A System F accounting for scalars

– arXiv:0903.3741 –

Pablo Arrighi and Alejandro Díaz-Caro

Université de Grenoble
Laboratoire d'Informatique de Grenoble

November 19th, 2009. PPS (Paris)

LIG

Global context

- Oddity of Quantum theory $\Longrightarrow$ Quantum Logic?[1] (developed *ad hoc* before quantum computing, no clear relation with quantum programs).

[1] Birkhoff, G. and J. von Neumann, *The logic of quantum mechanics*, Annals of Mathematics **37** (1936), pp. 823–843.

Global context

- Oddity of Quantum theory $\Longrightarrow$ Quantum Logic?[1] (developed *ad hoc* before quantum computing, no clear relation with quantum programs).
- Models of Linear Logics $\Longrightarrow$ Quantum Theory? (Coherent spaces, Micromechanics loses duplicability.)

[1] Birkhoff, G. and J. von Neumann, *The logic of quantum mechanics*, Annals of Mathematics **37** (1936), pp. 823–843.

Global context

- Oddity of Quantum theory $\Longrightarrow$ Quantum Logic?[1] (developed *ad hoc* before quantum computing, no clear relation with quantum programs).
- Models of Linear Logics $\Longrightarrow$ Quantum Theory? (Coherent spaces, Micromechanics loses duplicability.)
- Curry-Howard : (programs,types)$\Longrightarrow$(proofs,logics).
  Quantum Computation : (quantum programs, quantum types).
  CH+QC : (quantum th. proofs, quantum th. logics)?
  Quantum logics : isolating the reasoning behind quantum algorithms?

[1] Birkhoff, G. and J. von Neumann, *The logic of quantum mechanics*, Annals of Mathematics **37** (1936), pp. 823–843.

Global context

- Oddity of Quantum theory $\Longrightarrow$ Quantum Logic?[1] (developed *ad hoc* before quantum computing, no clear relation with quantum programs).
- Models of Linear Logics $\Longrightarrow$ Quantum Theory? (Coherent spaces, Micromechanics loses duplicability.)
- Curry-Howard : (programs,types)$\Longrightarrow$(proofs,logics).
  Quantum Computation : (quantum programs, quantum types).
  CH+QC : (quantum th. proofs, quantum th. logics)?
  Quantum logics : isolating the reasoning behind quantum algorithms?

What are quantum types?

[1] Birkhoff, G. and J. von Neumann, *The logic of quantum mechanics*, Annals of Mathematics **37** (1936), pp. 823–843.

Quantum theory

- States are (normalized) vectors $\mathbf{v}$.
  Vector space of o.n.b. $(\mathbf{b}_i)$. Then $\mathbf{v} = \sum_i \alpha_i \mathbf{b}_i$.

Quantum theory

- States are (normalized) vectors $\mathbf{v}$.
  Vector space of o.n.b. $(\mathbf{b}_i)$. Then $\mathbf{v} = \sum_i \alpha_i \mathbf{b}_i$.
- Evolutions are (unitary) linear operators U.
  $\mathbf{v}' = U\mathbf{v}$.

Quantum theory

- States are (normalized) vectors $\mathbf{v}$.
  Vector space of o.n.b. $(\mathbf{b}_i)$. Then $\mathbf{v} = \sum_i \alpha_i \mathbf{b}_i$.
- Evolutions are (unitary) linear operators U.
  $\mathbf{v}' = U\mathbf{v}$.
- Systems are put next to one another with $\otimes$.
  Bilinear just like application :
  $\mathbf{u} + \mathbf{v} \otimes \mathbf{w} = \mathbf{u} \otimes \mathbf{w} + \mathbf{v} \otimes \mathbf{w}$,
  $\mathbf{u} \otimes \mathbf{v} + \mathbf{w} = \mathbf{u} \otimes \mathbf{v} + \mathbf{u} \otimes \mathbf{w}$, ...

Quantum theory

- States are (normalized) vectors $\mathbf{v}$.
  Vector space of o.n.b. $(\mathbf{b}_i)$. Then $\mathbf{v} = \sum_i \alpha_i \mathbf{b}_i$.
- Evolutions are (unitary) linear operators U.
  $\mathbf{v}' = U\mathbf{v}$.
- Systems are put next to one another with $\otimes$.
  Bilinear just like application :
  $\mathbf{u} + \mathbf{v} \otimes \mathbf{w} = \mathbf{u} \otimes \mathbf{w} + \mathbf{v} \otimes \mathbf{w}$,
  $\mathbf{u} \otimes \mathbf{v} + \mathbf{w} = \mathbf{u} \otimes \mathbf{v} + \mathbf{u} \otimes \mathbf{w}$, ...

No-cloning theorem!

No-cloning theorem

Statement: $\nexists U \ / \ \forall \mathbf{v} : \ U\mathbf{v} = \mathbf{v} \otimes \mathbf{v}$.
Proof:
Vector space of o.n.b. $(\mathbf{b}_i)$, so $\mathbf{v} = \sum_i \alpha_i \mathbf{b}_i$. We can have

$U\mathbf{b}_i = \mathbf{b}_i \otimes \mathbf{b}_i$ (=copying, OK)

But then

$$\begin{aligned} U\mathbf{v} &= U\sum_i \alpha_i \mathbf{b}_i = \sum_i \alpha_i U\mathbf{b}_i \\ &= \sum_i \alpha_i \mathbf{b}_i \otimes \mathbf{b}_i \neq \sum_{ij} \alpha_i\alpha_j \mathbf{b}_i \otimes \mathbf{b}_j \\ &= (\sum_i \alpha_i \mathbf{b}_i) \otimes (\sum_j \alpha_j \mathbf{b}_j) \\ &= \mathbf{v} \otimes \mathbf{v} \qquad \text{(=cloning, Not OK)} \end{aligned}$$

No-cloning theorem

Statement: $\nexists U \;/\; \forall \mathbf{v} : \; U\mathbf{v} = \mathbf{v} \otimes \mathbf{v}$.
Proof:
Vector space of o.n.b. $(\mathbf{b}_i)$, so $\mathbf{v} = \sum_i \alpha_i \mathbf{b}_i$. We can have

$U\mathbf{b}_i = \mathbf{b}_i \otimes \mathbf{b}_i$ (=copying, OK)

But then

$$\begin{aligned} U\mathbf{v} &= U\sum_i \alpha_i \mathbf{b}_i = \sum_i \alpha_i U\mathbf{b}_i \\ &= \sum_i \alpha_i \mathbf{b}_i \otimes \mathbf{b}_i \neq \sum_{ij} \alpha_i\alpha_j \mathbf{b}_i \otimes \mathbf{b}_j \\ &= (\sum_i \alpha_i \mathbf{b}_i) \otimes (\sum_j \alpha_j \mathbf{b}_j) \\ &= \mathbf{v} \otimes \mathbf{v} \qquad \text{(=cloning, Not OK)} \end{aligned}$$

Conflicts with β-reduction?

Linear-Algebraic λ-Calculus[2]

The language

Higher-order computation

$\mathbf{t} ::= \quad x \mid \lambda x.\mathbf{t} \mid (\mathbf{t}\,\mathbf{t}) \qquad \mid$

[2] Arrighi, P. and G. Dowek. *Linear-algebraic λ-calculus: higher-order, encodings and confluence*. Lecture Notes in Computer Science (RTA'08), **5117** (2008), pp. 17–31.

Linear-Algebraic λ-Calculus[2]

The language

Higher-order computation | **Linear algebra**

$\mathbf{t} ::= \quad x \mid \lambda x.\mathbf{t} \mid (\mathbf{t}\,\mathbf{t}) \quad \mid \quad \mathbf{t} + \mathbf{t} \mid \alpha.\mathbf{t} \mid \mathbf{0}$

[2] Arrighi, P. and G. Dowek. *Linear-algebraic λ-calculus: higher-order, encodings and confluence*. Lecture Notes in Computer Science (RTA'08), **5117** (2008), pp. 17–31.

