; \lambda f, x, y.\ f\langle x, y\rangle \qquad = \; \lambda f, x, y.\ f(\lambda g.g\ x\ y)$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \mathsf{curry} &:= \lambda f, x, y.\ f\langle x, y\rangle &= \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \mathsf{uncurry} &:= \lambda g, \langle x, y\rangle.\ g\ x\ y \end{aligned}$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned}
\text{curry} &:= \lambda f, x, y.\ f\langle x, y\rangle && = \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\
\text{uncurry} &:= \lambda g, \langle x, y\rangle.\ g\ x\ y && = \lambda g, p.p\ g
\end{aligned}$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{array}{rcl} \mathsf{curry} & := & \lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}\langle \mathsf{x}, \mathsf{y}\rangle & = & \lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\lambda \mathsf{g}.\mathsf{g}\ \mathsf{x}\ \mathsf{y}) \\ \mathsf{uncurry} & := & \lambda \mathsf{g}, \langle \mathsf{x}, \mathsf{y}\rangle.\ \mathsf{g}\ \mathsf{x}\ \mathsf{y} & = & \lambda \mathsf{g}, \mathsf{p}.\mathsf{p}\ \mathsf{g} \end{array}$$

Then

$$\mathsf{uncurry} \circ \mathsf{curry} = \mathbf{I}$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \mathsf{curry} &:= \lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}\langle \mathsf{x}, \mathsf{y}\rangle & = \lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\lambda \mathsf{g}.\mathsf{g}\ \mathsf{x}\ \mathsf{y}) \\ \mathsf{uncurry} &:= \lambda \mathsf{g}, \langle \mathsf{x}, \mathsf{y}\rangle.\ \mathsf{g}\ \mathsf{x}\ \mathsf{y} & = \lambda \mathsf{g}, \mathsf{p}.\mathsf{p}\ \mathsf{g} \end{aligned}$$

Then

$$\begin{aligned} &\mathsf{uncurry} \circ \mathsf{curry} = \mathbf{I} \\ &\mathsf{curry} \circ \mathsf{uncurry} \not\sqsubseteq \mathbf{I} \end{aligned}$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \mathsf{curry} &:= \lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}\langle \mathsf{x}, \mathsf{y}\rangle &&= \lambda \mathsf{f}, \mathsf{x}, \mathsf{y}.\ \mathsf{f}(\lambda \mathsf{g}.\mathsf{g}\ \mathsf{x}\ \mathsf{y}) \\ \mathsf{uncurry} &:= \lambda \mathsf{g}, \langle \mathsf{x}, \mathsf{y}\rangle.\ \mathsf{g}\ \mathsf{x}\ \mathsf{y} &&= \lambda \mathsf{g}, \mathsf{p}.\mathsf{p}\ \mathsf{g} \end{aligned}$$

Then

$$\begin{aligned} &\mathsf{uncurry} \circ \mathsf{curry} = \mathbf{I} \\ &\mathsf{curry} \circ \mathsf{uncurry} \sqsubseteq \mathbf{I} \qquad \textit{(enough)} \end{aligned}$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{array}{rcl} \mathsf{curry} & := & \lambda f, x, y.\ f\langle x, y\rangle & = & \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \mathsf{uncurry} & := & \lambda g, \langle x, y\rangle.\ g\ x\ y & = & \lambda g, p.p\ g \end{array}$$

Then

$$\mathsf{uncurry} \circ \mathsf{curry} = \mathbf{I}$$
$$\mathsf{curry} \circ \mathsf{uncurry} \ \sqsubseteq\!\!\!\!\!\mp \ \mathbf{I} \qquad \textit{(enough)}$$

For example is $q = \lambda f, \_.f(f\ \bot)\ :\ a \to a$?

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \mathsf{curry} &:= \lambda f, x, y.\ f\langle x, y\rangle & &= \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \mathsf{uncurry} &:= \lambda g, \langle x, y\rangle.\ g\ x\ y & &= \lambda g, p.p\ g \end{aligned}$$

Then

$$\begin{aligned} &\mathsf{uncurry} \circ \mathsf{curry} = \mathbf{I} \\ &\mathsf{curry} \circ \mathsf{uncurry} \;\;\sqsubseteq\!\!\!\!\!\!\!\!\;\;\; \mathbf{I} \qquad\qquad \textit{(enough)} \end{aligned}$$

For example is $q = \lambda f, \_.f(f\ \bot)\ :\ a \to a$?
How do we see the second f without diverging?

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \text{curry} &:= \lambda f, x, y.\ f\langle x, y\rangle && = \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \text{uncurry} &:= \lambda g, \langle x, y\rangle.\ g\ x\ y && = \lambda g, p.p\ g \end{aligned}$$

Then

$$\text{uncurry} \circ \text{curry} = \mathbf{I}$$
$$\text{curry} \circ \text{uncurry} \ \sqsubseteq\!\!\!\!\!\!\!\;\sqsupseteq \ \mathbf{I} \qquad \textit{(enough)}$$

For example is $q = \lambda f, \_.f(f\ \bot)\ :\ a \to a$?
How do we see the second f without diverging?

$$C \to U\ q$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \mathsf{curry} &:= \lambda f, x, y.\ f\langle x, y\rangle & &= \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \mathsf{uncurry} &:= \lambda g, \langle x, y\rangle.\ g\ x\ y & &= \lambda g, p.p\ g \end{aligned}$$

Then

$$\mathsf{uncurry} \circ \mathsf{curry} = \mathbf{I}$$
$$\mathsf{curry} \circ \mathsf{uncurry} \;\; \not\sqsubseteq \;\; \mathbf{I} \qquad \textit{(enough)}$$

For example is $q = \lambda f, \_.f(f\ \bot)\ :\ a \rightarrow a$?
How do we see the second f without diverging?

$$C \rightarrow U\ q = \lambda f.\ U\ (\lambda \_.\ C\ f\ (C\ f\ \bot))$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \mathsf{curry} &:= \lambda f, x, y.\ f\langle x, y\rangle & &= \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \mathsf{uncurry} &:= \lambda g, \langle x, y\rangle.\ g\ x\ y & &= \lambda g, p.p\ g \end{aligned}$$

Then

$$\begin{aligned} &\mathsf{uncurry} \circ \mathsf{curry} = \mathbf{I} \\ &\mathsf{curry} \circ \mathsf{uncurry} \ \sqsubseteq\ \mathbf{I} \qquad \qquad \textit{(enough)} \end{aligned}$$

For example is $q = \lambda f, \_.f(f\ \bot)\ :\ a \to a$?
How do we see the second f without diverging?

$$\begin{aligned} C \to U\ q &= \lambda f.\ U\ (\lambda\_.\ C\ f\ (C\ f\ \bot)) \\ &= \lambda f.\ (U \circ C)\ \lambda\_.f\ (C\ f\ \bot) \end{aligned}$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{aligned} \text{curry} &:= \lambda f, x, y.\ f\langle x, y\rangle & &= \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \text{uncurry} &:= \lambda g, \langle x, y\rangle.\ g\ x\ y & &= \lambda g, p.p\ g \end{aligned}$$

Then

$$\begin{aligned} &\text{uncurry} \circ \text{curry} = \mathbf{I} \\ &\text{curry} \circ \text{uncurry} \ \sqsubseteq\!\!\!\!\sqsupseteq\ \mathbf{I} \qquad \textit{(enough)} \end{aligned}$$

For example is $q = \lambda f, \_.f(f\ \bot)\ :\ a \to a$?
How do we see the second f without diverging?

$$\begin{aligned} C \to U\ q &= \lambda f.\ U\ (\lambda \_.\ C\ f\ (C\ f\ \bot)) \\ &= \lambda f.\ (U \circ C)\ \lambda \_.f\ (C\ f\ \bot) \\ &= \lambda f, \_.\ f\ (\lambda x.\ f\ \langle \bot, x\rangle) \end{aligned}$$

Protecting terms from divergence

We'll also need to make terms temporarily inert

$$\begin{array}{rcll} \text{curry} & := & \lambda f, x, y.\ f\langle x, y\rangle & = \ \lambda f, x, y.\ f(\lambda g.g\ x\ y) \\ \text{uncurry} & := & \lambda g, \langle x, y\rangle.\ g\ x\ y & = \ \lambda g, p.p\ g \end{array}$$

Then

$$\text{uncurry}\circ\text{curry} = \mathbf{I}$$
$$\text{curry}\circ\text{uncurry} \ \sqsubseteq\!\!\!\!\!\!\; \ \mathbf{I} \qquad \textit{(enough)}$$

For example is $q = \lambda f, \_.f(f\ \bot)\ :\ a \to a$?
How do we see the second f without diverging?

$$\begin{array}{rcl} C \to U\ q & = & \lambda f.\ U\ (\lambda\_.\ C\ f\ (C\ f\ \bot)) \\ & = & \lambda f.\ (U \circ C)\ \lambda\_.f\ (C\ f\ \bot) \\ & = & \lambda f, \_.\ f\ (\lambda x.\ f\ \langle \bot, x\rangle) \end{array}$$

Closing this operation: type $C \to U\ q = \lambda f, \_.f\ \top$.

Constructing simple concurrent types

Generalize to functors of mixed variance:
join over all sorts of section-retract pairs.

Constructing simple concurrent types

Generalize to functors of mixed variance:
join over all sorts of section-retract pairs.

$$\text{Simple} \;:=\; \text{any} \rightarrow \text{type}\ ($$

Constructing simple concurrent types

Generalize to functors of mixed variance:
join over all sorts of section-retract pairs.

$$\text{Simple} \ := \ \text{any} \to \text{type} \ (\\ \lambda\text{f. f curry uncurry}$$

Constructing simple concurrent types

Generalize to functors of mixed variance:
join over all sorts of section-retract pairs.

$$
\begin{aligned}
\mathsf{Simple} \;:=\; & \mathsf{any} \rightarrow \mathsf{type}\ (\\
& \lambda \mathsf{f}.\ \mathsf{f}\ \mathsf{curry}\ \mathsf{uncurry} \\
& \quad \mid \bigsqcup \mathsf{m}, \mathsf{n} \geq 0.\ \mathsf{f}\ \mathsf{R}_{\mathsf{mn}}\ \mathsf{L}_{\mathsf{mn}}
\end{aligned}
$$

Constructing simple concurrent types

Generalize to functors of mixed variance:
join over all sorts of section-retract pairs.

$$\begin{aligned}\mathsf{Simple} \;:=\; & \mathsf{any} \to \mathsf{type}\ (\\ & \quad \lambda f.\ f\ \mathsf{curry}\ \mathsf{uncurry} \\ & \qquad \mid \bigsqcup m, n \geq 0.\ f\ R_{mn}\ L_{mn} \\ &).\end{aligned}$$

▸ alternate definition

For example

$$\begin{aligned}\mathsf{div} &= \mathsf{Simple}\ \lambda a, a'.\ a' \\ \mathsf{nat} &<: \mathsf{Simple}\ \lambda a, a'.\ (a' \to a) \to a \to a'\end{aligned}$$

Constructing simple concurrent types

Generalize to functors of mixed variance:
join over all sorts of section-retract pairs.

$$\begin{aligned} \mathsf{Simple} \;:=\; & \mathsf{any} \to \mathsf{type}\; (\\ & \quad \lambda f.\; f\ \mathsf{curry}\ \mathsf{uncurry} \\ & \qquad \mid \bigsqcup m, n \geq 0.\; f\ R_{mn}\ L_{mn} \\ &). \end{aligned}$$

▸ alternate definition

For example

$$\begin{aligned} \mathsf{div} &= \mathsf{Simple}\ \lambda a, a'.\ a' \\ \mathsf{nat} &<: \mathsf{Simple}\ \lambda a, a'.\ (a' \to a) \to a \to a' \\ \mathsf{Prod} &<: \lambda a{:}\mathsf{type}, b{:}\mathsf{type}.\ \mathsf{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c' \end{aligned}$$

Constructing simple concurrent types

Generalize to functors of mixed variance:
join over all sorts of section-retract pairs.

$$\begin{aligned}\mathsf{Simple} \;&:=\; \mathsf{any} \to \mathsf{type}\ (\\ &\quad \lambda f.\ f\ \mathsf{curry}\ \mathsf{uncurry}\\ &\qquad \mid \bigsqcup m, n \geq 0.\ f\ R_{mn}\ L_{mn}\\ &).\end{aligned}$$

▸ alternate definition

For example

$$\begin{aligned}\mathsf{div} &= \mathsf{Simple}\ \lambda a, a'.\ a'\\ \mathsf{nat} &<:\ \mathsf{Simple}\ \lambda a, a'.\ (a' \to a) \to a \to a'\\ \mathsf{Prod} &<:\ \lambda a{:}\mathsf{type}, b{:}\mathsf{type}.\ \mathsf{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c'\end{aligned}$$

This is amost enough, but there may be $\top$'s in the body.

