The λ-calculus: from simple types to non-idempotent intersection types

Day 2: The simply typed λ-calculus and the Curry-Howard correspondence

Giulio Guerrieri

Department of Informatics, University of Sussex (Brighton, UK)
✉ g.guerrieri@sussex.ac.uk https://pageperso.lis-lab.fr/~giulio.guerrieri/

37th Escuela de Ciencias Informaticas (ECI 2024)
Buenos Aires (Argentina), 30 July 2024

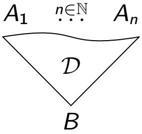
Outline

Outline

1. The Curry-Howard correspondence and the simply typed λ-calculus
2. Strong normalization of the simply typed λ-calculus
3. Conclusion, exercises and bibliography

A computational interpretation of ND

Idea. A derivation $\mathcal{D} \triangleright_{\mathrm{ND}} A_1, \ldots, A_n \vdash B$ can be seen as a function $t(x_1, \ldots, x_n)$



that associates with derivations
$\mathcal{D}_1 \triangleright_{\mathrm{ND}} \vdash A_1$, ..., $\mathcal{D}_n \triangleright_{\mathrm{ND}} \vdash A_n$,
a derivation $t(\mathcal{D}_1/x_1, \ldots, \mathcal{D}_n/x_n) \triangleright_{\mathrm{ND}} \vdash B$.

$\rightsquigarrow$ Let us see how, by induction on $\mathcal{D}$.

- A derivation consisting of a single hypothesis A is represented by a variable x. Different formulas are associated with different variables. For several occurrences of A as hypotheses, we chose the same x or another variable. $\rightsquigarrow$ A variable represents a (possibly empty) parcel of hypotheses of the same formula.
- If $\mathcal{D}$ ends in $\Rightarrow_i$ let $s(y, x_1, \ldots, x_n)$ be the function associated with the $\Rightarrow_i$-premise. Let x be the variable associated with the parcel of hypotheses C discharged by $\Rightarrow_i$. The function $t(x_1, \ldots, x_n)$ associated with $\mathcal{D}$ maps $\mathcal{D}' \triangleright_{\mathrm{ND}} \vdash C$ to $s(\mathcal{D}'/y, x_1, \ldots, x_n)$.

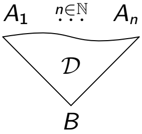
  (abstraction) $\quad t(x_1, \ldots, x_n) \coloneqq \lambda y.s(y, x_1, \ldots, x_n) \quad$ (i.e. $y \mapsto s(y, x_1, \ldots, x_n)$)

- If $\mathcal{D}$ ends in $\Rightarrow_e$, let $s_1(x_1, \ldots, x_n)$ and $s_2(x_1, \ldots, x_n)$ be the functions associated with the two premises of $\Rightarrow_e$. The function $t(x_1, \ldots, x_n)$ associated with $\mathcal{D}$ is the application (noted as juxtaposition) of $s_1(x_1, \ldots, x_n)$ to $s_2(x_1, \ldots, x_n)$.

  (application) $\quad t(x_1, \ldots, x_n) \coloneqq s_1(x_1, \ldots, x_n)s_2(x_1, \ldots, x_n)$

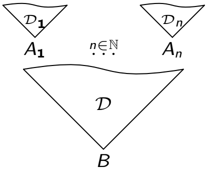
A computational interpretation of ND

Idea. A derivation $\mathcal{D} \triangleright_{\mathsf{ND}} A_1, \ldots, A_n \vdash B$ can be seen as a function $t(x_1, \ldots, x_n)$



that associates with derivations
$\mathcal{D}_1 \triangleright_{\mathsf{ND}} \vdash A_1$, ..., $\mathcal{D}_n \triangleright_{\mathsf{ND}} \vdash A_n$,
a derivation $t(\mathcal{D}_1/x_1, \ldots, \mathcal{D}_n/x_n) \triangleright_{\mathsf{ND}} \vdash B$.

$\rightsquigarrow$ Let us see how, by induction on $\mathcal{D}$.



- A derivation consisting of a single hypothesis A is represented by a variable x.
  Different formulas are associated with different variables.
  For several occurrences of A as hypotheses, we chose the same x or another variable.
  $\rightsquigarrow$ A variable represents a (possibly empty) parcel of hypotheses of the same formula.
- If $\mathcal{D}$ ends in $\Rightarrow_i$ let $s(y, x_1, \ldots, x_n)$ be the function associated with the $\Rightarrow_i$-premise.
  Let x be the variable associated with the parcel of hypotheses C discharged by $\Rightarrow_i$.
  The function $t(x_1, \ldots, x_n)$ associated with $\mathcal{D}$ maps $\mathcal{D}' \triangleright_{\mathsf{ND}} \vdash C$ to $s(\mathcal{D}'/y, x_1, \ldots, x_n)$.

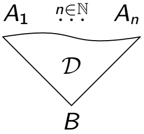
  (abstraction) $\quad t(x_1, \ldots, x_n) \coloneqq \lambda y.s(y, x_1, \ldots, x_n) \quad$ (i.e. $y \mapsto s(y, x_1, \ldots, x_n)$)

- If $\mathcal{D}$ ends in $\Rightarrow_e$, let $s_1(x_1, \ldots, x_n)$ and $s_2(x_1, \ldots, x_n)$ be the functions associated with the two premises of $\Rightarrow_e$. The function $t(x_1, \ldots, x_n)$ associated with $\mathcal{D}$ is the application (noted as juxtaposition) of $s_1(x_1, \ldots, x_n)$ to $s_2(x_1, \ldots, x_n)$.

  (application) $\quad t(x_1, \ldots, x_n) \coloneqq s_1(x_1, \ldots, x_n)s_2(x_1, \ldots, x_n)$

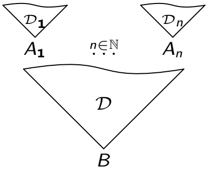
A computational interpretation of ND

Idea. A derivation $\mathcal{D} \rhd_{\mathsf{ND}} A_1, \dots, A_n \vdash B$ can be seen as a function $t(x_1, \dots, x_n)$



that associates with derivations
$\mathcal{D}_1 \rhd_{\mathsf{ND}} \vdash A_1$, ..., $\mathcal{D}_n \rhd_{\mathsf{ND}} \vdash A_n$,
a derivation $t(\mathcal{D}_1/x_1, \dots, \mathcal{D}_n/x_n) \rhd_{\mathsf{ND}} \vdash B$.