Linear-Algebraic λ-Calculus[2]

The language

Higher-order computation | **Linear algebra**

$\mathbf{t} ::= \quad x \mid \lambda x.\mathbf{t} \mid (\mathbf{t}\,\mathbf{t}) \qquad \mid \qquad \mathbf{t} + \mathbf{t} \mid \alpha.\mathbf{t} \mid \mathbf{0}$

- $\lambda x.\mathbf{t}\,\mathbf{b} \to \mathbf{t}[\mathbf{b}/x]\ (*)$

(*) **b** an abstraction or a variable.

[2] Arrighi, P. and G. Dowek. *Linear-algebraic λ-calculus: higher-order, encodings and confluence*. Lecture Notes in Computer Science (RTA'08), **5117** (2008), pp. 17–31.

Linear-Algebraic λ-Calculus[2]

The language

Higher-order computation

$\mathbf{t} ::= \quad x \mid \lambda x.\mathbf{t} \mid (\mathbf{t}\ \mathbf{t}) \quad \mid$

- $\lambda x.\mathbf{t}\ \mathbf{b} \to \mathbf{t}[\mathbf{b}/x]\ (*)$

(*) **b** an abstraction or a variable.

(**) **u** closed normal.
(***) **u** and $\mathbf{u}+\mathbf{v}$ closed normal.

Linear algebra

$\mathbf{t}+\mathbf{t} \mid \alpha.\mathbf{t} \mid \mathbf{0}$

- Elementary rules such as $\mathbf{u}+\mathbf{0} \to \mathbf{u}$ and $\alpha.(\mathbf{u}+\mathbf{v}) \to \alpha.\mathbf{u}+\alpha.\mathbf{v}$.
- Factorisation rules such as $\alpha.\mathbf{u}+\beta.\mathbf{u} \to (\alpha+\beta).\mathbf{u}$. (**)
- Application rules such as $\mathbf{u}\ (\mathbf{v}+\mathbf{w}) \to (\mathbf{u}\ \mathbf{v})+(\mathbf{u}\ \mathbf{w})$. (***)

[2] Arrighi, P. and G. Dowek. *Linear-algebraic λ-calculus: higher-order, encodings and confluence*. Lecture Notes in Computer Science (RTA'08), **5117** (2008), pp. 17–31.

Linear-Algebraic λ-Calculus

Why the restrictions : Copying vs Cloning

Untyped λ-calculus + linear algebra $\Rightarrow$ Cloning?

Linear-Algebraic λ-Calculus

Why the restrictions : Copying vs Cloning

Untyped λ-calculus + linear algebra $\Rightarrow$ Cloning?

$$\lambda x.(x \otimes x)\ \sum_i \alpha_i \mathbf{b}_i \rightarrow^* \sum_i \alpha_i \mathbf{b}_i \otimes \mathbf{b}_i$$

$$\downarrow$$

$$(\sum_i \alpha_i \mathbf{b}_i) \otimes (\sum_i \alpha_i \mathbf{b}_i)$$

Linear-Algebraic λ-Calculus

Why the restrictions : Copying vs Cloning

Untyped λ-calculus + linear algebra $\Rightarrow$ Cloning?

$$\lambda x.(x \otimes x) \ \sum_i \alpha_i \mathbf{b}_i \to^* \sum_i \alpha_i \mathbf{b}_i \otimes \mathbf{b}_i$$

$$\downarrow$$

$$(\sum_i \alpha_i \mathbf{b}_i) \otimes (\sum_i \alpha_i \mathbf{b}_i)$$

No-cloning says bottom reduction forbidden. We must delay beta reduction till after linearity. So restrict beta reduction to **base vectors** $\to$ *i.e.* abstractions or variables.

Linear-Algebraic λ-Calculus

Why the restrictions : Infinities

Untyped λ-calculus + linear algebra $\Rightarrow \infty$

Linear-Algebraic λ-Calculus

Why the restrictions : Infinities

Untyped λ-calculus + linear algebra $\Rightarrow \infty$

$$\mathbf{Yb} \equiv \lambda x.(\mathbf{b} + (x\ x))\ \lambda x.(\mathbf{b} + (x\ x))$$

$$\mathbf{Yb} \to \mathbf{b} + \mathbf{Yb}$$

Linear-Algebraic λ-Calculus

Why the restrictions : Infinities

Untyped λ-calculus + linear algebra $\Rightarrow \infty$

$$\mathbf{Yb} \equiv \lambda x.(\mathbf{b} + (x\ x))\ \lambda x.(\mathbf{b} + (x\ x))$$

$$\mathbf{Yb} \to \mathbf{b} + \mathbf{Yb}$$

But whoever says infinity says trouble says...

Linear-Algebraic λ-Calculus

Why the restrictions : Infinities

Untyped λ-calculus + linear algebra $\Rightarrow \infty$

$$\mathbf{Yb} \equiv \lambda x.(\mathbf{b} + (x\ x))\ \lambda x.(\mathbf{b} + (x\ x))$$

$$\mathbf{Yb} \to \mathbf{b} + \mathbf{Yb}$$

But whoever says infinity says trouble says... indefinite forms.

Linear-Algebraic λ-Calculus

Why the restrictions : Infinities

Untyped λ-calculus + linear algebra $\Rightarrow \infty$

$$\mathbf{Yb} \equiv \lambda x.(\mathbf{b} + (x\ x))\ \lambda x.(\mathbf{b} + (x\ x))$$

$$\mathbf{Yb} \to \mathbf{b} + \mathbf{Yb}$$

But whoever says infinity says trouble says... indefinite forms.

$$\begin{array}{l} \mathbf{Yb} - \mathbf{Yb} \to \mathbf{b} + \mathbf{Yb} - \mathbf{Yb} \to \mathbf{b} \\ \downarrow_* \\ \mathbf{0} \end{array}$$

Linear-Algebraic λ-Calculus

Why the restrictions : Infinities

Untyped λ-calculus + linear algebra $\Rightarrow \infty$

$$\mathbf{Yb} \equiv \lambda x.(\mathbf{b} + (x\ x))\ \lambda x.(\mathbf{b} + (x\ x))$$

$$\mathbf{Yb} \to \mathbf{b} + \mathbf{Yb}$$

But whoever says infinity says trouble says... indefinite forms.

$$\mathbf{Yb} - \mathbf{Yb} \to \mathbf{b} + \mathbf{Yb} - \mathbf{Yb} \to \mathbf{b}$$

$$\downarrow_*$$

$$\mathbf{0}$$

High school teacher says we must restrict factorization rules to **finite vectors** $\to$ *i.e.* closed-normal forms.

System F

Straightforward extension of System F ($\lambda 2^{la}$)

System F rules plus simple rules to type algebraic terms

$$\frac{}{\Gamma \vdash \mathbf{0} : A}\, ax_0 \qquad \frac{\Gamma \vdash \mathbf{u} : A \qquad \Gamma \vdash \mathbf{v} : A}{\Gamma \vdash \mathbf{u} + \mathbf{v} : A}\, +I \qquad \frac{\Gamma \vdash \mathbf{t} : A}{\Gamma \vdash \alpha.\mathbf{t} : A}\, \alpha I$$

System F

Straightforward extension of System F ($\lambda 2^{la}$)

System F rules plus simple rules to type algebraic terms

$$\frac{}{\Gamma \vdash \mathbf{0} : A}\ ax_0 \qquad \frac{\Gamma \vdash \mathbf{u} : A \qquad \Gamma \vdash \mathbf{v} : A}{\Gamma \vdash \mathbf{u} + \mathbf{v} : A}\ +I \qquad \frac{\Gamma \vdash \mathbf{t} : A}{\Gamma \vdash \alpha.\mathbf{t} : A}\ \alpha I$$

Theorem (Strong normalization)

$\Gamma \vdash \boldsymbol{t} : T \Rightarrow \boldsymbol{t}$ *is strongly normalising.*

System F

Straightforward extension of System F ($\lambda 2^{la}$)

System F rules plus simple rules to type algebraic terms

$$\frac{}{\Gamma \vdash \mathbf{0} : A}\ ax_0 \qquad \frac{\Gamma \vdash \mathbf{u} : A \qquad \Gamma \vdash \mathbf{v} : A}{\Gamma \vdash \mathbf{u} + \mathbf{v} : A}\ +I \qquad \frac{\Gamma \vdash \mathbf{t} : A}{\Gamma \vdash \alpha.\mathbf{t} : A}\ \alpha I$$

Theorem (Strong normalization)

$\Gamma \vdash \boldsymbol{t} : T \Rightarrow \boldsymbol{t}$ *is strongly normalising.*

Proof. Sketch: We extend the notion of saturated sets.