Checking the body for errors

We saw (Simple $\lambda a, a'.\ a \rightarrow a'$) $\lambda f, \_.f(f\ \bot)\ =\ \lambda f, \_.f\ \top$.

Checking the body for errors

We saw (Simple $\lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}'$) $\lambda \mathsf{f}, \_.\mathsf{f}(\mathsf{f}\ \bot) \ = \ \lambda \mathsf{f}, \_.\mathsf{f}\ \top$.
But is $\lambda \mathsf{f}, \_.\mathsf{f}\ \top \ :\ \mathsf{a} \to \mathsf{a}$?

Checking the body for errors

We saw (Simple λa, a′. a → a′) λf, _.f(f $\bot$) = λf, _.f $\top$.
But is λf, _.f $\top$: a → a?

Try combining intro and elim forms: λx.x $\mathbf{I} = \langle \mathbf{I} \rangle$.

Checking the body for errors

We saw (Simple λa, a′. a → a′) λf, _.f(f $\bot$) = λf, _.f $\top$.
But is λf, _.f $\top$: a → a?

Try combining intro and elim forms: λx.x $\mathbf{I} = \langle\mathbf{I}\rangle$.

$$\langle\mathbf{I}\rangle\ (\lambda \mathsf{f}, \_.\ \mathsf{f}\ \top)$$

Checking the body for errors

We saw (Simple λa, a′. a → a′) λf, _.f(f $\bot$) $=$ λf, _.f $\top$.
But is λf, _.f $\top$: a → a?

Try combining intro and elim forms: λx.x $\mathbf{I} = \langle\mathbf{I}\rangle$.

$$\langle\mathbf{I}\rangle\ (\lambda \text{f}, \_.\ \text{f}\ \top)\ =\ \lambda \text{x}.\ \mathbf{I}\ \top$$

Checking the body for errors

We saw (Simple $\lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}'$) $\lambda \mathsf{f}, \_.\mathsf{f}(\mathsf{f}\ \bot)\ =\ \lambda \mathsf{f}, \_.\mathsf{f}\ \top$.
But is $\lambda \mathsf{f}, \_.\mathsf{f}\ \top\ :\ \mathsf{a} \to \mathsf{a}$?

Try combining intro and elim forms: $\lambda \mathsf{x}.\mathsf{x}\ \mathbf{I} = \langle \mathbf{I} \rangle$.

$$\langle \mathbf{I} \rangle\ (\lambda \mathsf{f}, \_.\ \mathsf{f}\ \top)\ =\ \lambda \mathsf{x}.\ \mathbf{I}\ \top \qquad =\ \top$$

What about numerals?

Checking the body for errors

We saw (Simple $\lambda \mathsf{a}, \mathsf{a}'. \ \mathsf{a} \to \mathsf{a}'$) $\lambda \mathsf{f}, \_.\mathsf{f}(\mathsf{f} \ \bot) \ = \ \lambda \mathsf{f}, \_.\mathsf{f} \ \top$.
But is $\lambda \mathsf{f}, \_.\mathsf{f} \ \top \ : \ \mathsf{a} \to \mathsf{a}$?

Try combining intro and elim forms: $\lambda \mathsf{x}.\mathsf{x} \ \mathbf{I} = \langle \mathbf{I} \rangle$.

$$\langle \mathbf{I} \rangle \ (\lambda \mathsf{f}, \_. \ \mathsf{f} \ \top) \ = \ \lambda \mathsf{x}. \ \mathbf{I} \ \top \qquad = \ \top$$

What about numerals? Is $\lambda \mathsf{f}, \_.\mathsf{f}(\ldots(\mathsf{f} \ \top)\ldots) \ : \ \mathsf{nat}$?

Checking the body for errors

We saw (Simple $\lambda a, a'.\ a \rightarrow a'$) $\lambda f, \_.f(f\ \bot) \ = \ \lambda f, \_.f\ \top$.
But is $\lambda f, \_.f\ \top\ :\ a \rightarrow a$?

Try combining intro and elim forms: $\lambda x.x\ \mathbf{I} = \langle \mathbf{I} \rangle$.

$$\langle \mathbf{I} \rangle\ (\lambda f, \_.\ f\ \top) \ = \ \lambda x.\ \mathbf{I}\ \top \qquad = \ \top$$

What about numerals? Is $\lambda f, \_.f(\ldots(f\ \top)\ldots)\ :\ nat$?
Try intro and elim forms: $\lambda n.n\ succ\ zero = \langle succ, zero \rangle$.

Checking the body for errors

We saw (Simple $\lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}'$) $\lambda \mathsf{f}, \_.\mathsf{f}(\mathsf{f}\ \bot) \ = \ \lambda \mathsf{f}, \_.\mathsf{f}\ \top$.
But is $\lambda \mathsf{f}, \_.\mathsf{f}\ \top\ :\ \mathsf{a} \to \mathsf{a}$?

Try combining intro and elim forms: $\lambda \mathsf{x}.\mathsf{x}\ \mathbf{I} = \langle \mathbf{I} \rangle$.

$$\langle \mathbf{I} \rangle\ (\lambda \mathsf{f}, \_.\ \mathsf{f}\ \top) \ = \ \lambda \mathsf{x}.\ \mathbf{I}\ \top \qquad = \ \top$$

What about numerals? Is $\lambda \mathsf{f}, \_.\mathsf{f}(\ldots(\mathsf{f}\ \top)\ldots)\ :\ \mathsf{nat}$?
Try intro and elim forms: $\lambda \mathsf{n}.\mathsf{n}\ \mathsf{succ}\ \mathsf{zero} = \langle \mathsf{succ}, \mathsf{zero} \rangle$.

$$\langle \mathsf{s}, \mathsf{z} \rangle\ \lambda \mathsf{f}, \_.\mathsf{f}(\ldots(\mathsf{f}\ \top)\ldots) = \mathsf{s}(\ldots(\mathsf{s}\ \top)\ldots)$$

Checking the body for errors

We saw (Simple $\lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}'$) $\lambda \mathsf{f}, \_.\mathsf{f}(\mathsf{f}\ \bot) \ = \ \lambda \mathsf{f}, \_.\mathsf{f}\ \top$.
But is $\lambda \mathsf{f}, \_.\mathsf{f}\ \top\ :\ \mathsf{a} \to \mathsf{a}$?

Try combining intro and elim forms: $\lambda \mathsf{x}.\mathsf{x}\ \mathbf{I} = \langle \mathbf{I} \rangle$.

$$\langle \mathbf{I} \rangle\ (\lambda \mathsf{f}, \_.\ \mathsf{f}\ \top) \ = \ \lambda \mathsf{x}.\ \mathbf{I}\ \top \qquad = \ \top$$

What about numerals? Is $\lambda \mathsf{f}, \_.\mathsf{f}(\ldots(\mathsf{f}\ \top)\ldots)\ :\ \mathsf{nat}$?
Try intro and elim forms: $\lambda \mathsf{n}.\mathsf{n}\ \mathsf{succ}\ \mathsf{zero} = \langle \mathsf{succ}, \mathsf{zero} \rangle$.

$$\langle \mathsf{s}, \mathsf{z} \rangle\ \lambda \mathsf{f}, \_.\mathsf{f}(\ldots(\mathsf{f}\ \top)\ldots) = \mathsf{s}(\ldots(\mathsf{s}\ \top)\ldots) \ = \ \top$$

Checking the body for errors

We saw (Simple $\lambda a, a'.\ a \to a'$) $\lambda f, \_.f(f\ \bot)\ =\ \lambda f, \_.f\ \top$.
But is $\lambda f, \_.f\ \top\ :\ a \to a$?

Try combining intro and elim forms: $\lambda x.x\ \mathbf{I} = \langle \mathbf{I} \rangle$.

$$\langle \mathbf{I} \rangle\ (\lambda f, \_.\ f\ \top)\ =\ \lambda x.\ \mathbf{I}\ \top\ \quad =\ \top$$

What about numerals? Is $\lambda f, \_.f(\dots(f\ \top)\dots)\ :\ nat$?
Try intro and elim forms: $\lambda n.n\ succ\ zero = \langle succ, zero \rangle$.

$$\langle s, z \rangle\ \lambda f, \_.f(\dots(f\ \top)\dots) = s(\dots(s\ \top)\dots)\ =\ \top$$

This is enough: descend into body with intro and elim forms.

Intermezzo: concurrent head normal form

Definition

A head normal form is a λ-term

$$\lambda x_1, \ldots, x_v.\ x\ M_1\ \ \ldots\ \ M_a$$

Intermezzo: concurrent head normal form

Definition

A head normal form is a λ-term

$$\lambda \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{v}.\ \mathsf{x}\ \mathsf{M}_1\ \ \ldots\ \ \mathsf{M}_\mathsf{a}$$

where $\mathsf{a}, \mathsf{v} \geq 0$, and $\mathsf{M}_1, \ldots, \mathsf{M}_\mathsf{a}$ are concurrent λ-terms.

Intermezzo: concurrent head normal form

Definition

A head normal form is a λ-term

$$\lambda x_1, \ldots, x_v.\ x\ M_1\ \ \ldots\ \ M_a$$

where $a, v \geq 0$, and $M_1, \ldots, M_a$ are concurrent λ-terms.
Call x the head variable, and $M_1, \ldots, M_a$ the body.

Definition

A concurrent Böhm tree is a h.n.f.
where the M's are recursively joins of BT's.

Intermezzo: concurrent head normal form

Definition

A head normal form is a λ-term

$$\lambda x_1, \ldots, x_v.\ x\ M_1\ \ \ldots\ \ M_a$$

where $a, v \geq 0$, and $M_1, \ldots, M_a$ are concurrent λ-terms.
Call x the head variable, and $M_1, \ldots, M_a$ the body.

Definition

A concurrent Böhm tree is a h.n.f.
where the M's are recursively joins of BT's.

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

Intermezzo: concurrent head normal form

Definition

A head normal form is a λ-term

$$\lambda x_1, \ldots, x_v.\ x\ M_1\ \ \ldots\ \ M_a$$

where $a, v \geq 0$, and $M_1, \ldots, M_a$ are concurrent λ-terms.
Call x the head variable, and $M_1, \ldots, M_a$ the body.

Definition

A concurrent Böhm tree is a h.n.f.
where the M's are recursively joins of BT's.

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

E.g. $\mathbf{J} = \lambda x, y.x \mid y$

Intermezzo: concurrent head normal form

Definition

A head normal form is a λ-term

$$\lambda x_1, \ldots, x_v.\ x\ M_1\ \ \ldots\ \ M_a$$

where $a, v \geq 0$, and $M_1, \ldots, M_a$ are concurrent λ-terms.
Call x the head variable, and $M_1, \ldots, M_a$ the body.

Definition

A concurrent Böhm tree is a h.n.f.
where the M's are recursively joins of BT's.