$\rightsquigarrow$ Let us see how, by induction on $\mathcal{D}$.



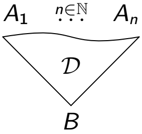
- A derivation consisting of a single hypothesis A is represented by a variable x.
  Different formulas are associated with different variables.
  For several occurrences of A as hypotheses, we chose the same x or another variable.
  $\rightsquigarrow$ A variable represents a (possibly empty) parcel of hypotheses of the same formula.
- If $\mathcal{D}$ ends in $\Rightarrow_i$ let $s(y, x_1, \dots, x_n)$ be the function associated with the $\Rightarrow_i$-premise.
  Let x be the variable associated with the parcel of hypotheses C discharged by $\Rightarrow_i$.
  The function $t(x_1, \dots, x_n)$ associated with $\mathcal{D}$ maps $\mathcal{D}' \rhd_{\mathsf{ND}} \vdash C$ to $s(\mathcal{D}'/y, x_1, \dots, x_n)$.

  (abstraction) $\quad t(x_1, \dots, x_n) \coloneqq \lambda y.s(y, x_1, \dots, x_n) \quad$ (i.e. $y \mapsto s(y, x_1, \dots, x_n)$)
- If $\mathcal{D}$ ends in $\Rightarrow_e$, let $s_1(x_1, \dots, x_n)$ and $s_2(x_1, \dots, x_n)$ be the functions associated with the two premises of $\Rightarrow_e$. The function $t(x_1, \dots, x_n)$ associated with $\mathcal{D}$ is the application (noted as juxtaposition) of $s_1(x_1, \dots, x_n)$ to $s_2(x_1, \dots, x_n)$.

  (application) $\quad t(x_1, \dots, x_n) \coloneqq s_1(x_1, \dots, x_n) s_2(x_1, \dots, x_n)$

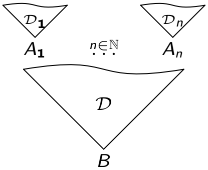
A computational interpretation of ND

Idea. A derivation $\mathcal{D} \rhd_{\mathsf{ND}} A_1, \dots, A_n \vdash B$ can be seen as a function $t(x_1, \dots, x_n)$



that associates with derivations
$\mathcal{D}_1 \rhd_{\mathsf{ND}} \vdash A_1,\ \dots,\ \mathcal{D}_n \rhd_{\mathsf{ND}} \vdash A_n$,
a derivation $t(\mathcal{D}_1/x_1, \dots, \mathcal{D}_n/x_n) \rhd_{\mathsf{ND}} \vdash B$.

⇝ Let us see how, by induction on $\mathcal{D}$.



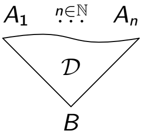
- A derivation consisting of a single hypothesis A is represented by a variable x.
  Different formulas are associated with different variables.
  For several occurrences of A as hypotheses, we chose the same x or another variable.
  ⇝ A variable represents a (possibly empty) parcel of hypotheses of the same formula.
- If $\mathcal{D}$ ends in $\Rightarrow_i$ let $s(y, x_1, \dots, x_n)$ be the function associated with the $\Rightarrow_i$-premise.
  Let x be the variable associated with the parcel of hypotheses C discharged by $\Rightarrow_i$.
  The function $t(x_1, \dots, x_n)$ associated with $\mathcal{D}$ maps $\mathcal{D}' \rhd_{\mathsf{ND}} \vdash C$ to $s(\mathcal{D}'/y, x_1, \dots, x_n)$.

  (abstraction) $\quad t(x_1, \dots, x_n) \coloneqq \lambda y.s(y, x_1, \dots, x_n) \quad$ (i.e. $y \mapsto s(y, x_1, \dots, x_n)$)
- If $\mathcal{D}$ ends in $\Rightarrow_e$, let $s_1(x_1, \dots, x_n)$ and $s_2(x_1, \dots, x_n)$ be the functions associated with the two premises of $\Rightarrow_e$. The function $t(x_1, \dots, x_n)$ associated with $\mathcal{D}$ is the application (noted as juxtaposition) of $s_1(x_1, \dots, x_n)$ to $s_2(x_1, \dots, x_n)$.

  (application) $\quad t(x_1, \dots, x_n) \coloneqq s_1(x_1, \dots, x_n)s_2(x_1, \dots, x_n)$

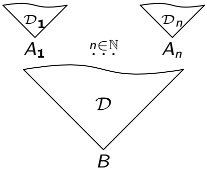
A computational interpretation of ND

Idea. A derivation $\mathcal{D} \triangleright_{\mathrm{ND}} A_1, \dots, A_n \vdash B$ can be seen as a function $t(x_1, \dots, x_n)$



that associates with derivations
$\mathcal{D}_1 \triangleright_{\mathrm{ND}} \vdash A_1, \dots, \mathcal{D}_n \triangleright_{\mathrm{ND}} \vdash A_n$,
a derivation $t(\mathcal{D}_1/x_1, \dots, \mathcal{D}_n/x_n) \triangleright_{\mathrm{ND}} \vdash B$.

$\rightsquigarrow$ Let us see how, by induction on $\mathcal{D}$.



- A derivation consisting of a single hypothesis A is represented by a variable x.
  Different formulas are associated with different variables.
  For several occurrences of A as hypotheses, we chose the same x or another variable.
  $\rightsquigarrow$ A variable represents a (possibly empty) parcel of hypotheses of the same formula.
- If $\mathcal{D}$ ends in $\Rightarrow_i$ let $s(y, x_1, \dots, x_n)$ be the function associated with the $\Rightarrow_i$-premise.
  Let x be the variable associated with the parcel of hypotheses C discharged by $\Rightarrow_i$.
  The function $t(x_1, \dots, x_n)$ associated with $\mathcal{D}$ maps $\mathcal{D}' \triangleright_{\mathrm{ND}} \vdash C$ to $s(\mathcal{D}'/y, x_1, \dots, x_n)$.

  $$(\text{abstraction}) \quad t(x_1, \dots, x_n) \coloneqq \lambda y.s(y, x_1, \dots, x_n) \quad (\text{i.e. } y \mapsto s(y, x_1, \dots, x_n))$$

- If $\mathcal{D}$ ends in $\Rightarrow_e$, let $s_1(x_1, \dots, x_n)$ and $s_2(x_1, \dots, x_n)$ be the functions associated with the two premises of $\Rightarrow_e$. The function $t(x_1, \dots, x_n)$ associated with $\mathcal{D}$ is the application (noted as juxtaposition) of $s_1(x_1, \dots, x_n)$ to $s_2(x_1, \dots, x_n)$.

