Scalar System F for Linear-Algebraic λ-Calculus

Towards a Quantum Physical Logic

Pablo Arrighi and Alejandro Díaz-Caro

{pablo.arrighi,alejandro.diaz-caro}@imag.fr
Université de Grenoble, LIG

April 9th, 2009. VI Quantum Physics and Logic. Oxford University

Motivation

- Quantum Logic[1] was 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.

Motivation

- Quantum Logic[1] was developed *ad hoc* before quantum computing (no clear relation with quantum programs).
- There is a need for a logic that could aid us to isolating the reasoning behind some quantum algorithms.

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

Motivation

- Quantum Logic[1] was developed *ad hoc* before quantum computing (no clear relation with quantum programs).
- There is a need for a logic that could aid us to isolating the reasoning behind some quantum algorithms.
- Usually the reasoning behind a program can be made to arise via a formally-defined logic from the study of type systems (Curry-Howard correspondence).

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

Motivation

- Quantum Logic[1] was developed *ad hoc* before quantum computing (no clear relation with quantum programs).
- There is a need for a logic that could aid us to isolating the reasoning behind some quantum algorithms.
- Usually the reasoning behind a program can be made to arise via a formally-defined logic from the study of type systems (Curry-Howard correspondence).

This work is a first step towards a formally-defined quantum physical logic arising via the Curry-Howard correspondence

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

Linear-Algebraic λ-Calculus[2]

Higher-order computation

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

[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]

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]

Higher-order computation | **Linear algebra**

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

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

(*) **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]

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

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

Linear-Algebraic λ-Calculus

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

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

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

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

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.

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

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

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

System F rules plus simple rules to type algebraic terms

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

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. Extension of Barendregt's proof (Barendregt, H.P., "Lambda calculi with types", Handbook of Logic in Computer Science **2**, Clarendon Press, Oxford, 1992).

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

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

Higher-order computation

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

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

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

Every typable term is strong normalizing

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

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.
Every typable term is strong normalizing
Hence **Yb** is no typable!

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 $\mathbf{u}\;(\mathbf{v} + \mathbf{w}) \to (\mathbf{u}\;\mathbf{v}) + (\mathbf{u}\;\mathbf{w})$.

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.
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 (I)

Types grammar:

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

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

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

The *scalar* type system (II)

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

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

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

Where $U \in \mathcal{U}$.

The *scalar* type system (II)

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

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

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

Where $U \in \mathcal{U}$.

The *scalar* type system (II)

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

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

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

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

Where $U \in \mathcal{U}$.

The *scalar* type system (II)

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

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

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

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

Where $U \in \mathcal{U}$.

Strong normalisation

Let $(\cdot)^\natural$ be a map from $\mathcal{T} \setminus \{\perp\}$ to $\mathbb{T}(\lambda 2^{la})$.

Strong normalisation

Let $(\cdot)^\natural$ be a map from $\mathcal{T} \setminus \{\bot\}$ to $\mathbb{T}(\lambda 2^{la})$.
Also, let use the following notation:
$\Gamma^\natural = \{(x : T^\natural) \mid (x : T) \in \Gamma\}$
$\bot^\natural = T$ for whatever type $T \in \mathbb{T}(\lambda 2^{la})$.

Strong normalisation

Let $(\cdot)^\natural$ be a map from $\mathcal{T} \setminus \{\perp\}$ to $\mathbb{T}(\lambda 2^{la})$.
Also, let use the following notation:
$\Gamma^\natural = \{(x : T^\natural) \mid (x : T) \in \Gamma\}$
$\perp^\natural = T$ for whatever type $T \in \mathbb{T}(\lambda 2^{la})$.

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

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

Strong normalisation

Let $(\cdot)^\natural$ be a map from $\mathcal{T} \setminus \{\perp\}$ to $\mathbb{T}(\lambda 2^{la})$.
Also, let use the following notation:
$\Gamma^\natural = \{(x : T^\natural) \mid (x : T) \in \Gamma\}$
$\perp^\natural = T$ for whatever type $T \in \mathbb{T}(\lambda 2^{la})$.

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.

Subject reduction

Theorem (Subject Reduction)

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

Subject reduction

Theorem (Subject Reduction)

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

Proof. There are 27 auxiliary lemmas to make the proof.

Probabilistic type system

Conditional functions $\longrightarrow$ same type on each branch.

[2] Di Pierro, A., C. Hanking and H. Wiklicky, *Probabilistic λ-calculus and quantitative program analysis*, Journal of Logic and Computation **15** (2005), pp. 159–179.

Probabilistic type system

Conditional functions $\longrightarrow$ same type on each branch.
By restricting the scalars to positive reals $\longrightarrow$ probabilistic type system.[2]

[2] Di Pierro, A., C. Hanking and H. Wiklicky, *Probabilistic λ-calculus and quantitative program analysis*, Journal of Logic and Computation **15** (2005), pp. 159–179.

Probabilistic type system

Conditional functions → same type on each branch.

By restricting the scalars to positive reals → probabilistic type system.[2]

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.

[2] Di Pierro, A., C. Hanking and H. Wiklicky, *Probabilistic λ-calculus and quantitative program analysis*, Journal of Logic and Computation **15** (2005), pp. 159–179.

Logical content: No-cloning theorem

Let Π a tree of typing rules and think of Π as a function from lists of sequents to proofs

Logical content: No-cloning theorem

Let Π a tree of typing rules and think of Π as a function from lists of sequents to proofs

Theorem (No-cloning)

$\nexists\Pi$ such that $\forall A, \Pi(\Gamma \vdash A)$ has as conclusion $\Delta \vdash A \otimes A$.

Remark: Think of Π as a "universal machine". Then this theorem says "there is not a universal clonning machine".

Sumary of conclusions and future work

- *Scalar* type system $\longrightarrow$ probabilistic type system guaranteeing probabilistic functions to be well defined.

Sumary of conclusions and future work

- *Scalar* type system $\longrightarrow$ probabilistic type system guaranteeing probabilistic functions to be well defined.
- Strong normalization theorem $\longrightarrow$ most restrictions can be lifted in the reduction rules.

Sumary of conclusions and future work

- *Scalar* type system $\rightarrow$ probabilistic type system guaranteeing probabilistic functions to be well defined.
- Strong normalization theorem $\rightarrow$ most restrictions can be lifted in the reduction rules.
- No-cloning theorem $\rightarrow$ thinking in terms of *machines* rather than linear-logic *resources*.

Sumary of conclusions and future work

- *Scalar* type system $\longrightarrow$ probabilistic type system guaranteeing probabilistic functions to be well defined.
- Strong normalization theorem $\longrightarrow$ most restrictions can be lifted in the reduction rules.
- No-cloning theorem $\longrightarrow$ thinking in terms of *machines* rather than linear-logic *resources*.
- This is the first step towards a future vectorial type system.
  - Scalar type system $\longrightarrow$ magnitude and signs for type vectors.
  - Future system $\longrightarrow$ direction, (*i.e.* addition and orthogonality of types).
    Then it would be possible to use amplituds rather than probabilities.