System F

Strong normalization of $\lambda 2^{la}$

- SN: Set of strongly normalising terms

System F

Strong normalization of $\lambda 2^{la}$

- SN: Set of strongly normalising terms
- A subset $X \in SN$ is saturated if
  1. $\forall n \geq 0$, $(((x\ \mathbf{t}_1)\dots)\ \mathbf{t}_n) \in X$ where $\mathbf{t}_i \in SN$;
  2. $\mathbf{v}[\mathbf{b}/x]\ \overrightarrow{\mathbf{t}} \in X \Rightarrow (\lambda x\ \mathbf{v})\ \mathbf{b}\ \overrightarrow{\mathbf{t}} \in X$;
  3. $\mathbf{t}, \mathbf{u} \in X \Rightarrow \mathbf{t} + \mathbf{u} \in X$;
  4. $\forall \alpha$, $\mathbf{t} \in X \Rightarrow \alpha.\mathbf{t} \in X$;
  5. $\forall i \in I$, $((\mathbf{u}_i\ \mathbf{w}_1)\dots\ \mathbf{w}_n) \in X \Rightarrow \left(\left(\sum_{i\in I} \mathbf{u}_i\right)\ \mathbf{w}_1\right)\dots\ \mathbf{w}_n \in X$;
  6. $\forall i \in I$, $(\mathbf{u}\ \mathbf{w}_i) \in X \Rightarrow \mathbf{u}\ \left(\sum_{i\in I} \mathbf{w}_i\right) \in X$;
  7. $\alpha.((\mathbf{t}_1\ \mathbf{t}_2)\dots\mathbf{t}_n) \in X \Leftrightarrow ((\mathbf{t}_1\ \mathbf{t}_2)\dots\alpha.\mathbf{t}_k)\dots\mathbf{t}_n \in X\ (1 \leq k \leq n)$;
  8. $\mathbf{0} \in X$;
  9. $\forall \overrightarrow{\mathbf{t}} \in SN$, $(\mathbf{0}\ \overrightarrow{\mathbf{t}}) \in X$;
  10. $\forall \mathbf{t}, \overrightarrow{\mathbf{u}} \in SN$, $(\mathbf{t}\ \mathbf{0})\ \overrightarrow{\mathbf{u}} \in X$.

X stable by "construction" and "anti-reduction"

System F

Strong normalization of $\lambda 2^{la}$

- SN: Set of strongly normalising terms
- A subset $X \in SN$ is saturated if
  1. $\forall n \geq 0$, $(((x\ \mathbf{t}_1)\dots)\ \mathbf{t}_n) \in X$ where $\mathbf{t}_i \in SN$;
  2. $\mathbf{v}[\mathbf{b}/x]\ \overrightarrow{\mathbf{t}} \in X \Rightarrow (\lambda x\ \mathbf{v})\ \mathbf{b}\ \overrightarrow{\mathbf{t}} \in X$;
  3. $\mathbf{t}, \mathbf{u} \in X \Rightarrow \mathbf{t} + \mathbf{u} \in X$;
  4. $\forall \alpha$, $\mathbf{t} \in X \Rightarrow \alpha.\mathbf{t} \in X$;
  5. $\forall i \in I$, $((\mathbf{u}_i\ \mathbf{w}_1)\dots\ \mathbf{w}_n) \in X \Rightarrow \left(\left(\sum_{i\in I}\mathbf{u}_i\right)\ \mathbf{w}_1\right)\dots\ \mathbf{w}_n \in X$;
  6. $\forall i \in I$, $(\mathbf{u}\ \mathbf{w}_i) \in X \Rightarrow \mathbf{u}\ \left(\sum_{i\in I}\mathbf{w}_i\right) \in X$;
  7. $\alpha.((\mathbf{t}_1\ \mathbf{t}_2)\dots\mathbf{t}_n) \in X \Leftrightarrow ((\mathbf{t}_1\ \mathbf{t}_2)\dots\alpha.\mathbf{t}_k)\dots\mathbf{t}_n \in X\ (1 \leq k \leq n)$;
  8. $\mathbf{0} \in X$;
  9. $\forall \overrightarrow{\mathbf{t}} \in SN$, $(\mathbf{0}\ \overrightarrow{\mathbf{t}}) \in X$;
  10. $\forall \mathbf{t}, \overrightarrow{\mathbf{u}} \in SN$, $(\mathbf{t}\ \mathbf{0})\ \overrightarrow{\mathbf{u}} \in X$.

  X stable by "construction" and "anti-reduction"
- SAT is the set of all saturated sets

System F

Strong normalization of $\lambda 2^{la}$ (II)

Refining the sketch:

- The idea is that types "correspond" to saturated sets.

System F

Strong normalization of $\lambda 2^{la}$ (II)

Refining the sketch:

- The idea is that types "correspond" to saturated sets.
- This correspondance is achived by a maping from types to SAT.

System F

Strong normalization of $\lambda 2^{la}$ (III)

Lemma

1. $SN \in SAT$,
2. $A, B \in SAT \Rightarrow A \to B \in SAT$,
3. *For all collection* A_i *of members of* SAT, $\bigcap_i A_i \in SAT$,

System F

Strong normalization of $\lambda 2^{la}$ (III)

Lemma

1. $SN \in SAT$,
2. $A, B \in SAT \Rightarrow A \to B \in SAT$,
3. *For all collection* A_i *of members of* SAT, $\bigcap_i A_i \in SAT$,

Definition (Mapping)

- $[\![X]\!]_\xi = \xi(X)$ (where $\xi(\cdot) : TVar \to SAT$)
- $[\![A \to B]\!]_\xi = [\![A]\!]_\xi \to [\![B]\!]_\xi$
- $[\![\forall X.T]\!]_\xi = \bigcap_{Y \in SAT} [\![T]\!]_{\xi(X:=Y)}$

System F

Strong normalization of $\lambda 2^{la}$ (III)

Lemma

1. $SN \in SAT$,
2. $A, B \in SAT \Rightarrow A \to B \in SAT$,
3. *For all collection* A_i *of members of* SAT, $\bigcap_i A_i \in SAT$,

Definition (Mapping)

- $[\![X]\!]_\xi = \xi(X)$ (where $\xi(\cdot) : TVar \to SAT$)
- $[\![A \to B]\!]_\xi = [\![A]\!]_\xi \to [\![B]\!]_\xi$
- $[\![\forall X.T]\!]_\xi = \bigcap_{Y \in SAT} [\![T]\!]_{\xi(X:=Y)}$

Lemma

Given a valuation ξ, $[\![T]\!]_\xi \in SAT$

System F

Strong normalization of $\lambda 2^{la}$ (IV)

Definition ($\vDash$)

For $\Gamma = x_1 : A_1, \dots, x_n : A_n$, $\Gamma \vDash \mathbf{t} : T$ means that $\forall \xi$,

$$x_1 \in [\![A_1]\!]_\xi, \dots x_n \in [\![A_n]\!]_\xi \Rightarrow \mathbf{t} \in [\![T]\!]_\xi$$

System F

Strong normalization of $\lambda 2^{la}$ (IV)

Definition ($\vDash$)

For $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, $\Gamma \vDash \mathbf{t} : T$ means that $\forall \xi$,

$$x_1 \in [\![A_1]\!]_\xi, \ldots x_n \in [\![A_n]\!]_\xi \Rightarrow \mathbf{t} \in [\![T]\!]_\xi$$