  $$(\text{application}) \quad t(x_1, \dots, x_n) \coloneqq s_1(x_1, \dots, x_n)s_2(x_1, \dots, x_n)$$

Cut-elimination as a computational step

$$
\dfrac{\dfrac{\begin{array}{c}[A]^y \\ \vdots \\ t(y,\vec{x}) : B\end{array}}{\lambda y.t(y,\vec{x}) : A \Rightarrow B}\Rightarrow_i^y \qquad \begin{array}{c}\vdots \\ s(\vec{x}) : A\end{array}}{(\lambda y.t(y,\vec{x}))s(\vec{x}) : B}\Rightarrow_e
\quad \longrightarrow_{\textbf{cut}} \quad
\begin{array}{c}\vdots \\ s(\vec{x}) : A \\ \vdots \\ t(s(\vec{x})/y,\vec{x}) : B\end{array}
$$

- We can decorate each formula occurrence in a derivation with a term.
  $\rightsquigarrow$ For every derivation $\mathcal{D}$, its term $(\mathcal{D})_\lambda$ is the decoration of its conclusion.
- This decoration commute with cut-elimination via the step:

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

where $t\{s/x\}$ stands for the substitution of s for the free occurrences of x in t.

$$
\begin{array}{ccc}
\mathcal{D} & \xrightarrow{\quad cut \quad} & \mathcal{D}' \\
\text{decoration} \rightsquigarrow & & \rightsquigarrow \text{decoration} \\
\mathcal{D}_\lambda & \xrightarrow{\quad \beta \quad} & (\mathcal{D}_\lambda)' = (\mathcal{D}')_\lambda
\end{array}
$$

Cut-elimination as a computational step

$$\dfrac{\dfrac{\begin{matrix}[A]^y\\ \vdots \\ t(y,\vec{x}) : B\end{matrix}}{\lambda y.t(y,\vec{x}) : A \Rightarrow B}\Rightarrow_i^y \qquad \begin{matrix}\vdots\\ s(\vec{x}) : A\end{matrix}}{(\lambda y.t(y,\vec{x}))s(\vec{x}) : B}\Rightarrow_e \qquad \to_{\text{cut}} \qquad \begin{matrix}\vdots\\ s(\vec{x}) : A\\ \vdots\\ t(s(\vec{x})/y,\vec{x}) : B\end{matrix}$$

- We can decorate each formula occurrence in a derivation with a term.
  ⇝ For every derivation $\mathcal{D}$, its term $(\mathcal{D})_\lambda$ is the decoration of its conclusion.
- This decoration commute with cut-elimination via the step:

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

where $t\{s/x\}$ stands for the substitution of s for the free occurrences of x in t.

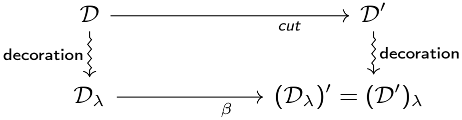
Cut-elimination as a computational step

$$\dfrac{\dfrac{\begin{array}{c}[A]^y \\ \vdots \\ t(y,\vec{x}) : B\end{array}}{\lambda y.t(y,\vec{x}) : A \Rightarrow B}\Rightarrow_i^y \qquad \begin{array}{c}\vdots \\ s(\vec{x}) : A\end{array}}{(\lambda y.t(y,\vec{x}))s(\vec{x}) : B}\Rightarrow_e \qquad \to_{\text{cut}} \qquad \begin{array}{c}\vdots \\ s(\vec{x}) : A \\ \vdots \\ t(s(\vec{x})/y,\vec{x}) : B\end{array}$$

- We can decorate each formula occurrence in a derivation with a term.
  $\rightsquigarrow$ For every derivation $\mathcal{D}$, its term $(\mathcal{D})_\lambda$ is the decoration of its conclusion.
- This decoration commute with cut-elimination via the step:

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

where $t\{s/x\}$ stands for the substitution of s for the free occurrences of x in t.



Examples of decorations of derivations in ND

$$x : A \qquad\qquad \dfrac{[x : A]^x}{\lambda x.x : A \Rightarrow A}\Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : A \Rightarrow A}\Rightarrow_i}{\lambda x.\lambda y.x : A \Rightarrow A \Rightarrow A}\Rightarrow_i^x \qquad \dfrac{\dfrac{[y : A]^y}{\lambda y.y : A \Rightarrow A}\Rightarrow_i^y}{\lambda x.\lambda y.y : A \Rightarrow A \Rightarrow A}\Rightarrow_i^x$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \Rightarrow (B \Rightarrow C)]^x \quad [z : A]^z}{xz : B \Rightarrow C}\Rightarrow_e \quad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B}\Rightarrow_e}{(xz)(yz) : C}\Rightarrow_e}{\lambda z.(xz)(yz) : A \Rightarrow C}\Rightarrow_i^z}{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow C)}\Rightarrow_i^y}{\lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)}\Rightarrow_i^x$$

Examples of decorations of derivations in ND

$$x : A \qquad\qquad \dfrac{[x : A]^x}{\lambda x.x : A \Rightarrow A}\Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : A \Rightarrow A}\Rightarrow_i}{\lambda x.\lambda y.x : A \Rightarrow A \Rightarrow A}\Rightarrow_i^x \qquad \dfrac{\dfrac{[y : A]^y}{\lambda y.y : A \Rightarrow A}\Rightarrow_i^y}{\lambda x.\lambda y.y : A \Rightarrow A \Rightarrow A}\Rightarrow_i^x$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \Rightarrow (B \Rightarrow C)]^x \quad [z : A]^z}{xz : B \Rightarrow C}\Rightarrow_e \qquad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B}\Rightarrow_e}{(xz)(yz) : C}\Rightarrow_e}{\lambda z.(xz)(yz) : A \Rightarrow C}\Rightarrow_i^z}{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow C)}\Rightarrow_i^y}{\lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)}\Rightarrow_i^x$$