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

E.g. $\mathbf{J} = \lambda x, y.x \mid y\ = (\lambda x, y.x) \mid (\lambda x, y.y)$, by η-conversion.

Intermezzo: interpolation by head normal forms

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

Intermezzo: interpolation by head normal forms

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

- Necessary for $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable closures.

Intermezzo: interpolation by head normal forms

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

- Necessary for $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable closures.
- Fails in Scott's model: step functions.

Intermezzo: interpolation by head normal forms

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

- Necessary for $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable closures.
- Fails in Scott's model: step functions.
- Trivially true in the completed term model $\mathcal{B}$.

Intermezzo: interpolation by head normal forms

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

- Necessary for $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable closures.
- Fails in Scott's model: step functions.
- Trivially true in the completed term model $\mathcal{B}$.

Corollary

If q *converges then* q *extends a h.n.f.;*

Intermezzo: interpolation by head normal forms

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

- Necessary for $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable closures.
- Fails in Scott's model: step functions.
- Trivially true in the completed term model $\mathcal{B}$.

Corollary

If q *converges then* q *extends a h.n.f.;*

If $\mathsf{q} \not\sqsubseteq \mathsf{q}'$ *then* $\mathsf{q} \sqsupseteq \mathsf{M} \not\sqsubseteq \mathsf{q}'$ *for a h.n.f.* M.

Intermezzo: interpolation by head normal forms

Proposition

Everything is a join of h.n.f.s (modulo observability $\mathcal{H}^*$).

- Necessary for $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable closures.
- Fails in Scott's model: step functions.
- Trivially true in the completed term model $\mathcal{B}$.

Corollary

If q *converges then* q *extends a h.n.f.;*

If $\mathsf{q} \not\sqsubseteq \mathsf{q}'$ *then* $\mathsf{q} \sqsupseteq \mathsf{M} \not\sqsubseteq \mathsf{q}'$ *for a h.n.f.* M*.*

...and now for the Main Example...

Can't say no? maybe you need a...

Can't say no? maybe you need a...

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \rightarrow \mathsf{a}')\ \mid\ \langle \mathbf{I} \rangle).$$

Can't say no? maybe you need a...

semi $:=$ type ((Simple $\lambda a, a'.\ a \rightarrow a'$) $\mid$ $\langle \mathbf{I} \rangle$).

Theorem

inhab(semi) $= \{\bot, \mathbf{I}, \top\}$.

Can't say no? maybe you need a...

$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \rightarrow \mathsf{a}')\ \mid\ \langle \mathbf{I} \rangle).$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}.$

Proof.

$\bot$: semi by β-reduction.

Can't say no? maybe you need a...

$$\text{semi} := \text{type}\ ((\text{Simple}\ \lambda a, a'.\ a \to a')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\text{inhab}(\text{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \text{semi}$ by β-reduction. Any other $q : \text{semi}$ converges,

Can't say no?   maybe you need a...

$$\text{semi} \;:=\; \text{type } ((\text{Simple } \lambda a, a'.\ a \rightarrow a') \mid \langle \mathbf{I} \rangle).$$

Theorem

$\text{inhab}(\text{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \text{semi}$ by β-reduction. Any other $q : \text{semi}$ converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Can't say no? maybe you need a...

$$\text{semi} \;:=\; \text{type}\;((\text{Simple}\;\lambda a, a'.\; a \to a') \;\mid\; \langle \mathbf{I} \rangle).$$

Theorem

$\text{inhab}(\text{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \text{semi}$ by β-reduction. Any other $q : \text{semi}$ converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\; z\; M_1 \;\ldots\; M_m$$

Show that either $q = \top$

Can't say no? maybe you need a...

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a') \mid \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot$: semi by β-reduction. Any other q : semi converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$ or

$$z = f,$$

Can't say no?   maybe you need a...

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}') \;\mid\; \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \mathsf{semi}$ by β-reduction. Any other $\mathsf{q} : \mathsf{semi}$ converges, say

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}.\ \mathsf{z}\ \mathsf{M}_1\ \ldots\ \mathsf{M}_\mathsf{m}$$

Show that either $\mathsf{q} = \top$ or

$$\mathsf{z} = \mathsf{f}, \quad \mathsf{m} = \mathsf{n}$$ *(use minefields),*

Can't say no? maybe you need a...

$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}')\ \mid\ \langle \mathbf{I} \rangle).$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}.$

Proof.

$\bot$: semi by β-reduction. Any other q : semi converges, say

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}.\ \mathsf{z}\ \mathsf{M}_1\ \ldots\ \mathsf{M}_\mathsf{m}$$

Show that either $\mathsf{q} = \top$ or

$\mathsf{z} = \mathsf{f}, \quad \mathsf{m} = \mathsf{n}$ *(use minefields),*

$\mathsf{M}_\mathsf{i} \sqsubseteq \mathsf{x}_\mathsf{i}$ *(descend, minefields, curry).*

Can't say no?   maybe you need a...

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \mathsf{semi}$ by β-reduction. Any other $q : \mathsf{semi}$ converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$ or

$z = f, \quad m = n$ *(use minefields),*

$M_i \sqsubseteq x_i$ *(descend, minefields, curry).*

Finally raise q' up to $\mathbf{I}$ with minefields.

Can't say no? maybe you need a...

$$\mathsf{semi} := \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \mathsf{semi}$ by β-reduction. Any other $q : \mathsf{semi}$ converges, say

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$ or

$z = f, \quad m = n$ *(use minefields),*

$M_i \sqsubseteq x_i$ *(descend, minefields, curry).*

Finally raise q' up to $\mathbf{I}$ with minefields. details

□

Enforcing sequentiality

Consider a bad definition of bool

$\mathsf{boool} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a} \to \mathsf{a}') \mid \langle \mathbf{K}, \mathbf{F} \rangle).$

Enforcing sequentiality

Consider a bad definition of bool
boool $:=$ type ((Simple λa, a$'$. a $\rightarrow$ a $\rightarrow$ a$'$) $\mid$ $\langle \mathbf{K}, \mathbf{F} \rangle$).

Theorem

inhab(boool) $= \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$. *(recall* $\mathbf{J} = \mathbf{K} \mid \mathbf{F}$*)*

Enforcing sequentiality

Consider a bad definition of bool
$\mathsf{boool} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a} \to \mathsf{a}') \mid \langle \mathbf{K}, \mathbf{F} \rangle).$

Theorem

$\mathsf{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$. *(recall* $\mathbf{J} = \mathbf{K} \mid \mathbf{F}$*)*

Proof.

Similar to semi, but now q can extend two h.n.f.'s:

Enforcing sequentiality

Consider a bad definition of bool
boool $:=$ type ((Simple λa, a$'$. a $\rightarrow$ a $\rightarrow$ a$'$) $\mid$ $\langle \mathbf{K}, \mathbf{F} \rangle$).

Theorem

inhab(boool) $= \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$. *(recall* $\mathbf{J} = \mathbf{K} \mid \mathbf{F}$*)*

Proof.

Similar to semi, but now q can extend two h.n.f.'s:

$$\mathsf{q} \;\sqsupseteq\; \lambda x, y.x = \mathbf{K}, \qquad \mathsf{q} \;\sqsupseteq\; \lambda x, y.y = \mathbf{F}$$

Enforcing sequentiality

Consider a bad definition of bool

boool $:=$ type ((Simple $\lambda a, a'.\ a \to a \to a'$) $\mid$ $\langle \mathbf{K}, \mathbf{F} \rangle$).

Theorem

inhab(boool) $= \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$. *(recall $\mathbf{J} = \mathbf{K} \mid \mathbf{F}$)*

Proof.

Similar to semi, but now q can extend two h.n.f.'s:

$$q \sqsupseteq \lambda x, y.x = \mathbf{K}, \qquad q \sqsupseteq \lambda x, y.y = \mathbf{F}$$

$\mathbf{J}$ extends both.

□

Enforcing sequentiality

Consider a bad definition of bool
boool $:=$ type $((\text{Simple } \lambda a, a'.\ a \to a \to a') \mid \langle \mathbf{K}, \mathbf{F} \rangle)$.

Theorem

inhab(boool) $= \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$. *(recall* $\mathbf{J} = \mathbf{K} \mid \mathbf{F}$*)*

Proof.

Similar to semi, but now q can extend two h.n.f.'s:

$$q \sqsupseteq \lambda x, y.x = \mathbf{K}, \qquad q \sqsupseteq \lambda x, y.y = \mathbf{F}$$

$\mathbf{J}$ extends both.

□

How to ensure sequentiality?

Enforcing sequentiality

Consider a bad definition of bool
$\mathsf{boool} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a} \to \mathsf{a}') \ \mid\ \langle \mathbf{K}, \mathbf{F} \rangle).$

Theorem

$\mathsf{inhab}(\mathsf{boool}) = \{\bot, \mathbf{K}, \mathbf{F}, \mathbf{J}, \top\}$. *(recall $\mathbf{J} = \mathbf{K} \mid \mathbf{F}$)*

Proof.

Similar to semi, but now q can extend two h.n.f.'s:

$$\mathsf{q} \;\sqsupseteq\; \lambda \mathsf{x}, \mathsf{y}.\mathsf{x} = \mathbf{K}, \qquad\qquad \mathsf{q} \;\sqsupseteq\; \lambda \mathsf{x}, \mathsf{y}.\mathsf{y} = \mathbf{F}$$

$\mathbf{J}$ extends both.

□

How to ensure sequentiality? Make q decide.

Corrected definition of bool

$$\text{make\_up\_your\_mind} \;:=\; \lambda \text{q. q} \perp (\text{q} \top \perp).$$

Corrected definition of bool

$$\text{make\_up\_your\_mind} \;:=\; \lambda \text{q. q} \perp (\text{q} \top \perp).$$

$$\text{bool} \;:=\; \text{type (boool} \;|\; \text{make\_up\_your\_mind).}$$

Corrected definition of bool

make_up_your_mind := λq. q $\perp$ (q $\top$ $\perp$).
bool := type (boool | make_up_your_mind).

Theorem

inhab(bool) $= \{\perp, \mathbf{K}, \mathbf{F}, \top\}$.

Corrected definition of bool

make_up_your_mind := λq. q $\bot$ (q $\top$ $\bot$).
bool := type (boool | make_up_your_mind).

Theorem

inhab(bool) $= \{\bot, \mathbf{K}, \mathbf{F}, \top\}$.

Proof.

Since bool $<:$ boool, we need only check inhabitants.

Corrected definition of bool

$$\text{make\_up\_your\_mind} \;:=\; \lambda \text{q. q} \perp (\text{q} \top \perp).$$
$$\text{bool} \;:=\; \text{type (boool} \;|\; \text{make\_up\_your\_mind).}$$

Theorem

inhab(bool) $= \{\perp, \mathbf{K}, \mathbf{F}, \top\}$.

Proof.

Since bool $<:$ boool, we need only check inhabitants.
All but $\mathbf{J}$ are fixed by make_up_your_mind:

$$(\lambda \text{q. q} \perp (\text{q} \top \perp))\ \mathbf{J}$$

Corrected definition of bool

$$\text{make\_up\_your\_mind} := \lambda \text{q. q} \perp (\text{q} \top \perp).$$
$$\text{bool} := \text{type (boool} \mid \text{make\_up\_your\_mind).}$$

Theorem

$\text{inhab(bool)} = \{\perp, \mathbf{K}, \mathbf{F}, \top\}$.

Proof.

Since bool $<:$ boool, we need only check inhabitants.
All but $\mathbf{J}$ are fixed by make_up_your_mind:

$$\begin{aligned}(\lambda \text{q. q} \perp (\text{q} \top \perp))\ \mathbf{J} \\ = \mathbf{J} \perp (\mathbf{J} \top \perp)\end{aligned}$$

Corrected definition of bool

$$\text{make\_up\_your\_mind} \;:=\; \lambda \text{q. q} \perp (\text{q} \top \perp).$$
$$\text{bool} \;:=\; \text{type (boool} \mid \text{make\_up\_your\_mind).}$$

Theorem

$\text{inhab(bool)} = \{\perp, \mathbf{K}, \mathbf{F}, \top\}.$

Proof.