Refining the sketch: Prove that $\Gamma \vdash \mathbf{t} : T \Rightarrow \Gamma \vDash \mathbf{t} : T$

We prove this by induction on the derivation of $\Gamma \vdash \mathbf{t} : T$ (In fact the definition of $\vDash$ is slightly different to strengthen the induction hypothesis)

System F

Strong normalization of $\lambda 2^{la}$ (V)

Then, the proof of the strong normalisation theorem is:

Let $\Gamma \vdash \mathbf{t} : T$

System F

Strong normalization of $\lambda 2^{la}$ (V)

Then, the proof of the strong normalisation theorem is:

Let $\Gamma \vdash \mathbf{t} : T$
$\Rightarrow \Gamma \vDash \mathbf{t} : T$

System F

Strong normalization of $\lambda 2^{la}$ (V)

Then, the proof of the strong normalisation theorem is:

Let $\Gamma \vdash \mathbf{t} : T$
$\Rightarrow \Gamma \models \mathbf{t} : T$
$\Rightarrow$ If $\forall (x_i : A_i) \in \Gamma$, $x_i \in [\![A_i]\!]_\xi$ then $\mathbf{t} \in [\![T]\!]_\xi$

System F

Strong normalization of $\lambda 2^{la}$ (V)

Then, the proof of the strong normalisation theorem is:

Let $\Gamma \vdash \mathbf{t} : T$
$\Rightarrow \Gamma \models \mathbf{t} : T$
$\Rightarrow$ If $\forall (x_i : A_i) \in \Gamma$, $x_i \in [\![A_i]\!]_\xi$ then $\mathbf{t} \in [\![T]\!]_\xi$

Note that

- $x_i \in [\![A_i]\!]_\xi$ because $[\![A_i]\!]_\xi$ is saturated,

System F

Strong normalization of $\lambda 2^{la}$ (V)

Then, the proof of the strong normalisation theorem is:

Let $\Gamma \vdash \mathbf{t} : T$
$\Rightarrow \Gamma \vDash \mathbf{t} : T$
$\Rightarrow$ If $\forall (x_i : A_i) \in \Gamma$, $x_i \in [\![A_i]\!]_\xi$ then $\mathbf{t} \in [\![T]\!]_\xi$

Note that

- $x_i \in [\![A_i]\!]_\xi$ because $[\![A_i]\!]_\xi$ is saturated,
- then $\mathbf{t}$ is strong normalising because $[\![T]\!]_\xi \subseteq SN$

System F

Linear-Algebraic λ-Calculus with $\lambda 2^{la}$

Higher-order computation

$\mathbf{t} ::= \quad x \mid \lambda x.\mathbf{t} \mid (\mathbf{t}\,\mathbf{t}) \qquad \mid$

- $\lambda x.\mathbf{t}\,\mathbf{b} \to \mathbf{t}[\mathbf{b}/x]\ (*)$

(*) **b** an abstraction or a variable.

Linear algebra

$\mathbf{t} + \mathbf{t} \mid \alpha.\mathbf{t} \mid \mathbf{0}$

- Elementary rules such as $\mathbf{u} + \mathbf{0} \to \mathbf{u}$ and $\alpha.(\mathbf{u} + \mathbf{v}) \to \alpha.\mathbf{u} + \alpha.\mathbf{v}$.
- Factorisation rules such as $\alpha.\mathbf{u} + \beta.\mathbf{u} \to (\alpha + \beta).\mathbf{u}$.
- Application rules such as $\mathbf{u}\ (\mathbf{v} + \mathbf{w}) \to (\mathbf{u}\ \mathbf{v}) + (\mathbf{u}\ \mathbf{w})$.

System F

Linear-Algebraic λ-Calculus with $\lambda 2^{la}$

Higher-order computation

$\mathbf{t} ::= \quad x \,|\, \lambda x.\mathbf{t} \,|\, (\mathbf{t}\,\mathbf{t}) \qquad |$

- $\lambda x.\mathbf{t}\,\mathbf{b} \to \mathbf{t}[\mathbf{b}/x]\,(*)$

(*) **b** an abstraction or a variable.

Every typable term is strong normalizing

Linear algebra

$\mathbf{t} + \mathbf{t} \,|\, \alpha.\mathbf{t} \,|\, \mathbf{0}$

- Elementary rules such as $\mathbf{u} + \mathbf{0} \to \mathbf{u}$ and $\alpha.(\mathbf{u} + \mathbf{v}) \to \alpha.\mathbf{u} + \alpha.\mathbf{v}$.
- Factorisation rules such as $\alpha.\mathbf{u} + \beta.\mathbf{u} \to (\alpha + \beta).\mathbf{u}$.
- Application rules such as $\mathbf{u}\ (\mathbf{v} + \mathbf{w}) \to (\mathbf{u}\ \mathbf{v}) + (\mathbf{u}\ \mathbf{w})$.

System F

Linear-Algebraic λ-Calculus with $\lambda 2^{la}$

Higher-order computation

$\mathbf{t} ::= \quad x \,|\, \lambda x.\mathbf{t} \,|\, (\mathbf{t}\,\mathbf{t}) \qquad |$

- $\lambda x.\mathbf{t}\,\mathbf{b} \to \mathbf{t}[\mathbf{b}/x]\,(*)$

(*) **b** an abstraction or a variable.

Every typable term is strong normalizing

Hence **Yb** is no typable!

Linear algebra

$\mathbf{t} + \mathbf{t} \,|\, \alpha.\mathbf{t} \,|\, \mathbf{0}$

- Elementary rules such as $\mathbf{u} + \mathbf{0} \to \mathbf{u}$ and $\alpha.(\mathbf{u} + \mathbf{v}) \to \alpha.\mathbf{u} + \alpha.\mathbf{v}$.
- Factorisation rules such as $\alpha.\mathbf{u} + \beta.\mathbf{u} \to (\alpha + \beta).\mathbf{u}$.
- Application rules such as $\mathbf{u}\;(\mathbf{v} + \mathbf{w}) \to (\mathbf{u}\;\mathbf{v}) + (\mathbf{u}\;\mathbf{w})$.

System F

Linear-Algebraic λ-Calculus with $\lambda 2^{la}$

Higher-order computation

$\mathbf{t} ::= \quad x \mid \lambda x.\mathbf{t} \mid (\mathbf{t}\,\mathbf{t}) \quad \mid$

- $\lambda x.\mathbf{t}\,\mathbf{b} \to \mathbf{t}[\mathbf{b}/x]\ (*)$

(*) **b** an abstraction or a variable.
Every typable term is strong normalizing
Hence **Yb** is no typable!
$\mathbf{t} - \mathbf{t} \to \mathbf{0}$ always, so it is not necesary to reduce **t** first. we can remove the closed-normal restrictions!

Linear algebra

$\mathbf{t} + \mathbf{t} \mid \alpha.\mathbf{t} \mid \mathbf{0}$

- Elementary rules such as $\mathbf{u} + \mathbf{0} \to \mathbf{u}$ and $\alpha.(\mathbf{u} + \mathbf{v}) \to \alpha.\mathbf{u} + \alpha.\mathbf{v}$.
- Factorisation rules such as $\alpha.\mathbf{u} + \beta.\mathbf{u} \to (\alpha + \beta).\mathbf{u}$.
- Application rules such as $\mathbf{u}\ (\mathbf{v} + \mathbf{w}) \to (\mathbf{u}\ \mathbf{v}) + (\mathbf{u}\ \mathbf{w})$.

The *scalar* type system
Grammar

Types grammar:

$$\mathcal{T} = \mathcal{U} \mid \forall X.\mathcal{T} \mid \alpha.\mathcal{T} \mid \bar{0},$$

$$\mathcal{U} = X \mid \mathcal{U} \to \mathcal{T} \mid \forall X.\mathcal{U}$$

where $\alpha \in \mathcal{S}$ and $(\mathcal{S}, +, \times)$ is a conmutative ring.