Examples of decorations of derivations in ND

$$x : A \qquad\qquad \dfrac{[x : A]^x}{\lambda x.x : A \Rightarrow A}\Rightarrow_i^x \qquad\qquad \dfrac{\dfrac{[x : A]^x}{\lambda y.x : B \Rightarrow A}\Rightarrow_i^y}{\lambda x.\lambda y.x : A \Rightarrow B \Rightarrow A}\Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : A \Rightarrow A}\Rightarrow_i}{\lambda x.\lambda y.x : A \Rightarrow A \Rightarrow A}\Rightarrow_i^x \qquad \dfrac{\dfrac{[y : A]^y}{\lambda y.y : A \Rightarrow A}\Rightarrow_i^y}{\lambda x.\lambda y.y : A \Rightarrow A \Rightarrow A}\Rightarrow_i^x$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \Rightarrow (B \Rightarrow C)]^x \quad [z : A]^z}{xz : B \Rightarrow C}\Rightarrow_e \qquad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B}\Rightarrow_e}{(xz)(yz) : C}\Rightarrow_e}{\lambda z.(xz)(yz) : A \Rightarrow C}\Rightarrow_i^z}{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow C)}\Rightarrow_i^y}{\lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)}\Rightarrow_i^x$$

Examples of decorations of derivations in ND

$$x : A$$

$$\dfrac{[x : A]^x}{\lambda x.x : A \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : B \Rightarrow A} \Rightarrow_i^y}{\lambda x.\lambda y.x : A \Rightarrow B \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : A \Rightarrow A} \Rightarrow_i}{\lambda x.\lambda y.x : A \Rightarrow A \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{[y : A]^y}{\lambda y.y : A \Rightarrow A} \Rightarrow_i^y}{\lambda x.\lambda y.y : A \Rightarrow A \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \Rightarrow (B \Rightarrow C)]^x \quad [z : A]^z}{xz : B \Rightarrow C} \Rightarrow_e \quad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B} \Rightarrow_e}{(xz)(yz) : C} \Rightarrow_e}{\lambda z.(xz)(yz) : A \Rightarrow C} \Rightarrow_i^z}{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow C)} \Rightarrow_i^y}{\lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)} \Rightarrow_i^x$$

Examples of decorations of derivations in ND

$$x : A$$

$$\dfrac{[x : A]^x}{\lambda x.x : A \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : B \Rightarrow A} \Rightarrow_i^y}{\lambda x.\lambda y.x : A \Rightarrow B \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : A \Rightarrow A} \Rightarrow_i}{\lambda x.\lambda y.x : A \Rightarrow A \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{[y : A]^y}{\lambda y.y : A \Rightarrow A} \Rightarrow_i^y}{\lambda x.\lambda y.y : A \Rightarrow A \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \Rightarrow (B \Rightarrow C)]^x \quad [z : A]^z}{xz : B \Rightarrow C} \Rightarrow_e \quad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B} \Rightarrow_e}{(xz)(yz) : C} \Rightarrow_e}{\lambda z.(xz)(yz) : A \Rightarrow C} \Rightarrow_i^z}{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow C)} \Rightarrow_i^y}{\lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)} \Rightarrow_i^x$$

Examples of decorations of derivations in ND

$$x : A \qquad \dfrac{[x : A]^x}{\lambda x.x : A \Rightarrow A} \Rightarrow_i^x \qquad \dfrac{\dfrac{[x : A]^x}{\lambda y.x : B \Rightarrow A} \Rightarrow_i^y}{\lambda x.\lambda y.x : A \Rightarrow B \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{[x : A]^x}{\lambda y.x : A \Rightarrow A} \Rightarrow_i}{\lambda x.\lambda y.x : A \Rightarrow A \Rightarrow A} \Rightarrow_i^x \qquad \dfrac{\dfrac{[y : A]^y}{\lambda y.y : A \Rightarrow A} \Rightarrow_i^y}{\lambda x.\lambda y.y : A \Rightarrow A \Rightarrow A} \Rightarrow_i^x$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \Rightarrow (B \Rightarrow C)]^x \quad [z : A]^z}{xz : B \Rightarrow C} \Rightarrow_e \quad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B} \Rightarrow_e}{(xz)(yz) : C} \Rightarrow_e}{\lambda z.(xz)(yz) : A \Rightarrow C} \Rightarrow_i^z}{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow C)} \Rightarrow_i^y}{\lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)} \Rightarrow_i^x$$

Example of decorations of derivations in ND with cut-elimination

$$
\dfrac{\dfrac{x : [X \Rightarrow X]^x}{\lambda x.x : (X \Rightarrow X) \Rightarrow X \Rightarrow X}\Rightarrow_i^x \qquad \dfrac{y : [X]^y}{\lambda y.y : X \Rightarrow X}\Rightarrow_i^y}{(\lambda x.x)\lambda y.y : X \Rightarrow X}\Rightarrow_e
\quad \to_{\text{cut}} \quad
\dfrac{y : [X]^x}{\lambda y.y : X \Rightarrow X}\Rightarrow_i^x
$$

Rmk. $(\lambda x.x)\lambda y.y \to_\beta x\{\lambda y.y/x\} = \lambda y.y \rightsquigarrow$ cut-elimination commutes with decoration.

Example of decorations of derivations in ND with cut-elimination

$$\dfrac{\dfrac{x : [X \Rightarrow X]^x}{\lambda x.x : (X \Rightarrow X) \Rightarrow X \Rightarrow X}\Rightarrow_i^x \qquad \dfrac{y : [X]^y}{\lambda y.y : X \Rightarrow X}\Rightarrow_i^y}{(\lambda x.x)\lambda y.y : X \Rightarrow X}\Rightarrow_e \quad \rightarrow_{\mathsf{cut}} \quad \dfrac{y : [X]^x}{\lambda y.y : X \Rightarrow X}\Rightarrow_i^x$$

Rmk. $(\lambda x.x)\lambda y.y \rightarrow_\beta x\{\lambda y.y/x\} = \lambda y.y \rightsquigarrow$ cut-elimination commutes with decoration.