Since bool $<:$ boool, we need only check inhabitants.
All but $\mathbf{J}$ are fixed by make_up_your_mind:

$$\begin{aligned} &(\lambda \text{q. q} \perp (\text{q} \top \perp))\ \mathbf{J} \\ &\quad = \mathbf{J} \perp (\mathbf{J} \top \perp) \\ &\quad = \perp \mid \top \mid \perp \end{aligned}$$

Corrected definition of bool

$$\text{make\_up\_your\_mind} \;:=\; \lambda \text{q. q} \perp (\text{q} \top \perp).$$
$$\text{bool} \;:=\; \text{type (boool} \mid \text{make\_up\_your\_mind).}$$

Theorem

$\text{inhab(bool)} = \{\perp, \mathbf{K}, \mathbf{F}, \top\}$.

Proof.

Since bool $<:$ boool, we need only check inhabitants.
All but $\mathbf{J}$ are fixed by make_up_your_mind:

$$\begin{aligned}
&(\lambda \text{q. q} \perp (\text{q} \top \perp))\ \mathbf{J} \\
&\quad = \mathbf{J} \perp (\mathbf{J} \top \perp) \\
&\quad = \perp \mid \top \mid \perp \\
&\quad = \top
\end{aligned}$$

□

Outline of constructing types

This technique generalizes to more complicated types.

$$
\begin{array}{l}
\mathsf{bool} \;:=\; \mathsf{type}\ (\\
\quad (\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \rightarrow \mathsf{a} \rightarrow \mathsf{a}') \\
\quad \mid\ \langle \mathbf{K}, \mathbf{F} \rangle \\
\quad \mid\ \lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot) \\
).
\end{array}
$$

Outline of constructing types

This technique generalizes to more complicated types.

$$\begin{aligned} \mathsf{bool} \ :=\ & \mathsf{type}\ (\\ & \quad \textcolor{red}{(\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \rightarrow \mathsf{a} \rightarrow \mathsf{a}')} \\ & \quad \mid\ \langle \mathbf{K}, \mathbf{F} \rangle \\ & \quad \mid\ \lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot) \\ &). \end{aligned}$$

(1) enforce simple concurrent typing

Outline of constructing types

This technique generalizes to more complicated types.

$$
\begin{aligned}
&\mathsf{bool} \;:=\; \mathsf{type}\ (\\
&\quad (\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a} \to \mathsf{a}')\\
&\quad \mid\ \langle \mathbf{K}, \mathbf{F} \rangle\\
&\quad \mid\ \lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot)\\
&).
\end{aligned}
$$

(1) enforce simple concurrent typing

(2) descend Böhm tree with intro and elim forms

Outline of constructing types

This technique generalizes to more complicated types.

$$\begin{aligned}
\mathsf{bool} \;&:=\; \mathsf{type}\ (\\
&\quad (\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a} \to \mathsf{a}')\\
&\quad \mid\ \langle \mathbf{K}, \mathbf{F} \rangle\\
&\quad \mid\ \lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot)\\
).
\end{aligned}$$

(1) enforce simple concurrent typing

(2) descend Böhm tree with intro and elim forms

(3) enforce sequentiality: one head variable only

Outline of constructing types

This technique generalizes to more complicated types.

$$
\begin{aligned}
\mathsf{bool} \;:=\; & \mathsf{type}\ (\\
& \quad (\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a} \to \mathsf{a}') \\
& \quad \mid\ \langle \mathbf{K}, \mathbf{F} \rangle \\
& \quad \mid\ \lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot) \\
&).
\end{aligned}
$$

(1) enforce simple concurrent typing

(2) descend Böhm tree with intro and elim forms

(3) enforce sequentiality: one head variable only

Product types (actually dropped product)

$$
\begin{aligned}
&\mathsf{Prod} \;:=\; \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
&\quad \lambda \mathsf{a}, \mathsf{b}.\ (\mathsf{Simple}\ \lambda \mathsf{c}, \mathsf{c}'.\ (\mathsf{a} \to \mathsf{b} \to \mathsf{c}) \to \mathsf{c}')\\
&\qquad\quad \mid\ \langle \lambda \mathsf{x}, \mathsf{y}. \langle \mathsf{x}, \mathsf{y} \rangle \rangle\\
&\qquad\quad \mid\ \lambda \mathsf{q}.\ \langle \mathsf{q}\ \mathbf{K},\ \mathsf{q}\ \mathbf{F} \rangle\\
&).
\end{aligned}
$$

Product types (actually dropped product)

$$
\begin{aligned}
\mathsf{Prod} \;:=\; & \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
& \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c')\\
& \quad \mid\ \langle \lambda x, y. \langle x, y\rangle\rangle\\
& \quad \mid\ \lambda q.\ \langle q\ \mathbf{K},\ q\ \mathbf{F}\rangle\\
).
\end{aligned}
$$

Theorem

For $a, b : \mathsf{type}$, $\mathsf{inhab}(\mathsf{Prod}\ a\ b) = \{\top\} \cup \{\langle x, y\rangle \mid x : a, y : b\}$

Product types (actually dropped product)

$$
\begin{aligned}
\mathsf{Prod} \;:=\; & \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
& \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c') \\
& \quad \mid\ \langle \lambda x, y. \langle x, y \rangle \rangle \\
& \quad \mid\ \lambda q.\ \langle q\ \mathbf{K},\ q\ \mathbf{F} \rangle \\
).
\end{aligned}
$$

Theorem

For $a, b : \mathsf{type}$, $\mathsf{inhab}(\mathsf{Prod}\ a\ b) = \{\top\} \cup \{\langle x, y \rangle \mid x : a, y : b\}$

Proof.

Any h.n.f. below $q : \mathsf{Prod}\ a\ b$ must be $\langle a\ x, b\ y \rangle$ or $\top$.

Product types (actually dropped product)

$$
\begin{aligned}
\text{Prod} \;:=\;& \text{type} \to \text{type} \to \text{type}\ (\\
& \lambda a, b.\ (\text{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c')\\
& \qquad \mid\ \langle \lambda x, y. \langle x, y \rangle \rangle\\
& \qquad \mid\ \lambda q.\ \langle q\ \mathbf{K},\ q\ \mathbf{F} \rangle\\
).&
\end{aligned}
$$

Theorem

For $a, b{:}\text{type}$, $\text{inhab}(\text{Prod}\ a\ b) = \{\top\} \cup \{\langle x, y\rangle \mid x{:}a, y{:}b\}$

Proof.

Any h.n.f. below $q{:}\text{Prod}\ a\ b$ must be $\langle a\ x, b\ y\rangle$ or $\top$.
What is the maximal such?

Product types (actually dropped product)

$$
\begin{array}{l}
\mathsf{Prod} \;:=\; \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
\quad \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c')\\
\qquad \mid\ \langle \lambda x, y. \langle x, y \rangle \rangle\\
\qquad \mid\ \lambda q.\ \langle q\ \mathbf{K},\ \ q\ \mathbf{F} \rangle\\
).
\end{array}
$$

Theorem

For $a, b : \mathsf{type}$, $\mathsf{inhab}(\mathsf{Prod}\ a\ b) = \{\top\} \cup \{\langle x, y \rangle \mid x : a, y : b\}$

Proof.

Any h.n.f. below $q : \mathsf{Prod}\ a\ b$ must be $\langle a\ x, b\ y \rangle$ or $\top$.
What is the maximal such?
First component is $q\ \mathbf{K}$.

Product types (actually dropped product)

$$
\begin{aligned}
\mathsf{Prod} \;:=\; & \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
& \quad \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c') \\
& \qquad\quad \mid\ \langle \lambda x, y. \langle x, y \rangle \rangle \\
& \qquad\quad \mid\ \lambda q.\ \langle q\ \mathbf{K},\ q\ \mathbf{F} \rangle \\
).
\end{aligned}
$$

Theorem

For $a, b : \mathsf{type}$, $\mathsf{inhab}(\mathsf{Prod}\ a\ b) = \{\top\} \cup \{\langle x, y \rangle \mid x : a, y : b\}$

Proof.

Any h.n.f. below $q : \mathsf{Prod}\ a\ b$ must be $\langle a\ x, b\ y \rangle$ or $\top$.
What is the maximal such?
First component is $q\ \mathbf{K}$. Second component is $q\ \mathbf{F}$.

Product types (actually dropped product)

$$
\begin{array}{l}
\mathsf{Prod} \;:=\; \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
\quad \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to b \to c) \to c') \\
\qquad\quad \mid\ \langle \lambda x, y. \langle x, y \rangle \rangle \\
\qquad\quad \mid\ \lambda q.\ \langle q\ \mathbf{K},\ q\ \mathbf{F} \rangle \\
).
\end{array}
$$

Theorem

For $a, b : \mathsf{type}$, $\mathsf{inhab}(\mathsf{Prod}\ a\ b) = \{\top\} \cup \{\langle x, y \rangle \mid x : a, y : b\}$

Proof.

Any h.n.f. below $q : \mathsf{Prod}\ a\ b$ must be $\langle a\ x, b\ y \rangle$ or $\top$.
What is the maximal such?
First component is $q\ \mathbf{K}$. Second component is $q\ \mathbf{F}$.
So $\lambda q.\ \langle q\ \mathbf{K},\ q\ \mathbf{F} \rangle$ ensures sequentiality. □

Sum types (actually dropped, lifted sum)

$$
\begin{aligned}
\mathsf{Sum} \;:=\; & \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
& \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to c) \to (b \to c) \to c') \\
& \quad \mid\ \langle \mathsf{inl},\ \mathsf{inr} \rangle \\
& \quad \mid\ \lambda q, f, g.\ q\ (\mathbf{K}\ \mathbf{I})\ \bot\ (q\ f\ \top) \\
& \qquad\qquad \mid\ q\ \bot\ (\mathbf{K}\ \mathbf{I})\ (q\ \top\ g) \\
).
\end{aligned}
$$

Sum types (actually dropped, lifted sum)

$$
\begin{array}{lll}
\mathsf{Sum} & := & \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
& \lambda a, b. & (\mathsf{Simple}\ \lambda c, c'.\ (a \to c) \to (b \to c) \to c') \\
& & \mid\ \langle \mathsf{inl},\ \mathsf{inr} \rangle \\
& & \mid\ \lambda q, f, g.\ q\ (\mathbf{K}\ \mathbf{I})\ \bot\ (q\ f\ \top) \\
& & \qquad\qquad \mid\ q\ \bot\ (\mathbf{K}\ \mathbf{I})\ (q\ \top\ g) \\
). & &
\end{array}
$$

where

$$\mathsf{inl} = \lambda x, f, \_.f\ x, \qquad \mathsf{inr} = \lambda y, \_, g.g\ x$$

Sum types (actually dropped, lifted sum)

$$
\begin{aligned}
\mathsf{Sum} \;\; &:= \;\; \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
&\quad \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to c) \to (b \to c) \to c')\\
&\qquad\quad \mid\ \langle \mathsf{inl},\ \mathsf{inr} \rangle\\
&\qquad\quad \mid\ \lambda q, f, g.\ q\ (\mathbf{K}\ \mathbf{I})\ \bot\ (q\ f\ \top)\\
&\qquad\qquad\qquad \mid\ q\ \bot\ (\mathbf{K}\ \mathbf{I})\ (q\ \top\ g)\\
&).
\end{aligned}
$$

where

$$\mathsf{inl} = \lambda x, f, \_.f\ x, \qquad \mathsf{inr} = \lambda y, \_, g.g\ x$$

Theorem

$\mathsf{inhab}(\mathsf{Sum}\ a\ b) = \{\top, \bot\} \cup \{\mathsf{inl}\ x \mid x : a\} \cup \{\mathsf{inr}\ y \mid y : b\}.$

Sum types (actually dropped, lifted sum)

$$
\begin{array}{l}
\mathsf{Sum} \;:=\; \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
\quad \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to c) \to (b \to c) \to c')\\
\qquad\quad \mid\ \langle \mathsf{inl},\ \mathsf{inr} \rangle\\
\qquad\quad \mid\ \lambda q, f, g.\ q\ (\mathbf{K}\ \mathbf{I})\ \bot\ (q\ f\ \top)\\
\qquad\qquad\qquad \mid\ q\ \bot\ (\mathbf{K}\ \mathbf{I})\ (q\ \top\ g)\\
).
\end{array}
$$

where

$$\mathsf{inl} = \lambda x, f, \_.f\ x, \qquad \mathsf{inr} = \lambda y, \_, g.g\ x$$

Theorem

$\mathsf{inhab}(\mathsf{Sum}\ a\ b) = \{\top, \bot\} \cup \{\mathsf{inl}\ x \mid x{:}a\} \cup \{\mathsf{inr}\ y \mid y{:}b\}.$

Proof.