The *scalar* type system

Type inference rules

$$\frac{}{\Gamma, x : U \vdash x : U}\ ax[U]$$

$$\frac{\Gamma \vdash \mathbf{u} : U \to T \qquad \Gamma \vdash \mathbf{v} : U}{\Gamma \vdash (\mathbf{u}\ \mathbf{v}) : T} \to E \qquad \frac{\Gamma, x : U \vdash \mathbf{t} : T}{\Gamma \vdash \lambda x\ \mathbf{t} : U \to T} \to I[U]$$

$$\frac{\Gamma \vdash \mathbf{u} : \forall X.T}{\Gamma \vdash \mathbf{u} : T[U/X]}\ \forall E[X := U] \qquad \frac{\Gamma \vdash \mathbf{u} : T}{\Gamma \vdash \mathbf{u} : \forall X.T}\ \forall I[X] \text{ with } X \notin FV(\Gamma)$$

The *scalar* type system

Type inference rules

$$\frac{}{\Gamma, x : U \vdash x : U}\ ax[U]$$

$$\frac{\Gamma \vdash \mathbf{u} : U \to T \qquad \Gamma \vdash \mathbf{v} : U}{\Gamma \vdash (\mathbf{u}\ \mathbf{v}) : T} \to E \qquad \frac{\Gamma, x : U \vdash \mathbf{t} : T}{\Gamma \vdash \lambda x\ \mathbf{t} : U \to T} \to I[U]$$

$$\frac{\Gamma \vdash \mathbf{u} : \forall X.T}{\Gamma \vdash \mathbf{u} : T[U/X]}\ \forall E[X := U] \qquad \frac{\Gamma \vdash \mathbf{u} : T}{\Gamma \vdash \mathbf{u} : \forall X.T}\ \forall I[X] \text{ with } X \notin FV(\Gamma)$$

$$\frac{}{\Gamma \vdash \mathbf{0} : T}\ ax_0 \qquad \frac{\Gamma \vdash \mathbf{u} : T \qquad \Gamma \vdash \mathbf{v} : T}{\Gamma \vdash \mathbf{u} + \mathbf{v} : T} +I \qquad \frac{\Gamma \vdash \mathbf{u} : T}{\Gamma \vdash \alpha.\mathbf{u} : T}\ \alpha I$$

The *scalar* type system

Type inference rules

$$\frac{}{\Gamma, x:U \vdash x:U}\ ax[U]$$

$$\frac{\Gamma \vdash \mathbf{u}:\alpha.(U \to T) \qquad \Gamma \vdash \mathbf{v}:\beta.U}{\Gamma \vdash (\mathbf{u}\ \mathbf{v}):(\alpha \times \beta).T} \to E \qquad \frac{\Gamma, x:U \vdash \mathbf{t}:T}{\Gamma \vdash \lambda x\ \mathbf{t}:U \to T} \to I[U]$$

$$\frac{\Gamma \vdash \mathbf{u}:\forall X.T}{\Gamma \vdash \mathbf{u}:T[U/X]}\ \forall E[X := U] \qquad \frac{\Gamma \vdash \mathbf{u}:T}{\Gamma \vdash \mathbf{u}:\forall X.T}\ \forall I[X] \text{ with } X \notin FV(\Gamma)$$

$$\frac{}{\Gamma \vdash \mathbf{0}:\overline{0}}\ ax_{\overline{0}} \quad \frac{\Gamma \vdash \mathbf{u}:\alpha.T \qquad \Gamma \vdash \mathbf{v}:\beta.T}{\Gamma \vdash \mathbf{u}+\mathbf{v}:(\alpha+\beta).T} +I \qquad \frac{\Gamma \vdash \mathbf{u}:T}{\Gamma \vdash \alpha.\mathbf{u}:\alpha.T}\ sI[\alpha]$$

Where $U \in \mathcal{U}$ and types in contexts are are in $\mathcal{U}$.

The *scalar* type system
Strong normalisation

- $(\cdot)^\natural$: map that take types and remove all the scalars on it.

The *scalar* type system
Strong normalisation

- $(\cdot)^\natural$: map that take types and remove all the scalars on it.
  Example: $(U \to \alpha.X)^\natural = U^\natural \to X$

The *scalar* type system
Strong normalisation

- $(\cdot)^\natural$: map that take types and remove all the scalars on it.
  Example: $(U \to \alpha.X)^\natural = U^\natural \to X$
- We also define $\overline{0}^\natural = T$ for some T without scalars.

The *scalar* type system
Strong normalisation

- $(\cdot)^\natural$: map that take types and remove all the scalars on it. Example: $(U \to \alpha.X)^\natural = U^\natural \to X$
- We also define $\overline{0}^\natural = T$ for some T without scalars.

Lemma (Correspondence with $\lambda 2^{la}$)

$\Gamma \vdash \boldsymbol{t}\colon T \Rightarrow \Gamma^\natural \vdash_{\lambda 2^{la}} \boldsymbol{t}\colon T^\natural$.

The *scalar* type system
Strong normalisation

- $(\cdot)^\natural$: map that take types and remove all the scalars on it.
  Example: $(U \to \alpha.X)^\natural = U^\natural \to X$
- We also define $\overline{0}^\natural = T$ for some T without scalars.

Lemma (Correspondence with $\lambda 2^{la}$)

$\Gamma \vdash \boldsymbol{t} : T \Rightarrow \Gamma^\natural \vdash_{\lambda 2^{la}} \boldsymbol{t} : T^\natural$.

Theorem (Strong normalisation)

$\Gamma \vdash \boldsymbol{t} : T \Rightarrow \boldsymbol{t}$ *is strongly normalising.*

Proof. By previous lemma $\Gamma^\natural \vdash_{\lambda 2^{la}} \mathbf{t} : T^\natural$, then $\mathbf{t}$ is strong normalising.

The *scalar* type system

Subject reduction

Theorem (Subject Reduction)

Let $t \to^* t'$. *Then* $\Gamma \vdash t : T \Rightarrow \Gamma \vdash t' : T$

The *scalar* type system
Subject reduction

Theorem (Subject Reduction)

Let $t \to^* t'$. Then $\Gamma \vdash t : T \Rightarrow \Gamma \vdash t' : T$

Proof. (sketch)

- We proof rule by rule that if $\mathbf{t} \to \mathbf{t}'$ using that rule and $\Gamma \vdash \mathbf{t} : T$, then $\Gamma \vdash \mathbf{t}' : T$.
- In general, the method is to take the term $\mathbf{t}$, decompose it into its small parts and recompose to $\mathbf{t}'$.

The *scalar* type system

Subject reduction proof: example (I)

To show a rule as example, we need some auxiliary lemmas and definitions:

The *scalar* type system

Subject reduction proof: example (I)

To show a rule as example, we need some auxiliary lemmas and definitions:

Order:

- Write $A > B$ if either
  - $B \equiv \forall X.A$ or
  - $A \equiv \forall X.C$ and $B \equiv C[U/X]$ for some $U \in \mathcal{U}$.
- $\geq$ is the reflexive and transitive closure of $>$.

The *scalar* type system
Subject reduction proof: example (I)

To show a rule as example, we need some auxiliary lemmas and definitions:

Order:

- Write $A > B$ if either
  - $B \equiv \forall X.A$ or
  - $A \equiv \forall X.C$ and $B \equiv C[U/X]$ for some $U \in \mathcal{U}$.
- $\geq$ is the reflexive and transitive closure of $>$.

Instuition of this definition:

Types in the numerator of

$$\frac{\Gamma \vdash \mathbf{t} : A}{\Gamma \vdash \mathbf{t} : \forall X.A} \forall I \text{ with } X \notin FV(\Gamma) \qquad \text{or} \qquad \frac{\Gamma \vdash \mathbf{t} : \forall X.C}{\Gamma \vdash \mathbf{t} : C[U/X]} \forall E$$

are greater than the types in the denominator.