Example of decorations of derivations in ND with cut-elimination

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \Rightarrow (B \Rightarrow A)]^x \quad [z : A]^z}{xz : B \Rightarrow A}\Rightarrow_e \quad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B}\Rightarrow_e}{(xz)(yz) : A}\Rightarrow_e}{\lambda z.(xz)(yz) : A \Rightarrow A}\Rightarrow_i^z}{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow A)}\Rightarrow_i^y}{\lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow A)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow A)}\Rightarrow_i^x \quad \dfrac{\dfrac{a : [A]^a}{\lambda b.a : B \Rightarrow A}\Rightarrow_i}{\lambda a.\lambda b.a : A \Rightarrow (B \Rightarrow A)}\Rightarrow_i^a}{(\lambda x.\lambda y.\lambda z.(xz)(yz))\lambda a.\lambda b.a : (A \Rightarrow B) \Rightarrow (A \Rightarrow A)}\Rightarrow_e
$$

$$\downarrow_{cut}$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[a : A]^a}{\lambda b.a : B \Rightarrow A}\Rightarrow_i}{\lambda a.\lambda b.a : A \Rightarrow (B \Rightarrow A)}\Rightarrow_i^a \quad [z : A]^z}{(\lambda a.\lambda b.a)z : B \Rightarrow A}\Rightarrow_e \quad \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B}\Rightarrow_e}{((\lambda a.\lambda b.a)z)(yz) : A}\Rightarrow_e}{\lambda z.((\lambda a.\lambda b.a)z)(yz) : A \Rightarrow A}\Rightarrow_i^z}{\lambda y.\lambda z.((\lambda a.\lambda b.a)z)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow A)}\Rightarrow_i^y
$$

Rmk. $(\lambda x.\lambda y.\lambda z.(xz)(yz))\lambda a.\lambda b.a \to_\beta (\lambda y.\lambda z.(xz)(yz))\{\lambda a.\lambda b.a/x\} = \lambda y.\lambda z.((\lambda a.\lambda b.a)z)(yz)$ $\rightsquigarrow$ cut-elimination commutes with decoration.

Example of decorations of derivations in ND with cut-elimination

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{[x : A \Rightarrow (B \Rightarrow A)]^x \quad [z : A]^z}{xz : B \Rightarrow A}\Rightarrow_e
        \quad
        \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B}\Rightarrow_e
      }{(xz)(yz) : A}\Rightarrow_e
    }{\lambda z.(xz)(yz) : A \Rightarrow A}\Rightarrow_i^z
  }{\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow A)}\Rightarrow_i^y
  \\
  \hline
  \lambda x.\lambda y.\lambda z.(xz)(yz) : (A \Rightarrow (B \Rightarrow A)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow A) \; \Rightarrow_i^x
  \quad
  \dfrac{\dfrac{a : [A]^a}{\lambda b.a : B \Rightarrow A}\Rightarrow_i}{\lambda a.\lambda b.a : A \Rightarrow (B \Rightarrow A)}\Rightarrow_i^a
}{(\lambda x.\lambda y.\lambda z.(xz)(yz))\lambda a.\lambda b.a : (A \Rightarrow B) \Rightarrow (A \Rightarrow A)}\Rightarrow_e
$$

$$\downarrow_{cut}$$

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{\dfrac{[a : A]^a}{\lambda b.a : B \Rightarrow A}\Rightarrow_i}{\lambda a.\lambda b.a : A \Rightarrow (B \Rightarrow A)}\Rightarrow_i^a \quad [z : A]^z
      }{(\lambda a.\lambda b.a)z : B \Rightarrow A}\Rightarrow_e
      \quad
      \dfrac{[y : A \Rightarrow B]^y \quad [z : A]^z}{yz : B}\Rightarrow_e
    }{((\lambda a.\lambda b.a)z)(yz) : A}\Rightarrow_e
  }{\lambda z.((\lambda a.\lambda b.a)z)(yz) : A \Rightarrow A}\Rightarrow_i^z
}{\lambda y.\lambda z.((\lambda a.\lambda b.a)z)(yz) : (A \Rightarrow B) \Rightarrow (A \Rightarrow A)}\Rightarrow_i^y
$$

Rmk. $(\lambda x.\lambda y.\lambda z.(xz)(yz))\lambda a.\lambda b.a \to_\beta (\lambda y.\lambda z.(xz)(yz))\{\lambda a.\lambda b.a/x\} = \lambda y.\lambda z.((\lambda a.\lambda b.a)z)(yz)$ $\rightsquigarrow$ cut-elimination commutes with decoration.

Inverse decoration: from terms to derivations

Question. Given the term $\lambda f.\lambda x.fx$, what is the derivation associated with it?

Problem. Without knowing the formulas associated with variables, there is no answer.

Question. Given the term $\lambda f^{X \Rightarrow X}.\lambda x^{X}.fx$, what is the derivation associated with it?
Question. Given the term $\lambda f^{X}.\lambda x^{X}.fx$, what is the derivation associated with it?

Rmk. Fixing formulas for variables (and hence for the whole term) is crucial!

Inverse decoration: from terms to derivations

Question. Given the term $\lambda f.\lambda x.fx$, what is the derivation associated with it?

Problem. Without knowing the formulas associated with variables, there is no answer.

Question. Given the term $\lambda f^{X\Rightarrow X}.\lambda x^{X}.fx$, what is the derivation associated with it?
Question. Given the term $\lambda f^{X}.\lambda x^{X}.fx$, what is the derivation associated with it?

Rmk. Fixing formulas for variables (and hence for the whole term) is crucial!

Inverse decoration: from terms to derivations

Question. Given the term $\lambda f.\lambda x.fx$, what is the derivation associated with it?

Problem. Without knowing the formulas associated with variables, there is no answer.

Question. Given the term $\lambda f^{X \Rightarrow X}.\lambda x^{X}.fx$, what is the derivation associated with it?
Question. Given the term $\lambda f^{X}.\lambda x^{X}.fx$, what is the derivation associated with it?

Rmk. Fixing formulas for variables (and hence for the whole term) is crucial!

Inverse decoration: from terms to derivations

Question. Given the term $\lambda f.\lambda x.fx$, what is the derivation associated with it?

Problem. Without knowing the formulas associated with variables, there is no answer.

Question. Given the term $\lambda f^{X \Rightarrow X}.\lambda x^{X}.fx$, what is the derivation associated with it?
Question. Given the term $\lambda f^{X}.\lambda x^{X}.fx$, what is the derivation associated with it?

Rmk. Fixing formulas for variables (and hence for the whole term) is crucial!

The simply typed λ-calculus in Curry-style

Types: $A, B ::= X \mid A \Rightarrow B$ (given a set of ground types ranged over by $X, Y, Z \dots$)

(λ-)Terms: $s, t ::= x \mid \lambda x.t \mid st$ (called variable, abstraction, application, respectively)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \qquad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

The free variables of a term t are the variables that are not bound to a λ. Formally,

$$\mathsf{fv}(x) = \{x\} \qquad \mathsf{fv}(st) = \mathsf{fv}(s) \cup \mathsf{fv}(t) \qquad \mathsf{fv}(\lambda x.t) = \mathsf{fv}(t) \setminus \{x\}$$

Proposition (If $\Gamma \vdash t : A$ is derivable , Γ is essentially a type assignment for $\mathsf{fv}(t)$)

1. If $\Gamma \vdash t : A$ is derivable, then so is $\Gamma, x : B \vdash t : A$, for any type B and $x \notin \mathrm{dom}(\Gamma)$.
2. If $\Gamma \vdash t : A$ is derivable, then $\mathsf{fv}(t) \subseteq \mathrm{dom}(\Gamma)$ and $\Gamma\restriction_{\mathsf{fv}(t)} \vdash t : A$ is derivable.