Combine proofs of bool

Sum types (actually dropped, lifted sum)

$$
\begin{aligned}
\mathsf{Sum} \;:=\; & \mathsf{type} \to \mathsf{type} \to \mathsf{type}\ (\\
& \lambda a, b.\ (\mathsf{Simple}\ \lambda c, c'.\ (a \to c) \to (b \to c) \to c') \\
& \qquad \mid\ \langle \mathsf{inl},\ \mathsf{inr} \rangle \\
& \qquad \mid\ \lambda q, f, g.\ q\ (\mathbf{K}\ \mathbf{I})\ \bot\ (q\ f\ \top) \\
& \qquad\qquad\qquad \mid\ q\ \bot\ (\mathbf{K}\ \mathbf{I})\ (q\ \top\ g) \\
).
\end{aligned}
$$

where

$$
\mathsf{inl} = \lambda x, f, \_.f\ x, \qquad \mathsf{inr} = \lambda y, \_, g.g\ x
$$

Theorem

$\mathsf{inhab}(\mathsf{Sum}\ a\ b) = \{\top, \bot\} \cup \{\mathsf{inl}\ x \mid x{:}a\} \cup \{\mathsf{inr}\ y \mid y{:}b\}.$

Proof.

Combine proofs of bool and Prod.

□

Self-recursing numerals: motivation

Church numerals have simple type $(a \to a) \to a \to a$.

Self-recursing numerals: motivation

Church numerals have simple type $(a \to a) \to a \to a$.
but predecessor has problems:

- on well-defined terms, it is linear-time.

Self-recursing numerals: motivation

Church numerals have simple type $(a \rightarrow a) \rightarrow a \rightarrow a$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Self-recursing numerals: motivation

Church numerals have simple type $(\mathsf{a} \longrightarrow \mathsf{a}) \longrightarrow \mathsf{a} \longrightarrow \mathsf{a}$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $\mathsf{nat} \longrightarrow (\mathsf{nat} \longrightarrow \mathsf{a} \longrightarrow \mathsf{a}) \longrightarrow \mathsf{a} \longrightarrow \mathsf{a}$.

Self-recursing numerals: motivation

Church numerals have simple type $(\mathsf{a} \rightarrow \mathsf{a}) \rightarrow \mathsf{a} \rightarrow \mathsf{a}$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $\mathsf{nat} \rightarrow (\mathsf{nat} \rightarrow \mathsf{a} \rightarrow \mathsf{a}) \rightarrow \mathsf{a} \rightarrow \mathsf{a}$.
For self recursion, redefine $\mathsf{nat} = \mu\ \mathsf{n}.\ (\mathsf{n} \rightarrow \mathsf{a} \rightarrow \mathsf{a}) \rightarrow \mathsf{a} \rightarrow \mathsf{a}$:

Self-recursing numerals: motivation

Church numerals have simple type $(a \to a) \to a \to a$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $nat \to (nat \to a \to a) \to a \to a$.
For self recursion, redefine $nat = \mu\ n.\ (n \to a \to a) \to a \to a$:

$$zero \;:=\; \lambda\_, x.\ x.$$

Self-recursing numerals: motivation

Church numerals have simple type $(a \to a) \to a \to a$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $nat \to (nat \to a \to a) \to a \to a$.
For self recursion, redefine $nat = \mu\ n.\ (n \to a \to a) \to a \to a$:

$$\begin{aligned} zero &:= \lambda\_, x.\ x. \\ succ &:= \lambda n, f, x.\ n\ f\ (f\ n\ x) \end{aligned}$$

Self-recursing numerals: motivation

Church numerals have simple type $(\mathsf{a} \to \mathsf{a}) \to \mathsf{a} \to \mathsf{a}$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $\mathsf{nat} \to (\mathsf{nat} \to \mathsf{a} \to \mathsf{a}) \to \mathsf{a} \to \mathsf{a}$.
For self recursion, redefine $\mathsf{nat} = \mu\ \mathsf{n}.\ (\mathsf{n} \to \mathsf{a} \to \mathsf{a}) \to \mathsf{a} \to \mathsf{a}$:

$$\begin{aligned} \mathsf{zero} &:= \lambda\_, \mathsf{x}.\ \mathsf{x}. \\ \mathsf{succ} &:= \lambda \mathsf{n}, \mathsf{f}, \mathsf{x}.\ \mathsf{n}\ \mathsf{f}\ (\mathsf{f}\ \mathsf{n}\ \mathsf{x}) \end{aligned}$$

These nats are redundant;

Self-recursing numerals: motivation

Church numerals have simple type $(a \to a) \to a \to a$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $nat \to (nat \to a \to a) \to a \to a$.
For self recursion, redefine $nat = \mu\ n.\ (n \to a \to a) \to a \to a$:

$$\begin{aligned} \text{zero} &:= \lambda\_, x.\ x. \\ \text{succ} &:= \lambda n, f, x.\ n\ f\ (f\ n\ x) \end{aligned}$$

These nats are redundant; exponentially large normal forms:

Self-recursing numerals: motivation

Church numerals have simple type $(a \rightarrow a) \rightarrow a \rightarrow a$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $\text{nat} \rightarrow (\text{nat} \rightarrow a \rightarrow a) \rightarrow a \rightarrow a$.
For self recursion, redefine $\text{nat} = \mu\ n.\ (n \rightarrow a \rightarrow a) \rightarrow a \rightarrow a$:

$$\begin{aligned} \text{zero} &:= \lambda\_, x.\ x. \\ \text{succ} &:= \lambda n, f, x.\ n\ f\ (f\ n\ x) \end{aligned}$$

These nats are redundant; exponentially large normal forms:

$$2 = \lambda f, x.\ f\ (\lambda f, x. f(\lambda f, x. x) x)\ (f(\lambda f, x. x) x)$$

Self-recursing numerals: motivation

Church numerals have simple type $(a \to a) \to a \to a$.
but predecessor has problems:

- on well-defined terms, it is linear-time.
- on partially-defined terms it diverges.

Gödel's recursor has type $nat \to (nat \to a \to a) \to a \to a$.
For self recursion, redefine $nat = \mu\ n.\ (n \to a \to a) \to a \to a$:

$$\begin{aligned} \text{zero} &:= \lambda\_, x.\ x. \\ \text{succ} &:= \lambda n, f, x.\ n\ f\ (f\ n\ x) \end{aligned}$$

These nats are redundant; exponentially large normal forms:

$$2 = \lambda f, x.\ f\ (\lambda f, x. f(\lambda f, x. x) x)\ (f(\lambda f, x. x) x)$$

Self-recursing numerals: correctness

$$
\begin{array}{l}
\mathsf{nat} \;:=\; \mathsf{type}\ (\\
\quad \mathbf{Y}\ \lambda \mathsf{a}.\ (\mathsf{Simple}\ \lambda \mathsf{b}, \mathsf{b}'.\ (\mathsf{a} \to \mathsf{b}' \to \mathsf{b}) \to \mathsf{b} \to \mathsf{b}') \\
\qquad \mid\ \langle \lambda \mathsf{n}:\mathsf{a}, \_, \mathsf{f}:\mathsf{a} \to \mathsf{a}, \mathsf{x}:\mathsf{a}.\ \mathsf{f}\ \mathsf{n}(\mathsf{n}\ \mathsf{f}\ \mathsf{x}), \quad \lambda \_, \mathsf{x}:\mathsf{a}.\ \mathsf{x} \rangle \\
\qquad \mid\ \langle \lambda \_, \mathsf{n}:\mathsf{a}, \mathsf{f}:\mathsf{a} \to \mathsf{a}, \mathsf{x}:\mathsf{a}.\ \mathsf{f}\ \mathsf{n}(\mathsf{n}\ \mathsf{f}\ \mathsf{x}), \quad \lambda \_, \mathsf{x}:\mathsf{a}.\ \mathsf{x} \rangle \\
\qquad \mid\ \lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot) \\
).
\end{array}
$$

Note the two different ways of descending: left and right.

Self-recursing numerals: correctness

$$
\begin{aligned}
\mathsf{nat} \;:=\; & \mathsf{type}\ (\\
& \quad \mathbf{Y}\ \lambda a.\ (\mathsf{Simple}\ \lambda b, b'.\ (a \to b' \to b) \to b \to b') \\
& \qquad \mid\ \langle \lambda n{:}a, \_, f{:}a \to a, x{:}a.\ f\ n(n\ f\ x),\quad \lambda \_, x{:}a.\ x\rangle \\
& \qquad \mid\ \langle \lambda \_, n{:}a, f{:}a \to a, x{:}a.\ f\ n(n\ f\ x),\quad \lambda \_, x{:}a.\ x\rangle \\
& \qquad \mid\ \lambda q.\ q\ \bot\ (q\ \top\ \bot) \\
&).
\end{aligned}
$$

Note the two different ways of descending: left and right.

Theorem

$\mathsf{inhab}(\mathsf{nat}) = \{\top\}\ \cup\ \{\mathsf{succ}^n\ z \mid n \in \mathbb{N},\ z \in \{\bot, \mathsf{zero}\}\}.$

Self-recursing numerals: correctness

$$
\begin{array}{l}
\mathsf{nat} \;:=\; \mathsf{type}\ (\\
\quad \mathbf{Y}\ \lambda \mathsf{a}.\ (\mathsf{Simple}\ \lambda \mathsf{b}, \mathsf{b}'.\ (\mathsf{a} \to \mathsf{b}' \to \mathsf{b}) \to \mathsf{b} \to \mathsf{b}') \\
\qquad \mid\ \langle \lambda \mathsf{n}{:}\mathsf{a}, \_, \mathsf{f}{:}\mathsf{a} \to \mathsf{a}, \mathsf{x}{:}\mathsf{a}.\ \mathsf{f}\ \mathsf{n}(\mathsf{n}\ \mathsf{f}\ \mathsf{x}), \quad \lambda \_, \mathsf{x}{:}\mathsf{a}.\ \mathsf{x} \rangle \\
\qquad \mid\ \langle \lambda \_, \mathsf{n}{:}\mathsf{a}, \mathsf{f}{:}\mathsf{a} \to \mathsf{a}, \mathsf{x}{:}\mathsf{a}.\ \mathsf{f}\ \mathsf{n}(\mathsf{n}\ \mathsf{f}\ \mathsf{x}), \quad \lambda \_, \mathsf{x}{:}\mathsf{a}.\ \mathsf{x} \rangle \\
\qquad \mid\ \lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot) \\
).
\end{array}
$$

Note the two different ways of descending: left and right.