The *scalar* type system

Subject reduction proof: example (II)

Some lemmas needed:

The *scalar* type system
Subject reduction proof: example (II)

Some lemmas needed:

1. $A \geq B$ and $\Gamma \vdash \mathbf{t} : A \Rightarrow \Gamma \vdash \mathbf{t} : B$

The *scalar* type system

Subject reduction proof: example (II)

Some lemmas needed:

1. $A \geq B$ and $\Gamma \vdash \mathbf{t} : A \Rightarrow \Gamma \vdash \mathbf{t} : B$
2. $A \geq B \Rightarrow \alpha A \geq \alpha B$

The *scalar* type system
Subject reduction proof: example (II)

Some lemmas needed:

1. $A \geq B$ and $\Gamma \vdash \mathbf{t} : A \Rightarrow \Gamma \vdash \mathbf{t} : B$
2. $A \geq B \Rightarrow \alpha A \geq \alpha B$
3. Generation lemma (app): Let $\Gamma \vdash (\mathbf{u}\ \mathbf{v}) : \gamma.T$, then

$$\begin{cases} \Gamma \vdash \mathbf{u} : \beta.U \to T' \\ \Gamma \vdash \mathbf{v} : \alpha.U \\ T' \geq T \\ \gamma = \alpha \times \beta \end{cases}$$

The *scalar* type system

Subject reduction proof: example (II)

Some lemmas needed:

1. $A \geq B$ and $\Gamma \vdash \mathbf{t} : A \Rightarrow \Gamma \vdash \mathbf{t} : B$
2. $A \geq B \Rightarrow \alpha A \geq \alpha B$
3. Generation lemma (app): Let $\Gamma \vdash (\mathbf{u}\ \mathbf{v}) : \gamma.T$, then

$$\begin{cases} \Gamma \vdash \mathbf{u} : \beta.U \rightarrow T' \\ \Gamma \vdash \mathbf{v} : \alpha.U \\ T' \geq T \\ \gamma = \alpha \times \beta \end{cases}$$

4. Generation lemma (sum): Let $\Gamma \vdash \mathbf{u} + \mathbf{v} : \gamma.T$, then

$$\begin{cases} \Gamma \vdash \mathbf{u} : \alpha.T \\ \Gamma \vdash \mathbf{v} : \beta.T \\ \gamma = \alpha + \beta \end{cases}$$

The *scalar* type system
Subject reduction proof: example (III)

Example: Rule $(\mathbf{u} + \mathbf{v})\ \mathbf{w} \to (\mathbf{u}\ \mathbf{w}) + (\mathbf{v}\ \mathbf{w})$.

The *scalar* type system

Subject reduction proof: example (III)

Example: Rule $(\mathbf{u}+\mathbf{v})\ \mathbf{w} \to (\mathbf{u}\ \mathbf{w}) + (\mathbf{v}\ \mathbf{w})$.

- Let $\Gamma \vdash (\mathbf{u}+\mathbf{v})\ \mathbf{w} : T$.

The *scalar* type system

Subject reduction proof: example (III)

Example: Rule $(\mathbf{u} + \mathbf{v})\ \mathbf{w} \to (\mathbf{u}\ \mathbf{w}) + (\mathbf{v}\ \mathbf{w})$.

- Let $\Gamma \vdash (\mathbf{u} + \mathbf{v})\ \mathbf{w} : T$.
- Using the previous lemmas we can prove that

$$\begin{cases} \Gamma \vdash \mathbf{u} : (\delta \times \beta).U \to T' \\ \Gamma \vdash \mathbf{v} : ((1-\delta) \times \beta).U \to T' \\ \Gamma \vdash \mathbf{w} : \alpha.U \end{cases}$$

where $\alpha \times \beta = 1$, $T' \geq T$ and δ is some scalar.

The *scalar* type system

Subject reduction proof: example (IV)

Then

$$\frac{\Gamma \vdash \mathbf{u} : (\delta \times \beta).U \to T' \qquad \Gamma \vdash \mathbf{w} : \alpha.U}{\Gamma \vdash (\mathbf{u}\ \mathbf{w}) : (\delta \times \beta \times \alpha).T' \geq \delta.T} \to E$$

The *scalar* type system

Subject reduction proof: example (IV)

Then

$$\frac{\Gamma \vdash \mathbf{u} : (\delta \times \beta).U \to T' \qquad \Gamma \vdash \mathbf{w} : \alpha.U}{\Gamma \vdash (\mathbf{u}\ \mathbf{w}) : (\delta \times \beta \times \alpha).T' \geq \delta.T} \to E$$

Also,

$$\frac{\Gamma \vdash \mathbf{v} : ((1-\delta) \times \beta).(U \to T') \qquad \Gamma \vdash \mathbf{w} : \alpha.U}{\Gamma \vdash (\mathbf{v}\ \mathbf{w}) : ((1-\delta) \times \beta \times \alpha).T' \geq (1-\delta).T} \to E$$

The *scalar* type system

Subject reduction proof: example (IV)

Then

$$\dfrac{\Gamma \vdash \mathbf{u} : (\delta \times \beta).U \rightarrow T' \qquad \Gamma \vdash \mathbf{w} : \alpha.U}{\Gamma \vdash (\mathbf{u}\ \mathbf{w}) : (\delta \times \beta \times \alpha).T' \geq \delta.T} \rightarrow E$$

Also,

$$\dfrac{\Gamma \vdash \mathbf{v} : ((1-\delta) \times \beta).(U \rightarrow T') \qquad \Gamma \vdash \mathbf{w} : \alpha.U}{\Gamma \vdash (\mathbf{v}\ \mathbf{w}) : ((1-\delta) \times \beta \times \alpha).T' \geq (1-\delta).T} \rightarrow E$$

So

$$\dfrac{\Gamma \vdash (\mathbf{u}\ \mathbf{w}) : \delta.T \qquad \Gamma \vdash (\mathbf{v}\ \mathbf{w}) : (1-\delta).T}{\Gamma \vdash (\mathbf{u}\ \mathbf{w}) + (\mathbf{v}\ \mathbf{w}) : T} +I \text{ and } \equiv$$

The *scalar* type system
Probabilistic type system: Intuition

- *Conditional* functions → same type on each branch.

The *scalar* type system

Probabilistic type system: Intuition

- *Conditional* functions → same type on each branch.
- By restricting the scalars to positive reals → probabilistic type system.

The *scalar* type system
Probabilistic type system: Intuition

- *Conditional* functions → same type on each branch.
- By restricting the scalars to positive reals → probabilistic type system.
- For example, one can type functions such as
$$\lambda x\ \{x\ [\tfrac{1}{2}.(\mathrm{true}+\mathrm{false})]\ [\tfrac{1}{4}.\mathrm{true}+\tfrac{3}{4}.\mathrm{false}]\}:\mathcal{B}\to\mathcal{B}$$
with the type system serving as a guarantee that the function conserves probabilities summing to one.

The *scalar* type system

Probabilistic type system: Formalisation

We define the *probabilistic* type system to be the *scalar* type system with the following restrictions:

The *scalar* type system
Probabilistic type system: Formalisation

We define the *probabilistic* type system to be the *scalar* type system with the following restrictions:

- $S = \mathbb{R}^+$,

The *scalar* type system

Probabilistic type system: Formalisation

We define the *probabilistic* type system to be the *scalar* type system with the following restrictions:

- $\mathcal{S} = \mathbb{R}^+$,
- Contexts: Types in contexts are classic types ($\mathcal{C}$), *i.e.* types in $\mathcal{U}$ exempt of any scalar,

The *scalar* type system
Probabilistic type system: Formalisation

We define the *probabilistic* type system to be the *scalar* type system with the following restrictions:

- $\mathcal{S} = \mathbb{R}^+$,
- Contexts: Types in contexts are classic types ($\mathcal{C}$), *i.e.* types in $\mathcal{U}$ exempt of any scalar,
- We change the rule $\forall E$ to

$$\frac{\Gamma \vdash \mathbf{t} : \forall X.T}{\Gamma \vdash \mathbf{t} : T[C/X]}\ \forall E \text{ with } C \in \mathcal{C}$$

The *scalar* type system

Probabilistic type system: Formalisation

We define the *probabilistic* type system to be the *scalar* type system with the following restrictions:

- $\mathcal{S} = \mathbb{R}^+$,
- Contexts: Types in contexts are classic types ($\mathcal{C}$), *i.e.* types in $\mathcal{U}$ exempt of any scalar,
- We change the rule $\forall E$ to

$$\frac{\Gamma \vdash \mathbf{t} : \forall X.T}{\Gamma \vdash \mathbf{t} : T[C/X]} \forall E \text{ with } C \in \mathcal{C}$$

- The final conclusion must be classic.