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

The simply typed λ-calculus in Curry-style

Types: $A, B ::= X \mid A \Rightarrow B$ (given a set of ground types ranged over by $X, Y, Z \ldots$)

(λ-)Terms: $s, t ::= x \mid \lambda x.t \mid st$ (called variable, abstraction, application, respectively)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \ldots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}@$$

The free variables of a term t are the variables that are not bound to a λ. Formally,

$$\mathrm{fv}(x) = \{x\} \qquad \mathrm{fv}(st) = \mathrm{fv}(s) \cup \mathrm{fv}(t) \qquad \mathrm{fv}(\lambda x.t) = \mathrm{fv}(t) \setminus \{x\}$$

Proposition (If $\Gamma \vdash t : A$ is derivable , Γ is essentially a type assignment for $\mathrm{fv}(t)$)

1. If $\Gamma \vdash t : A$ is derivable, then so is $\Gamma, x : B \vdash t : A$, for any type B and $x \notin \mathrm{dom}(\Gamma)$.
2. If $\Gamma \vdash t : A$ is derivable, then $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$ and $\Gamma\restriction_{\mathrm{fv}(t)} \vdash t : A$ is derivable.

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

The simply typed λ-calculus in Curry-style

Types: $A, B ::= X \mid A \Rightarrow B$ (given a set of ground types ranged over by $X, Y, Z \dots$)

(λ-)Terms: $s, t ::= x \mid \lambda x.t \mid st$ (called variable, abstraction, application, respectively)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}@$$

The free variables of a term t are the variables that are not bound to a λ. Formally,

$$\mathrm{fv}(x) = \{x\} \qquad \mathrm{fv}(st) = \mathrm{fv}(s) \cup \mathrm{fv}(t) \qquad \mathrm{fv}(\lambda x.t) = \mathrm{fv}(t) \setminus \{x\}$$

Proposition (If $\Gamma \vdash t : A$ is derivable , Γ is essentially a type assignment for $\mathrm{fv}(t)$)

1. If $\Gamma \vdash t : A$ is derivable, then so is $\Gamma, x : B \vdash t : A$, for any type B and $x \notin \mathrm{dom}(\Gamma)$.
2. If $\Gamma \vdash t : A$ is derivable, then $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$ and $\Gamma\restriction_{\mathrm{fv}(t)} \vdash t : A$ is derivable.

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

The simply typed λ-calculus in Curry-style

Types: $A, B ::= X \mid A \Rightarrow B$ (given a set of ground types ranged over by $X, Y, Z \ldots$)

(λ-)Terms: $s, t ::= x \mid \lambda x.t \mid st$ (called variable, abstraction, application, respectively)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \ldots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \qquad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

The free variables of a term t are the variables that are not bound to a λ. Formally,

$$\mathrm{fv}(x) = \{x\} \qquad \mathrm{fv}(st) = \mathrm{fv}(s) \cup \mathrm{fv}(t) \qquad \mathrm{fv}(\lambda x.t) = \mathrm{fv}(t) \setminus \{x\}$$

Proposition (If $\Gamma \vdash t : A$ is derivable , Γ is essentially a type assignment for $\mathrm{fv}(t)$)

1. If $\Gamma \vdash t : A$ is derivable, then so is $\Gamma, x : B \vdash t : A$, for any type B and $x \notin \mathrm{dom}(\Gamma)$.
2. If $\Gamma \vdash t : A$ is derivable, then $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$ and $\Gamma\restriction_{\mathrm{fv}(t)} \vdash t : A$ is derivable.

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

The simply typed λ-calculus in Curry-style

Types: $A, B ::= X \mid A \Rightarrow B$ (given a set of ground types ranged over by $X, Y, Z \dots$)

(λ-)Terms: $s, t ::= x \mid \lambda x.t \mid st$ (called variable, abstraction, application, respectively)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\mathsf{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \qquad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

The free variables of a term t are the variables that are not bound to a λ. Formally,

$$\mathsf{fv}(x) = \{x\} \qquad \mathsf{fv}(st) = \mathsf{fv}(s) \cup \mathsf{fv}(t) \qquad \mathsf{fv}(\lambda x.t) = \mathsf{fv}(t) \setminus \{x\}$$

Proposition (If $\Gamma \vdash t : A$ is derivable , Γ is essentially a type assignment for $\mathsf{fv}(t)$)

1. If $\Gamma \vdash t : A$ is derivable, then so is $\Gamma, x : B \vdash t : A$, for any type B and $x \notin \mathsf{dom}(\Gamma)$.
2. If $\Gamma \vdash t : A$ is derivable, then $\mathsf{fv}(t) \subseteq \mathsf{dom}(\Gamma)$ and $\Gamma\restriction_{\mathsf{fv}(t)} \vdash t : A$ is derivable.

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x.t)s \to_\beta t\{s/x\}$$

The capture-avoiding substitution

Naive substitution $t[s/x]$: replacement of the free occurrences of the variable x in t by s.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t[s/x] = \lambda y.y(yy)$.
Problem: The free variable y in s has been captured by the λ in t. $\rightsquigarrow$ Undesirable.

Solution: Capture-avoiding substitution $t\{s/x\}$

1. rename the bound variables in t with variables that do not occur in t or s;
2. perform the substitution in of s for x in t.

$\rightsquigarrow$ So, the free variables of s are not captured by the λ's in t.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t\{s/x\} = \lambda z.z(yy)$.

Rmk: The operation of renaming the bound variables in a term is called α-equivalence.
$\rightsquigarrow$ Capture-avoiding substitution makes sense, as we identify terms up to α-equivalence.

The capture-avoiding substitution

Naive substitution $t[s/x]$: replacement of the free occurrences of the variable x in t by s.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t[s/x] = \lambda y.y(yy)$.
Problem: The free variable y in s has been captured by the λ in t. $\rightsquigarrow$ Undesirable.

Solution: Capture-avoiding substitution $t\{s/x\}$

1. rename the bound variables in t with variables that do not occur in t or s;
2. perform the substitution in of s for x in t.