Theorem

$\mathsf{inhab}(\mathsf{nat}) = \{\top\} \ \cup\ \{\mathsf{succ}^{\mathsf{n}}\ \mathsf{z} \ \mid\ \mathsf{n} \in \mathbb{N},\ \mathsf{z} \in \{\bot, \mathsf{zero}\}\}.$

Proof.

As above, only we need to ensure consistency across BT.

Self-recursing numerals: correctness

$$
\begin{aligned}
\mathsf{nat} \;:=\; & \mathsf{type}\ (\\
& \quad \mathbf{Y}\ \lambda a.\ (\mathsf{Simple}\ \lambda b, b'.\ (a \to b' \to b) \to b \to b') \\
& \qquad \mid\ \langle \lambda n{:}a, \_, f{:}a \to a, x{:}a.\ f\ n(n\ f\ x),\quad \lambda \_, x{:}a.\ x \rangle \\
& \qquad \mid\ \langle \lambda \_, n{:}a, f{:}a \to a, x{:}a.\ f\ n(n\ f\ x),\quad \lambda \_, x{:}a.\ x \rangle \\
& \qquad \mid\ \lambda q.\ q\ \bot\ (q\ \top\ \bot) \\
&).
\end{aligned}
$$

Note the two different ways of descending: left and right.

Theorem

$\mathsf{inhab}(\mathsf{nat}) = \{\top\} \;\cup\; \{\mathsf{succ}^n\ z \mid n \in \mathbb{N},\ z \in \{\bot, \mathsf{zero}\}\}.$

Proof.

As above, only we need to ensure consistency across BT.
At root, descend in either direction.

Self-recursing numerals: correctness

$$
\begin{array}{l}
\mathsf{nat} \;:=\; \mathsf{type}\ (\\
\quad \mathbf{Y}\ \lambda a.\ (\mathsf{Simple}\ \lambda b, b'.\ (a \to b' \to b) \to b \to b')\\
\qquad \mid\ \langle \lambda n{:}a, \_, f{:}a \to a, x{:}a.\ f\ n(n\ f\ x),\quad \lambda \_, x{:}a.\ x \rangle\\
\qquad \mid\ \langle \lambda \_, n{:}a, f{:}a \to a, x{:}a.\ f\ n(n\ f\ x),\quad \lambda \_, x{:}a.\ x \rangle\\
\qquad \mid\ \lambda q.\ q\ \bot\ (q\ \top\ \bot)\\
).
\end{array}
$$

Note the two different ways of descending: left and right.

Theorem

$\mathsf{inhab}(\mathsf{nat}) = \{\top\} \cup \{\mathsf{succ}^n\ z \mid n \in \mathbb{N},\ z \in \{\bot, \mathsf{zero}\}\}$.

Proof.

As above, only we need to ensure consistency across BT.
At root, descend in either direction.
The a in $(a \to b' \to b) \to b \to b'$ descends below root. □

Quotient types

What is an r.e. set (of x : a's)?

Quotient types

What is an r.e. set (of $x : a$'s)?
$\longrightarrow$ A sequence : $nat \longrightarrow a$?

Quotient types

What is an r.e. set (of $x : a$'s)?
$\longrightarrow$ A sequence $: nat \longrightarrow a$? ...but order doesn't matter

Quotient types

What is an r.e. set (of $x : a$'s)?
$\longrightarrow$ A sequence : $nat \longrightarrow a$? ...but order doesn't matter
$\longrightarrow$ A semipredicate : $a \longrightarrow semi$?

Quotient types

What is an r.e. set (of $x : a$'s)?
$\longrightarrow$ A sequence : nat $\longrightarrow$ a? ...but order doesn't matter
$\longrightarrow$ A semipredicate : a $\longrightarrow$ semi? ...but no mapping

Quotient types

What is an r.e. set (of $x : a$'s)?
$\rightarrow$ A sequence : $nat \rightarrow a$? ...but order doesn't matter
$\rightarrow$ A semipredicate : $a \rightarrow semi$? ...but no mapping
$\rightarrow$ A semiset : $(a \rightarrow b) \rightarrow b$?

Quotient types

What is an r.e. set (of $x : a$'s)?
$\longrightarrow$ A sequence : nat $\longrightarrow$ a? ...but order doesn't matter
$\longrightarrow$ A semipredicate : a $\longrightarrow$ semi? ...but no mapping
$\longrightarrow$ A semiset : (a $\longrightarrow$ b) $\longrightarrow$ b? ...works in concurrent CA.

Quotient types

What is an r.e. set (of $x : a$'s)?
$\rightarrow$ A sequence : $nat \rightarrow a$? ...but order doesn't matter
$\rightarrow$ A semipredicate : $a \rightarrow semi$? ...but no mapping
$\rightarrow$ A semiset : $(a \rightarrow b) \rightarrow b$? ...works in concurrent CA.

$$\text{Semiset} := \text{type} \rightarrow \text{type}\ (\lambda a.\ \text{Simple}\ \lambda b, b'.\ (a \rightarrow b) \rightarrow b').$$

Now we can define quotient types.
Let $M : \text{Semiset}(a \rightarrow a)$ generate a monoid action on a.

Quotient types

What is an r.e. set (of $x : a$'s)?
$\rightarrow$ A sequence : $nat \rightarrow a$? ...but order doesn't matter
$\rightarrow$ A semipredicate : $a \rightarrow semi$? ...but no mapping
$\rightarrow$ A semiset : $(a \rightarrow b) \rightarrow b$? ...works in concurrent CA.

$$\text{Semiset} \;:=\; \text{type} \rightarrow \text{type}\ (\lambda a.\ \text{Simple}\ \lambda b, b'.\ (a \rightarrow b) \rightarrow b').$$

Now we can define quotient types.
Let $M : \text{Semiset}(a \rightarrow a)$ generate a monoid action on a.
The quotient type of M-orbits is Mod M,

Quotient types

What is an r.e. set (of $x : a$'s)?
$\rightarrow$ A sequence : $\text{nat} \rightarrow a$? ...but order doesn't matter
$\rightarrow$ A semipredicate : $a \rightarrow \text{semi}$? ...but no mapping
$\rightarrow$ A semiset : $(a \rightarrow b) \rightarrow b$? ...works in concurrent CA.

$$\text{Semiset} \;:=\; \text{type} \rightarrow \text{type}\ (\lambda a.\ \text{Simple}\ \lambda b, b'.\ (a \rightarrow b) \rightarrow b').$$

Now we can define quotient types.
Let $M : \text{Semiset}(a \rightarrow a)$ generate a monoid action on a.
The quotient type of M-orbits is Mod M, where

$$\begin{aligned} \text{Mod} \;:=\; & (\forall a : \text{close}.\ (\text{Semiset}\ a \rightarrow a) \;\rightarrow\; (\text{Sub}\ (\text{Semiset}\ a)))\ (\\ & \quad \lambda a.\ \ \lambda M.M \lambda m.\ \ \lambda X.X \lambda x.\ \ \langle m\ x \rangle \\ &). \end{aligned}$$

Summary and Questions

- Concurrent CA is inadvertantly typed

Summary and Questions

- Concurrent CA is inadvertantly typed
  (sequential CA is not).

Summary and Questions

- Concurrent CA is inadvertantly typed
  (sequential CA is not).

- $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable types required head normal forms:

Summary and Questions

- Concurrent CA is inadvertantly typed
  (sequential CA is not).
- $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable types required head normal forms:
  $\mathcal{P}\omega$ fails,

Summary and Questions

- Concurrent CA is inadvertantly typed
  (sequential CA is not).
- $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable types required head normal forms:
  $\mathcal{P}\omega$ fails, $\mathcal{D}_\infty$ fails,

Summary and Questions

- Concurrent CA is inadvertently typed
  (sequential CA is not).
- $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable types required head normal forms:
  $\mathcal{P}\omega$ fails, $\mathcal{D}_\infty$ fails, completed term model works.

Summary and Questions

- Concurrent CA is inadvertantly typed
  (sequential CA is not).
- $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable types required head normal forms:
  $\mathcal{P}\omega$ fails, $\mathcal{D}_\infty$ fails, completed term model works.
- Very rich type structure.

Summary and Questions

- Concurrent CA is inadvertantly typed
  (sequential CA is not).
- $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable types required head normal forms:
  $\mathcal{P}\omega$ fails, $\mathcal{D}_\infty$ fails, completed term model works.
- Very rich type structure.

Questions.

- Exactly which types are definable?

Summary and Questions

- Concurrent CA is inadvertantly typed
  (sequential CA is not).
- $\mathbf{S}, \mathbf{K}, \mathbf{J}$-definable types required head normal forms:
  $\mathcal{P}\omega$ fails, $\mathcal{D}_\infty$ fails, completed term model works.
- Very rich type structure.

Questions.

- Exactly which types are definable?
- Are sequential simple types uniformly definable?

Definition of raising and lowering operators

Define raising and lowering operators

$$\begin{aligned} \text{raise} &:= (\lambda x, \_.x) &= \mathbf{K}. \\ \text{lower} &:= (\lambda x.x \ \top) &= \langle \top \rangle. \end{aligned}$$

so that

$$\begin{aligned} &\text{lower} \circ \text{raise} = \mathbf{I}, \\ &\text{raise} \circ \text{lower} = \lambda x, \_. \ x \ \top \quad \sqsupseteq \quad \mathbf{I} \end{aligned}$$

Definition of raising and lowering operators

Define raising and lowering operators

$$\begin{aligned} \text{raise} &:= (\lambda x, \_.x) = \mathbf{K}. \\ \text{lower} &:= (\lambda x.x\ \top) = \langle \top \rangle. \end{aligned}$$

so that

$$\begin{aligned} \text{lower} \circ \text{raise} &= \mathbf{I}, \\ \text{raise} \circ \text{lower} &= \lambda x, \_.\ x\ \top \quad \sqsupseteq \quad \mathbf{I} \end{aligned}$$

Similarly at function type,

$$\begin{aligned} \mathbf{I} \to \text{raise} &= \lambda f, x, \_.f\ x \\ \mathbf{I} \to \text{lower} &= \lambda f, x.\ f\ x\ \top \end{aligned}$$

so that

$$\begin{aligned} (\mathbf{I} \to \text{lower}) \circ (\mathbf{I} \to \text{raise}) &= \mathbf{I}, \\ (\mathbf{I} \to \text{raise}) \circ (\mathbf{I} \to \text{lower}) &= \lambda f, x, \_.\ f\ x\ \top \quad \sqsupseteq \quad \mathbf{I} \end{aligned}$$

Constructing simple concurrent types

Now these operators generate our previous $\mathsf{L}_{mn}, \mathsf{R}_{mn}$:

$$\mathsf{R}_{mn} = (\mathsf{m\ raise})\circ(\mathsf{n}\ \mathbf{I}\rightarrow\mathsf{raise})$$
$$\mathsf{L}_{mn} = (\mathsf{n}\ \mathbf{I}\rightarrow\mathsf{lower})\circ(\mathsf{m\ lower})$$

Hence we have a simple definition of semi

$$\begin{array}{l} \mathsf{Simple} \ := \ \mathsf{any}\rightarrow\mathsf{type}\ (\\ \quad \lambda \mathsf{f}.\ \mathsf{curry}\rightarrow\mathsf{uncurry} \\ \qquad |\ \mathsf{f}\ \mathbf{I}\ \mathbf{I} \\ \qquad |\ \mathsf{f\ raise\ lower} \\ \qquad |\ \mathsf{f}\ \mathbf{I}\rightarrow\mathsf{raise}\ \mathbf{I}\rightarrow\mathsf{lower} \\ ). \end{array}$$

Applications to typechecking

Now the boolean type becomes

$$
\begin{array}{lll}
\mathsf{bool} & := & \mathsf{type}\ (\mathsf{curry} \rightarrow \mathsf{curry} \rightarrow \mathsf{uncurry} \\
& \mid & \mathsf{raise} \rightarrow \mathsf{raise} \rightarrow \mathsf{lower} \\
& \mid & (\mathbf{I} \rightarrow \mathsf{raise}) \rightarrow (\mathbf{I} \rightarrow \mathsf{raise}) \rightarrow (\mathbf{I} \rightarrow \mathsf{lower}) \\
& \mid & \langle \mathbf{K}, \mathbf{F} \rangle \\
& \mid & (\lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot)) \\
). & &
\end{array}
$$

We can now reduce typechecking $\mathsf{x} : \mathsf{bool}$ to five checks,

Applications to typechecking

Now the boolean type becomes

$$
\begin{array}{rcl}
\mathsf{bool} & := & \mathsf{type}\ (\mathsf{curry} \to \mathsf{curry} \to \mathsf{uncurry} \\
 & | & \mathsf{raise} \to \mathsf{raise} \to \mathsf{lower} \\
 & | & (\mathbf{I} \to \mathsf{raise}) \to (\mathbf{I} \to \mathsf{raise}) \to (\mathbf{I} \to \mathsf{lower}) \\
 & | & \langle \mathbf{K}, \mathbf{F} \rangle \\
 & | & (\lambda \mathsf{q}.\ \mathsf{q}\ \bot\ (\mathsf{q}\ \top\ \bot)) \\
 &). &
\end{array}
$$

We can now reduce typechecking $\mathsf{x} : \mathsf{bool}$ to five checks, which may succeed even under β-η conversion!

back

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\;((\mathsf{Simple}\;\lambda \mathsf{a}, \mathsf{a}'.\; \mathsf{a} \to \mathsf{a}') \;\mid\; \langle \mathbf{I} \rangle).$$

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}') \;\mid\; \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot$: semi by β-reduction.