The *scalar* type system

Probabilistic type system: Proof

Definition (Weight function to check probability distributions)

Let $\omega : \Lambda \to \mathbb{R}^+$ be a function defined inductively by:

$$\omega(\mathbf{0}) = 0 \qquad \omega(\mathbf{t}_1 + \mathbf{t}_2) = \omega(\mathbf{t}_1) + \omega(\mathbf{t}_2)$$
$$\omega(\mathbf{b}) = 1 \qquad \omega(\alpha.\mathbf{t}) = \alpha \times \omega(\mathbf{t})$$
$$\omega(\mathbf{t}_1\ \mathbf{t}_2) = \omega(\mathbf{t}_1) \times \omega(\mathbf{t}_2)$$

The *scalar* type system

Probabilistic type system: Proof

Definition (Weight function to check probability distributions**)**

Let $\omega : \Lambda \to \mathbb{R}^+$ be a function defined inductively by:

$$\omega(\mathbf{0}) = 0 \qquad \omega(\mathbf{t}_1 + \mathbf{t}_2) = \omega(\mathbf{t}_1) + \omega(\mathbf{t}_2)$$
$$\omega(\mathbf{b}) = 1 \qquad \omega(\alpha.\mathbf{t}) = \alpha \times \omega(\mathbf{t})$$
$$\omega(\mathbf{t}_1\ \mathbf{t}_2) = \omega(\mathbf{t}_1) \times \omega(\mathbf{t}_2)$$

Theorem (Normal terms in *probabilistic* have weight 1)

$\Gamma_{\mathcal{C}} \vdash \boldsymbol{t} : C \Rightarrow \omega(\boldsymbol{t}\downarrow) = 1.$

Proof. We prove that $\Gamma \vdash \mathbf{t} : \alpha.C \Rightarrow \omega(\mathbf{t}\downarrow) = \alpha$ by structural induction over $\mathbf{t}\downarrow$.

The *scalar* type system

Probabilistic type system: Proof

Definition (Weight function to check probability distributions)

Let $\omega : \Lambda \to \mathbb{R}^+$ be a function defined inductively by:

$$\omega(\mathbf{0}) = 0 \qquad \omega(\mathbf{t}_1 + \mathbf{t}_2) = \omega(\mathbf{t}_1) + \omega(\mathbf{t}_2)$$
$$\omega(\mathbf{b}) = 1 \qquad \omega(\alpha.\mathbf{t}) = \alpha \times \omega(\mathbf{t})$$
$$\omega(\mathbf{t}_1\ \mathbf{t}_2) = \omega(\mathbf{t}_1) \times \omega(\mathbf{t}_2)$$

Theorem (Normal terms in *probabilistic* have weight 1)

$\Gamma_{\mathcal{C}} \vdash \boldsymbol{t} : C \Rightarrow \omega(\boldsymbol{t}\downarrow) = 1.$

Proof. We prove that $\Gamma \vdash \mathbf{t} : \alpha.C \Rightarrow \omega(\mathbf{t}\downarrow) = \alpha$ by structural induction over $\mathbf{t}\downarrow$.
Example: $2.(\lambda x\ \frac{1}{2}.x)\ y$

The *scalar* type system

Probabilistic type system: Proof

Definition (Weight function to check probability distributions)

Let $\omega : \Lambda \to \mathbb{R}^+$ be a function defined inductively by:

$$\omega(\mathbf{0}) = 0 \qquad \omega(\mathbf{t}_1 + \mathbf{t}_2) = \omega(\mathbf{t}_1) + \omega(\mathbf{t}_2)$$
$$\omega(\mathbf{b}) = 1 \qquad \omega(\alpha.\mathbf{t}) = \alpha \times \omega(\mathbf{t})$$
$$\omega(\mathbf{t}_1\ \mathbf{t}_2) = \omega(\mathbf{t}_1) \times \omega(\mathbf{t}_2)$$

Theorem (Normal terms in *probabilistic* have weight 1)

$\Gamma_{\mathcal{C}} \vdash \boldsymbol{t} : C \Rightarrow \omega(\boldsymbol{t}\!\downarrow) = 1.$

Proof. We prove that $\Gamma \vdash \mathbf{t} : \alpha.C \Rightarrow \omega(\mathbf{t}\!\downarrow) = \alpha$ by structural induction over $\mathbf{t}\!\downarrow$.

Example: $2.(\lambda x\ \frac{1}{2}.x)\ y$

$$\omega(2.(\lambda x\ \tfrac{1}{2}.x)\ y) = 2$$

The *scalar* type system
Probabilistic type system: Proof

Definition (Weight function to check probability distributions)

Let $\omega : \Lambda \to \mathbb{R}^+$ be a function defined inductively by:

$$\omega(\mathbf{0}) = 0 \qquad \omega(\mathbf{t}_1 + \mathbf{t}_2) = \omega(\mathbf{t}_1) + \omega(\mathbf{t}_2)$$
$$\omega(\mathbf{b}) = 1 \qquad \omega(\alpha.\mathbf{t}) = \alpha \times \omega(\mathbf{t})$$
$$\omega(\mathbf{t}_1\ \mathbf{t}_2) = \omega(\mathbf{t}_1) \times \omega(\mathbf{t}_2)$$

Theorem (Normal terms in *probabilistic* have weight 1)

$\Gamma_{\mathcal{C}} \vdash \boldsymbol{t} : C \Rightarrow \omega(\boldsymbol{t}\downarrow) = 1.$

Proof. We prove that $\Gamma \vdash \mathbf{t} : \alpha.C \Rightarrow \omega(\mathbf{t}\downarrow) = \alpha$ by structural induction over $\mathbf{t}\downarrow$.

Example: $2.(\lambda x\ \frac{1}{2}.x)\ y \to y$

$$\omega(2.(\lambda x\ \tfrac{1}{2}.x)\ y) = 2 \qquad \omega(y) = 1$$

The *scalar* type system
Logical content: No-cloning theorem (I)

Definition (Proof method of depth n)

$$\Pi_0(S) = S$$
$$\Pi_n(S) = \frac{\Pi_{n-1}(S)}{P_S}\, R \quad \text{or} \quad \frac{\Pi_k(S) \quad \pi_h}{P_S}\, R \quad \text{or} \quad \frac{\pi_k \quad \Pi_h(S)}{P_S}\, R$$

where

- S is a sequent,
- π_n is a constant derivation tree of size n,
- $\max\{k, h\} = n - 1$,
- R is a typing rule, and
- P_S is a sequent such that the resulting derivation tree is *well-formed*.

The *scalar* type system

Logical content: No-cloning theorem (I)

Definition (Proof method of depth n)

$$\Pi_0(S) = S$$

$$\Pi_n(S) = \frac{\Pi_{n-1}(S)}{P_S}\,R \quad \text{or} \quad \frac{\Pi_k(S) \quad \pi_h}{P_S}\,R \quad \text{or} \quad \frac{\pi_k \quad \Pi_h(S)}{P_S}\,R$$

where

- S is a sequent,
- π_n is a constant derivation tree of size n,
- $\max\{k, h\} = n - 1$,
- R is a typing rule, and
- P_S is a sequent such that the resulting derivation tree is *well-formed*.

$C(\Pi_n(S))$ denote the conclusion (root) of the tree $\Pi_n(S)$.