$\rightsquigarrow$ So, the free variables of s are not captured by the λ's in t.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t\{s/x\} = \lambda z.z(yy)$.

Rmk: The operation of renaming the bound variables in a term is called α-equivalence.
$\rightsquigarrow$ Capture-avoiding substitution makes sense, as we identify terms up to α-equivalence.

The capture-avoiding substitution

Naive substitution $t[s/x]$: replacement of the free occurrences of the variable x in t by s.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t[s/x] = \lambda y.y(yy)$.
Problem: The free variable y in s has been captured by the λ in t. $\leadsto$ Undesirable.

Solution: Capture-avoiding substitution $t\{s/x\}$

1. rename the bound variables in t with variables that do not occur in t or s;
2. perform the substitution in of s for x in t.

$\leadsto$ So, the free variables of s are not captured by the λ's in t.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t\{s/x\} = \lambda z.z(yy)$.

Rmk: The operation of renaming the bound variables in a term is called α-equivalence.
$\leadsto$ Capture-avoiding substitution makes sense, as we identify terms up to α-equivalence.

The capture-avoiding substitution

Naive substitution $t[s/x]$: replacement of the free occurrences of the variable x in t by s.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t[s/x] = \lambda y.y(yy)$.
Problem: The free variable y in s has been captured by the λ in t. $\rightsquigarrow$ Undesirable.

Solution: Capture-avoiding substitution $t\{s/x\}$

1. rename the bound variables in t with variables that do not occur in t or s;
2. perform the substitution in of s for x in t.

$\rightsquigarrow$ So, the free variables of s are not captured by the λ's in t.

Ex: Let $t = \lambda y.yx$ and $s = yy$. Then, $t\{s/x\} = \lambda z.z(yy)$.

Rmk: The operation of renaming the bound variables in a term is called α-equivalence.
$\rightsquigarrow$ Capture-avoiding substitution makes sense, as we identify terms up to α-equivalence.

Some remarks about the simply typed λ-calculus in Curry-style

Rmk. The search for a derivation is uniquely determined by the term (syntax-directed).
$\rightsquigarrow$ To build a derivation $\mathcal{D}$ of $\Gamma \vdash t : A$, just look at t to know the last rule of $\mathcal{D}$ (if any).

The types used in the simply typed λ-calculus are exactly the formulas of minimal logic.
The inference rules for the simply typed λ-calculus are the ones of $\mathsf{ND}_{\mathsf{seq}}$ plus decoration.
$\rightsquigarrow$ Every derivation in $\mathsf{ND}/\mathsf{ND}_{\mathsf{seq}}$ corresponds to a unique λ-term typed in Curry-style.

Question: With every typable term in Curry-style is it associated a unique derivation? No!

$$\dfrac{\dfrac{}{x : X \vdash x : X}\,\mathsf{var}}{\vdash \lambda x.x : X \Rightarrow X}\,\lambda \qquad \dfrac{\dfrac{}{x : X \Rightarrow X \vdash x : X \Rightarrow X}\,\mathsf{var}}{\vdash \lambda x.x : (X \Rightarrow X) \Rightarrow X \Rightarrow X}\,\lambda$$

$\rightsquigarrow$ The map from typable terms in Curry-style to $\mathsf{ND}/\mathsf{ND}_{\mathsf{seq}}$ derivations is not injective!

Idea: In Curry-style, types are extrinsic to terms (dynamic typing, *a posteriori*)
$\rightsquigarrow$ Let us make them intrinsic to terms (static typing, *a priori*): Church-style.

Some remarks about the simply typed λ-calculus in Curry-style

Rmk. The search for a derivation is uniquely determined by the term (syntax-directed).
$\rightsquigarrow$ To build a derivation $\mathcal{D}$ of $\Gamma \vdash t : A$, just look at t to know the last rule of $\mathcal{D}$ (if any).

The types used in the simply typed λ-calculus are exactly the formulas of minimal logic.
The inference rules for the simply typed λ-calculus are the ones of $\mathsf{ND_{seq}}$ plus decoration.
$\rightsquigarrow$ Every derivation in ND/$\mathsf{ND_{seq}}$ corresponds to a unique λ-term typed in Curry-style.

Question: With every typable term in Curry-style is it associated a unique derivation? No!

$$\dfrac{\dfrac{}{x : X \vdash x : X}\,\mathsf{var}}{\vdash \lambda x.x : X \Rightarrow X}\,\lambda \qquad \dfrac{\dfrac{}{x : X \Rightarrow X \vdash x : X \Rightarrow X}\,\mathsf{var}}{\vdash \lambda x.x : (X \Rightarrow X) \Rightarrow X \Rightarrow X}\,\lambda$$

$\rightsquigarrow$ The map from typable terms in Curry-style to ND/$\mathsf{ND_{seq}}$ derivations is not injective!

Idea: In Curry-style, types are extrinsic to terms (dynamic typing, *a posteriori*)
$\rightsquigarrow$ Let us make them intrinsic to terms (static typing, *a priori*): Church-style.

Some remarks about the simply typed λ-calculus in Curry-style

Rmk. The search for a derivation is uniquely determined by the term (syntax-directed).
$\rightsquigarrow$ To build a derivation $\mathcal{D}$ of $\Gamma \vdash t : A$, just look at t to know the last rule of $\mathcal{D}$ (if any).

The types used in the simply typed λ-calculus are exactly the formulas of minimal logic.
The inference rules for the simply typed λ-calculus are the ones of $\mathsf{ND}_{\mathsf{seq}}$ plus decoration.
$\rightsquigarrow$ Every derivation in $\mathsf{ND}/\mathsf{ND}_{\mathsf{seq}}$ corresponds to a unique λ-term typed in Curry-style.

Question: With every typable term in Curry-style is it associated a unique derivation? No!

$$\dfrac{\dfrac{}{x : X \vdash x : X}\,\mathsf{var}}{\vdash \lambda x.x : X \Rightarrow X}\,\lambda \qquad\qquad \dfrac{\dfrac{}{x : X \Rightarrow X \vdash x : X \Rightarrow X}\,\mathsf{var}}{\vdash \lambda x.x : (X \Rightarrow X) \Rightarrow X \Rightarrow X}\,\lambda$$

$\rightsquigarrow$ The map from typable terms in Curry-style to $\mathsf{ND}/\mathsf{ND}_{\mathsf{seq}}$ derivations is not injective!