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda \mathsf{a}, \mathsf{a}'.\ \mathsf{a} \to \mathsf{a}') \;\mid\; \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \mathsf{semi}$ by β-reduction. Any other $\mathsf{q} : \mathsf{semi}$ converges,

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot : \mathsf{semi}$ by β-reduction. Any other $q : \mathsf{semi}$ converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot$: semi by β-reduction. Any other q : semi converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$

Correctness of semi: overview

$$\text{semi} \;:=\; \text{type } ((\text{Simple } \lambda a, a'.\ a \to a') \;\mid\; \langle \mathbf{I} \rangle).$$

Theorem

$\text{inhab}(\text{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot$: semi by β-reduction. Any other q : semi converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$ or

$$z = f \qquad \textit{(head is in the right place)},$$

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot$: semi by β-reduction. Any other q : semi converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$ or

$z = f$ *(head is in the right place),*

$m = n$ *(right number of limbs), and*

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a')\ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot$: semi by β-reduction. Any other q : semi converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$ or

$z = f$	*(head is in the right place),*
$m = n$	*(right number of limbs), and*
$M_i \sqsubseteq x_i$	*(each limb is healthy).*

Correctness of semi: overview

$$\mathsf{semi} \;:=\; \mathsf{type}\ ((\mathsf{Simple}\ \lambda a, a'.\ a \to a') \ \mid\ \langle \mathbf{I} \rangle).$$

Theorem

$\mathsf{inhab}(\mathsf{semi}) = \{\bot, \mathbf{I}, \top\}$.

Proof.

$\bot$: semi by β-reduction. Any other q : semi converges, say

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

Show that either $q = \top$ or

$z = f$	*(head is in the right place),*
$m = n$	*(right number of limbs), and*
$M_i \sqsubseteq x_i$	*(each limb is healthy).*

Finally raise a healthy $q' = \lambda f, \underline{x}.f\ \underline{M}$ up to $\mathbf{I}$.

□

Correctness of semi: head is in the right place

We know q : semi and

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_n.\ \mathsf{z}\ \mathsf{M}_1\ \ \ldots\ \ \mathsf{M}_m$$

Correctness of semi: head is in the right place

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

If $z \neq f$ then $z = x_i$ for some i.

Correctness of semi: head is in the right place

We know $q : \mathrm{semi}$ and

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ \ldots\ \ M_m$$

If $z \neq f$ then $z = x_i$ for some i.

Cover all the x_j's with a minefield (n,0):

$$R \rightarrow L\ q'\ f$$

Correctness of semi: head is in the right place

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

If $z \neq f$ then $z = x_i$ for some i.

Cover all the x_j's with a minefield (n,0):

$$R \rightarrow L\ q'\ f = q\ (n\ \mathbf{K}\ f)\ \top^{\sim n}$$

Correctness of semi: head is in the right place

We know $q : \mathsf{semi}$ and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

If $z \neq f$ then $z = x_i$ for some i.

Cover all the x_j's with a minefield (n,0):

$$\begin{aligned} R \to L\ q'\ f &= q\ (n\ \mathbf{K}\ f)\ \top^{\sim n} \\ &= (\lambda \underline{x}.\ x_i\ \underline{M})\ \top^{\sim n} \end{aligned}$$

Correctness of semi: head is in the right place

We know q : semi and

$$q \ \sqsupseteq \ q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1 \ \ldots \ M_m$$

If $z \neq f$ then $z = x_i$ for some i.

Cover all the x_j's with a minefield (n,0):

$$\begin{aligned} R \to L\ q'\ f &= \ q\ (n\ \mathbf{K}\ f)\ \top^{\sim n} \\ &= \ (\lambda \underline{x}.\ x_i\ \underline{M})\ \top^{\sim n} \\ &= \ \top\ \underline{M} \end{aligned}$$

Correctness of semi: head is in the right place

We know $\mathsf{q} : \mathsf{semi}$ and

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_n.\ \mathsf{z}\ \mathsf{M}_1\ \ldots\ \mathsf{M}_m$$

If $\mathsf{z} \neq \mathsf{f}$ then $\mathsf{z} = \mathsf{x}_i$ for some i.

Cover all the x_j's with a minefield (n,0):

$$\begin{aligned} \mathsf{R}\!\to\!\mathsf{L}\ \mathsf{q}'\ \mathsf{f} &= \mathsf{q}\ (\mathsf{n}\ \mathbf{K}\ \mathsf{f})\ \top^{\sim \mathsf{n}} \\ &= (\lambda \underline{\mathsf{x}}.\ \mathsf{x}_i\ \underline{\mathsf{M}})\ \top^{\sim \mathsf{n}} \\ &= \top\ \underline{\mathsf{M}} \quad = \top \end{aligned}$$

Correctness of semi: head is in the right place

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

If $z \neq f$ then $z = x_i$ for some i.

Cover all the x_j's with a minefield (n,0):

$$\begin{aligned} R \rightarrow L\ q'\ f = &\ q\ (n\ \mathbf{K}\ f)\ \top^{\sim n} \\ = &\ (\lambda \underline{x}.\ x_i\ \underline{M})\ \top^{\sim n} \\ = &\ \top\ \underline{M} \quad = \ \top \end{aligned}$$

So $q = \top$.

Correctness of semi: head is in the right place

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ z\ M_1\ \ldots\ M_m$$

If $z \neq f$ then $z = x_i$ for some i.

Cover all the x_j's with a minefield (n,0):

$$\begin{aligned} R \rightarrow L\ q'\ f = &\ q\ (n\ \mathbf{K}\ f)\ \top^{\sim n} \\ = &\ (\lambda \underline{x}.\ x_i\ \underline{M})\ \top^{\sim n} \\ = &\ \top\ \underline{M} \quad = \ \top \end{aligned}$$

So $q = \top$.
otherwise...

Correctness of semi: right number of limbs

We know $\mathsf{q}:\mathsf{semi}$ and

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}.\ \mathsf{f}\ \mathsf{M}_1\ \ldots\ \mathsf{M}_\mathsf{m}$$

Correctness of semi: right number of limbs

We know q : semi and

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}.\ \mathsf{f}\ \mathsf{M}_1\ \ldots\ \mathsf{M}_\mathsf{m}$$

Make q' navigate a big minefield,

Correctness of semi: right number of limbs

We know q : semi and

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_n.\ \mathsf{f}\ \mathsf{M}_1\ \ldots\ \mathsf{M}_m$$

Make q′ navigate a big minefield, say $(n + m, n + m)$

$$\mathsf{R}{\rightarrow}\mathsf{L}\ \mathsf{q}'\ \mathsf{f}\ \mathsf{x} = \ \mathsf{q}'\ (\lambda \underline{\mathsf{u}}, \mathsf{v}, \underline{\mathsf{w}}.\ \mathsf{f}\ \mathsf{v})\ \mathsf{T}^{\sim m+n}\ \mathsf{x}\ \mathsf{T}^{\sim m+n}$$

Correctness of semi: right number of limbs

We know $\mathsf{q} : \mathsf{semi}$ and

$$\mathsf{q} \sqsupseteq \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}.\ \mathsf{f}\ \mathsf{M}_1\ \ldots\ \mathsf{M}_\mathsf{m}$$

Make q' navigate a big minefield, say $(\mathsf{n}+\mathsf{m}, \mathsf{n}+\mathsf{m})$

$$\begin{aligned} \mathsf{R} \rightarrow \mathsf{L}\ \mathsf{q}'\ \mathsf{f}\ \mathsf{x} &= \mathsf{q}'\ (\lambda \underline{\mathsf{u}}, \mathsf{v}, \underline{\mathsf{w}}.\ \mathsf{f}\ \mathsf{v})\ \mathsf{T}^{\sim \mathsf{m}+\mathsf{n}}\ \mathsf{x}\ \mathsf{T}^{\sim \mathsf{m}+\mathsf{n}} \\ &= (\lambda \underline{\mathsf{x}}, \underline{\mathsf{u}}, \mathsf{v}, \underline{\mathsf{w}}.\ \mathsf{f}\ \mathsf{v})\ \underline{\mathsf{M}}\ \top^{\sim \mathsf{m}+\mathsf{n}}\ \mathsf{x}\ \top^{\sim \mathsf{m}+\mathsf{n}} \end{aligned}$$

Correctness of semi: right number of limbs

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_m$$

Make q' navigate a big minefield, say $(n+m, n+m)$

$$\begin{aligned} R \to L\ q'\ f\ x &= q'\ (\lambda \underline{u}, v, \underline{w}.\ f\ v)\ T^{\sim m+n}\ x\ T^{\sim m+n} \\ &= (\lambda \underline{x}, \underline{u}, v, \underline{w}.\ f\ v)\ \underline{M}\ \top^{\sim m+n}\ x\ \top^{\sim m+n} \end{aligned}$$

How far off can q' be?

Correctness of semi: right number of limbs

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_m$$

Make q' navigate a big minefield, say $(n + m, n + m)$

$$\begin{aligned} R \to L\ q'\ f\ x = &\ q'\ (\lambda \underline{u}, v, \underline{w}.\ f\ v)\ T^{\sim m+n}\ x\ T^{\sim m+n} \\ = &\ (\lambda \underline{x}, \underline{u}, v, \underline{w}.\ f\ v)\ \underline{M}\ \top^{\sim m+n}\ x\ \top^{\sim m+n} \end{aligned}$$

How far off can q' be?

$$|\ \underline{x}, \underline{u}\ | = 2n + m$$

Correctness of semi: right number of limbs

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_m$$

Make q' navigate a big minefield, say $(n+m, n+m)$

$$\begin{aligned} R \to L\ q'\ f\ x &= q'\ (\lambda \underline{u}, v, \underline{w}.\ f\ v)\ T^{\sim m+n}\ x\ T^{\sim m+n} \\ &= (\lambda \underline{x}, \underline{u}, v, \underline{w}.\ f\ v)\ \underline{M}\ \top^{\sim m+n}\ x\ \top^{\sim m+n} \end{aligned}$$

How far off can q' be?

$$|\ \underline{x}, \underline{u}\ | = 2n + m \quad \overset{?}{=} \quad n + 2m = |\ \underline{M}\ \top^{\sim m+n}\ |$$

Correctness of semi: right number of limbs

We know $q : \mathrm{semi}$ and

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_m$$

Make q' navigate a big minefield, say $(n+m, n+m)$

$$\begin{aligned} R{\rightarrow}L\ q'\ f\ x = &\ q'\ (\lambda \underline{u}, v, \underline{w}.\ f\ v)\ T^{\sim m+n}\ x\ T^{\sim m+n} \\ = &\ (\lambda \underline{x}, \underline{u}, v, \underline{w}.\ f\ v)\ \underline{M}\ \top^{\sim m+n}\ x\ \top^{\sim m+n} \end{aligned}$$

How far off can q' be?

$$|\ \underline{x}, \underline{u}\ | = 2n + m \quad \stackrel{?}{=} \quad n + 2m = |\ \underline{M}\ \top^{\sim m+n}\ |$$

If $n \neq m$ then $\mathrm{semi}\ q' = \top$.