The *scalar* type system
Logical content: No-cloning theorem (II)

Examples:

$$\Pi_1(S) = \frac{S}{P_S}\,\forall I$$

The *scalar* type system

Logical content: No-cloning theorem (II)

Examples:

$$\Pi_1(S) = \frac{S}{P_S}\,\forall I \qquad \Pi_1(\Gamma \vdash \mathbf{t} : T) = \frac{\Gamma \vdash \mathbf{t} : T}{\Gamma \vdash \mathbf{t} : \forall X.T}\,\forall I$$

The *scalar* type system

Logical content: No-cloning theorem (II)

Examples:

$$\Pi_1(S) = \frac{S}{P_S}\,\forall I \qquad \Pi_1(\Gamma \vdash \mathbf{t} : T) = \frac{\Gamma \vdash \mathbf{t} : T}{\Gamma \vdash \mathbf{t} : \forall X.T}\,\forall I$$

$$\Pi_n(\Gamma \vdash \mathbf{u} : \alpha.A) = \frac{\Gamma \vdash \mathbf{u} : \alpha.A \qquad \dfrac{\pi_{n-2}}{\Gamma \vdash \mathbf{v} : 2.A}}{\Gamma \vdash \mathbf{u} + \mathbf{v} : (2+\alpha).A}\,+I$$

(Partial function or pattern matching)

The *scalar* type system

Logical content: No-cloning theorem (II)

Examples:

$$\Pi_1(S) = \frac{S}{P_S}\,\forall I \qquad \Pi_1(\Gamma \vdash \mathbf{t} : T) = \frac{\Gamma \vdash \mathbf{t} : T}{\Gamma \vdash \mathbf{t} : \forall X.T}\,\forall I$$

$$\Pi_n(\Gamma \vdash \mathbf{u} : \alpha.A) = \frac{\Gamma \vdash \mathbf{u} : \alpha.A \qquad \dfrac{\pi_{n-2}}{\Gamma \vdash \mathbf{v} : 2.A}}{\Gamma \vdash \mathbf{u} + \mathbf{v} : (2+\alpha).A}\,+I$$

(Partial function or pattern matching) Basically $C(\Pi_n(S)) = P_S$ means that P_S can be derived from S by using S once, with the fixed proof method Π.

The *scalar* type system

Logical content: No-cloning theorem (III)

Theorem (No-cloning of scalars)

$\nexists\Pi_n$ such that $\forall\alpha, C(\Pi_n(\Gamma \vdash \alpha.U)) = \Delta \vdash (\delta \times \alpha^s + \gamma).V$ with $\delta \neq 0$ and γ constants in $\mathcal{S}$, $s \in \mathbb{N}^{>1}$ and U, V constants in $\mathcal{U}$.

Proof. Induction over n.

The *scalar* type system

Logical content: No-cloning theorem (III)

Theorem (No-cloning of scalars)

$\nexists \Pi_n$ *such that* $\forall\alpha, C(\Pi_n(\Gamma \vdash \alpha.U)) = \Delta \vdash (\delta \times \alpha^s + \gamma).V$ *with* $\delta \neq 0$ *and* γ *constants in* $\mathcal{S}$, $s \in \mathbb{N}^{>1}$ *and* U, V *constants in* $\mathcal{U}$.

Proof. Induction over n.

Corollary (No-cloning Theorem)

$\nexists \Pi_n$ *such that* $\forall T, C(\Pi_n(\Gamma \vdash T)) = \Delta \vdash T \otimes T$.

where $A \otimes B$ is the classical encoding for the type of tuples.

The *scalar* type system

Logical content: No-cloning theorem (III)

Theorem (No-cloning of scalars)

$\nexists \Pi_n$ such that $\forall \alpha, C(\Pi_n(\Gamma \vdash \alpha.U)) = \Delta \vdash (\delta \times \alpha^s + \gamma).V$ with $\delta \neq 0$ and γ constants in $\mathcal{S}$, $s \in \mathbb{N}^{>1}$ and U, V constants in $\mathcal{U}$.

Proof. Induction over n.

Corollary (No-cloning Theorem)

$\nexists \Pi_n$ such that $\forall T, C(\Pi_n(\Gamma \vdash T)) = \Delta \vdash T \otimes T$.

where $A \otimes B$ is the classical encoding for the type of tuples.
Proof.

- It is easy to show that $\forall T, T \equiv \alpha.U$ with $U \in \mathcal{U}$

The *scalar* type system

Logical content: No-cloning theorem (III)

Theorem (No-cloning of scalars)

$\nexists \Pi_n$ such that $\forall \alpha, C(\Pi_n(\Gamma \vdash \alpha.U)) = \Delta \vdash (\delta \times \alpha^s + \gamma).V$ with $\delta \neq 0$ and γ constants in $\mathcal{S}$, $s \in \mathbb{N}^{>1}$ and U, V constants in $\mathcal{U}$.

Proof. Induction over n.

Corollary (No-cloning Theorem)

$\nexists \Pi_n$ such that $\forall T, C(\Pi_n(\Gamma \vdash T)) = \Delta \vdash T \otimes T$.

where $A \otimes B$ is the classical encoding for the type of tuples.
Proof.

- It is easy to show that $\forall T, T \equiv \alpha.U$ with $U \in \mathcal{U}$
- $T \otimes T \equiv \alpha.U \otimes \alpha.U \equiv \alpha^2.(U \otimes U) = (1 \times \alpha^2 + 0).(U \otimes U)$.

The *scalar* type system

Logical content: No-cloning theorem (III)

Theorem (No-cloning of scalars)

$\nexists \Pi_n$ such that $\forall \alpha, C(\Pi_n(\Gamma \vdash \alpha.U)) = \Delta \vdash (\delta \times \alpha^s + \gamma).V$ with $\delta \neq 0$ and γ constants in $\mathcal{S}$, $s \in \mathbb{N}^{>1}$ and U, V constants in $\mathcal{U}$.

Proof. Induction over n.

Corollary (No-cloning Theorem)

$\nexists \Pi_n$ such that $\forall T, C(\Pi_n(\Gamma \vdash T)) = \Delta \vdash T \otimes T$.

where $A \otimes B$ is the classical encoding for the type of tuples.
Proof.

- It is easy to show that $\forall T, T \equiv \alpha.U$ with $U \in \mathcal{U}$
- $T \otimes T \equiv \alpha.U \otimes \alpha.U \equiv \alpha^2.(U \otimes U) = (1 \times \alpha^2 + 0).(U \otimes U)$.
- By the previous theorem, the corollary holds.

The *scalar* type system
Summary of contributions

- S.N. : Simplified the Linear-algebraic λ-calculus by lifting most restrictions. S.R. OK.

The *scalar* type system
Summary of contributions

- S.N. : Simplified the Linear-algebraic λ-calculus by lifting most restrictions. S.R. OK.
- *Scalar* type system : types keep track of the 'amount of a type' by holding sum of amplitudes of terms of that types.

The *scalar* type system

Summary of contributions

- S.N. : Simplified the Linear-algebraic λ-calculus by lifting most restrictions. S.R. OK.
- *Scalar* type system : types keep track of the 'amount of a type' by holding sum of amplitudes of terms of that types.
- $\Longrightarrow$ probabilistic type system, yielding a higher-order probabilistic λ-calculus.

The *scalar* type system
Polemics, future work

The *scalar* type system
Polemics, future work

- Captured no-cloning theorem is a way that is faithful to quantum theory and linear algebra, unlike LL:
  - For all A one can find a copying *proof method*, but there is no *proof method* for cloning all A.
  - Algebraic linearity is about taking $\alpha.U$ to something in $\alpha\gamma + \delta \ldots$ not just for $\gamma = 1 = \delta + 1$.

The *scalar* type system

Polemics, future work

- Captured no-cloning theorem is a way that is faithful to quantum theory and linear algebra, unlike LL:
  - For all A one can find a copying *proof method*, but there is no *proof method* for cloning all A.
  - Algebraic linearity is about taking $\alpha.U$ to something in $\alpha\gamma + \delta \ldots$ not just for $\gamma = 1 = \delta + 1$.
- C.-H.+Q.C.=(quantum th. proofs, quantum th. logics)? Need a Vectorial type system:
  - Scalar type system $\rightarrow$ magnitude and signs for type vectors.
  - Future system $\rightarrow$ direction, (*i.e.* addition and orthogonality of types). Then it would be possible to check norm on amplitudes rather than probabilities.