Correctness of semi: right number of limbs

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_m$$

Make q' navigate a big minefield, say $(n + m, n + m)$

$$\begin{aligned} R \rightarrow L\ q'\ f\ x &= q'\ (\lambda \underline{u}, v, \underline{w}.\ f\ v)\ T^{\sim m+n}\ x\ T^{\sim m+n} \\ &= (\lambda \underline{x}, \underline{u}, v, \underline{w}.\ f\ v)\ \underline{M}\ \top^{\sim m+n}\ x\ \top^{\sim m+n} \end{aligned}$$

How far off can q' be?

$$|\ \underline{x}, \underline{u}\ | = 2n + m \quad \stackrel{?}{=} \quad n + 2m = |\ \underline{M}\ \top^{\sim m+n}\ |$$

If $n \neq m$ then semi $q' = \top$.
otherwise...

Correctness of semi: each limb is healthy

We know $\mathsf{q}:\mathsf{semi}$ and

$$\mathsf{q} \;\sqsupseteq\; \mathsf{q}' = \lambda \mathsf{f}, \mathsf{x}_1, \ldots, \mathsf{x}_n.\ \mathsf{f}\ \mathsf{M}_1\ \ \ldots\ \ \mathsf{M}_n$$

Correctness of semi: each limb is healthy

We know q : semi and

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\; f\; M_1\; \ldots\; M_n$$

If $M_i \not\sqsubseteq x_i$ then $M_i \sqsupseteq N \not\sqsubseteq x_i$ for some h.n.f N.

Correctness of semi: each limb is healthy

We know q : semi and

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
or an offending head variable z.

Correctness of semi: each limb is healthy

We know q : semi and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
or an offending head variable z.
If $z \notin \{f, \underline{x}\}$, descend with $\langle \mathbf{I} \rangle$ until it is.

Correctness of semi: each limb is healthy

We know $q : \mathrm{semi}$ and

$$q \sqsupseteq q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ldots\ M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
or an offending head variable z.
If $z \notin \{f, \underline{x}\}$, descend with $\langle \mathbf{I} \rangle$ until it is.
If $z = x_i$, make q navigate a minefield;

Correctness of semi: each limb is healthy

We know q : semi and

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\; f\; M_1\; \ldots\; M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
or an offending head variable z.
If $z \notin \{f, \underline{x}\}$, descend with $\langle \mathbf{I} \rangle$ until it is.
If $z = x_i$, make q navigate a minefield; then descend.

Correctness of semi: each limb is healthy

We know $q : \mathsf{semi}$ and

$$q \ \sqsupseteq \ q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1 \ \ldots \ M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
or an offending head variable z.
If $z \notin \{f, \underline{x}\}$, descend with $\langle \mathbf{I} \rangle$ until it is.
If $z = x_i$, make q navigate a minefield; then descend.
If $z = f$, make f inert with curry;

Correctness of semi: each limb is healthy

We know q : semi and

$$q \ \sqsupseteq \ q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ \ldots\ \ M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
or an offending head variable z.
If $z \notin \{f, \underline{x}\}$, descend with $\langle \mathbf{I} \rangle$ until it is.
If $z = x_i$, make q navigate a minefield; then descend.
If $z = f$, make f inert with curry; then descend.

Correctness of semi: each limb is healthy

We know q : semi and

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ \ldots\ \ M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
or an offending head variable z.
If $z \notin \{f, \underline{x}\}$, descend with $\langle \mathbf{I} \rangle$ until it is.
If $z = x_i$, make q navigate a minefield; then descend.
If $z = f$, make f inert with curry; then descend.

Eventually we hit a $\top$.

Correctness of semi: each limb is healthy

We know $q : \mathsf{semi}$ and

$$q \;\sqsupseteq\; q' = \lambda f, x_1, \ldots, x_n.\ f\ M_1\ \ \ldots\ \ M_n$$

If $M_i \not\sqsupseteq x_i$ then $M_i \sqsupseteq N \not\sqsupseteq x_i$ for some h.n.f N.
Somewhere down the BT of q' is either a $\top$,
 or an offending head variable z.
If $z \notin \{f, \underline{x}\}$, descend with $\langle \mathbf{I} \rangle$ until it is.
If $z = x_i$, make q navigate a minefield; then descend.
If $z = f$, make f inert with curry; then descend.

Eventually we hit a $\top$.
otherwise...

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda \mathsf{f}, \underline{\mathsf{x}}.\mathsf{f}\ \underline{\mathsf{x}} \quad \sqsupseteq \quad \mathsf{q}' \quad \sqsupseteq \quad \lambda \mathsf{f}, \underline{\mathsf{x}}.\ \mathsf{f}\ \bot^{\sim \mathsf{n}}$$

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda f, \underline{x}.f\ \underline{x} \quad \sqsupseteq \quad q' \quad \sqsupseteq \quad \lambda f, \underline{x}.\ f\ \bot^{\sim n}$$

Raise and lower n times to ignore faulty args

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda \mathsf{f}, \underline{\mathsf{x}}.\mathsf{f}\ \underline{\mathsf{x}} \quad \sqsupseteq \quad \mathsf{q}' \quad \sqsupseteq \quad \lambda \mathsf{f}, \underline{\mathsf{x}}.\ \mathsf{f}\ \bot^{\sim \mathsf{n}}$$

Raise and lower n times to ignore faulty args

$$\mathsf{n}\ \mathbf{K}{\rightarrow}\langle\top\rangle\ \mathsf{q}'\ \mathsf{f}$$

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda \mathsf{f}, \underline{\mathsf{x}}.\mathsf{f}\ \underline{\mathsf{x}} \quad \sqsupseteq \quad \mathsf{q}' \quad \sqsupseteq \quad \lambda \mathsf{f}, \underline{\mathsf{x}}.\ \mathsf{f}\ \bot^{\sim \mathsf{n}}$$

Raise and lower n times to ignore faulty args

$$\begin{aligned} &\mathsf{n}\ \mathbf{K} \rightarrow \langle \top \rangle\ \mathsf{q}'\ \mathsf{f} \\ &\quad = \ \mathsf{q}'\ (\mathsf{n}\ \mathbf{K}\ \mathsf{f})\ \top^{\sim \mathsf{n}} \end{aligned}$$

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda f, \underline{x}.f\ \underline{x} \quad \sqsupseteq \quad q' \quad \sqsupseteq \quad \lambda f, \underline{x}.\ f\ \bot^{\sim n}$$

Raise and lower n times to ignore faulty args

$$\begin{aligned} & n\ \mathbf{K} \rightarrow \langle \top \rangle\ q'\ f \\ & \quad = \ q'\ (n\ \mathbf{K}\ f)\ \top^{\sim n} \\ & \quad = \ (\lambda \underline{x}.\ n\ \mathbf{K}\ f\ \underline{M})\ \top^{\sim n} \end{aligned}$$

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda \mathsf{f}, \underline{\mathsf{x}}.\mathsf{f}\ \underline{\mathsf{x}} \quad \sqsupseteq \quad \mathsf{q}' \quad \sqsupseteq \quad \lambda \mathsf{f}, \underline{\mathsf{x}}.\ \mathsf{f}\ \bot^{\sim \mathsf{n}}$$

Raise and lower n times to ignore faulty args

$$\begin{aligned} &\mathsf{n}\ \mathbf{K} \!\rightarrow\! \langle \top \rangle\ \mathsf{q}'\ \mathsf{f} \\ &\quad = \ \mathsf{q}'\ (\mathsf{n}\ \mathbf{K}\ \mathsf{f})\ \top^{\sim \mathsf{n}} \\ &\quad = \ (\lambda \underline{\mathsf{x}}.\ \mathsf{n}\ \mathbf{K}\ \mathsf{f}\ \underline{\mathsf{M}})\ \top^{\sim \mathsf{n}} \\ &\quad = \ (\lambda \underline{\mathsf{x}}.\ \mathsf{f})\ \top^{\sim \mathsf{n}} \end{aligned}$$

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q′ is healthy, but not at full strength

$$\lambda f, \underline{x}. f\ \underline{x} \quad \sqsupseteq \quad q' \quad \sqsupseteq \quad \lambda f, \underline{x}.\ f\ \bot^{\sim n}$$

Raise and lower n times to ignore faulty args

$$\begin{aligned} & n\ \mathbf{K}{\rightarrow}\langle\top\rangle\ q'\ f \\ & \quad = \ q'\ (n\ \mathbf{K}\ f)\ \top^{\sim n} \\ & \quad = \ (\lambda\underline{x}.\ n\ \mathbf{K}\ f\ \underline{M})\ \top^{\sim n} \\ & \quad = \ (\lambda\underline{x}.\ f)\ \top^{\sim n} \\ & \quad = \ f \end{aligned}$$

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda f, \underline{x}.f\ \underline{x} \quad \sqsupseteq \quad q' \quad \sqsupseteq \quad \lambda f, \underline{x}.\ f\ \bot^{\sim n}$$

Raise and lower n times to ignore faulty args

$$\begin{aligned} & n\ \mathbf{K} \rightarrow \langle \top \rangle\ q'\ f \\ & \quad = \ q'\ (n\ \mathbf{K}\ f)\ \top^{\sim n} \\ & \quad = \ (\lambda \underline{x}.\ n\ \mathbf{K}\ f\ \underline{M})\ \top^{\sim n} \\ & \quad = \ (\lambda \underline{x}.\ f)\ \top^{\sim n} \\ & \quad = \ f \end{aligned}$$

So finally $q \sqsupseteq \text{semi}\ q' = \mathbf{I}$.

Correctness of semi: raising partial terms up to $\mathbf{I}$

We know q' is healthy, but not at full strength

$$\lambda \mathsf{f}, \underline{\mathsf{x}}.\mathsf{f}\ \underline{\mathsf{x}} \quad \sqsupseteq \quad \mathsf{q}' \quad \sqsupseteq \quad \lambda \mathsf{f}, \underline{\mathsf{x}}.\ \mathsf{f}\ \bot^{\sim \mathsf{n}}$$

Raise and lower n times to ignore faulty args

$$\begin{aligned} & \mathsf{n}\ \mathbf{K} \to \langle \top \rangle\ \mathsf{q}'\ \mathsf{f} \\ & \quad = \ \mathsf{q}'\ (\mathsf{n}\ \mathbf{K}\ \mathsf{f})\ \top^{\sim \mathsf{n}} \\ & \quad = \ (\lambda \underline{\mathsf{x}}.\ \mathsf{n}\ \mathbf{K}\ \mathsf{f}\ \underline{\mathsf{M}})\ \top^{\sim \mathsf{n}} \\ & \quad = \ (\lambda \underline{\mathsf{x}}.\ \mathsf{f})\ \top^{\sim \mathsf{n}} \\ & \quad = \ \mathsf{f} \end{aligned}$$

So finally $\mathsf{q} \sqsupseteq \text{semi}\ \mathsf{q}' = \mathbf{I}$. back