Idea: In Curry-style, types are extrinsic to terms (dynamic typing, *a posteriori*)
$\rightsquigarrow$ Let us make them intrinsic to terms (static typing, *a priori*): Church-style.

Some remarks about the simply typed λ-calculus in Curry-style

Rmk. The search for a derivation is uniquely determined by the term (syntax-directed).
$\rightsquigarrow$ To build a derivation $\mathcal{D}$ of $\Gamma \vdash t : A$, just look at t to know the last rule of $\mathcal{D}$ (if any).

The types used in the simply typed λ-calculus are exactly the formulas of minimal logic.
The inference rules for the simply typed λ-calculus are the ones of $\mathsf{ND_{seq}}$ plus decoration.
$\rightsquigarrow$ Every derivation in $\mathsf{ND}/\mathsf{ND_{seq}}$ corresponds to a unique λ-term typed in Curry-style.

Question: With every typable term in Curry-style is it associated a unique derivation? No!

$$\dfrac{\dfrac{}{x : X \vdash x : X}\,\mathsf{var}}{\vdash \lambda x.x : X \Rightarrow X}\,\lambda \qquad\qquad \dfrac{\dfrac{}{x : X \Rightarrow X \vdash x : X \Rightarrow X}\,\mathsf{var}}{\vdash \lambda x.x : (X \Rightarrow X) \Rightarrow X \Rightarrow X}\,\lambda$$

$\rightsquigarrow$ The map from typable terms in Curry-style to $\mathsf{ND}/\mathsf{ND_{seq}}$ derivations is not injective!

Idea: In Curry-style, types are extrinsic to terms (dynamic typing, *a posteriori*)
$\rightsquigarrow$ Let us make them intrinsic to terms (static typing, *a priori*): Church-style.

The simply typed λ-calculus in Church-style

Idea: Let us make types intrinsic to terms (static typing, *a priori*): Church-style.
$\rightsquigarrow$ Every abstracted variable in a term is associated with some type.

(λ-)Terms: $s, t ::= x \mid \lambda x^A.t \mid st$ (where A is any type, as defined for Curry-style)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x^A.t)s \to_\beta t\{s/x\}$$

Rmk. Syntax-directedness and proposition on p. 11 hold true in Church-style as well.

Notation: $\Gamma \vdash_{\text{Curry/Church}} t : A$ if there is a derivation of $\Gamma \vdash t : A$ in Curry/Church-style.

The simply typed λ-calculus in Church-style

Idea: Let us make types intrinsic to terms (static typing, *a priori*): Church-style.
$\rightsquigarrow$ Every abstracted variable in a term is associated with some type.

(λ-)Terms: $s, t ::= x \mid \lambda x^A.t \mid st$ (where A is any type, as defined for Curry-style)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x^A.t)s \to_\beta t\{s/x\}$$

Rmk. Syntax-directedness and proposition on p. 11 hold true in Church-style as well.

Notation: $\Gamma \vdash_{\text{Curry/Church}} t : A$ if there is a derivation of $\Gamma \vdash t : A$ in Curry/Church-style.

The simply typed λ-calculus in Church-style

Idea: Let us make types intrinsic to terms (static typing, *a priori*): Church-style.
$\rightsquigarrow$ Every abstracted variable in a term is associated with some type.

(λ-)Terms: $s, t ::= x \mid \lambda x^{A}.t \mid st$ (where A is any type, as defined for Curry-style)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\mathsf{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^{A}.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x^{A}.t)s \to_{\beta} t\{s/x\}$$

Rmk. Syntax-directedness and proposition on p. 11 hold true in Church-style as well.

Notation: $\Gamma \vdash_{\text{Curry/Church}} t : A$ if there is a derivation of $\Gamma \vdash t : A$ in Curry/Church-style.

The simply typed λ-calculus in Church-style

Idea: Let us make types intrinsic to terms (static typing, *a priori*): Church-style.
$\rightsquigarrow$ Every abstracted variable in a term is associated with some type.

(λ-)Terms: $\quad s, t ::= x \mid \lambda x^A.t \mid st \quad$ (where A is any type, as defined for Curry-style)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x^A.t)s \to_\beta t\{s/x\}$$

Rmk. Syntax-directedness and proposition on p. 11 hold true in Church-style as well.

Notation: $\Gamma \vdash_{\text{Curry/Church}} t : A$ if there is a derivation of $\Gamma \vdash t : A$ in Curry/Church-style.

The simply typed λ-calculus in Church-style

Idea: Let us make types intrinsic to terms (static typing, *a priori*): Church-style.
$\rightsquigarrow$ Every abstracted variable in a term is associated with some type.

(λ-)Terms: $s, t ::= x \mid \lambda x^A.t \mid st$ (where A is any type, as defined for Curry-style)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x^A.t)s \to_\beta t\{s/x\}$$

Rmk. Syntax-directedness and proposition on p. 11 hold true in Church-style as well.

Notation: $\Gamma \vdash_{\text{Curry/Church}} t : A$ if there is a derivation of $\Gamma \vdash t : A$ in Curry/Church-style.

The simply typed λ-calculus in Church-style

Idea: Let us make types intrinsic to terms (static typing, *a priori*): Church-style.
$\rightsquigarrow$ Every abstracted variable in a term is associated with some type.

$(\lambda$-$)$Terms: $\quad s, t ::= x \mid \lambda x^{A}.t \mid st \quad$ (where A is any type, as defined for Curry-style)

Environment: function from finitely many variables to types (noted $x_1 : A_1, \dots, x_n : A_n$).
The well-typed terms are the ones that can be constructed via the typing rules below.

$$\frac{}{\Gamma, x : A \vdash x : A}\,\text{var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^{A}.t : A \Rightarrow B}\,\lambda \qquad \frac{\Gamma \vdash s : B \Rightarrow A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A}\,@$$

β-reduction ($t\{s/x\}$ is the capture-avoiding substitution of s for the free occurrences of x in t):

$$(\lambda x^{A}.t)s \to_{\beta} t\{s/x\}$$

Rmk. Syntax-directedness and proposition on p. 11 hold true in Church-style as well.

Notation: $\Gamma \vdash_{\text{Curry/Church}} t : A$ if there is a derivation of $\Gamma \vdash t : A$ in Curry/Church-style.

Curry-style versus Church-style

Church-style terms are related to Curry-style terms by the forgetful function $\lceil \cdot \rceil$:

$$\lceil x \rceil = x \qquad \lceil \lambda x^A . t \rceil = \lambda x . t \qquad \lceil st \rceil = \lceil s \rceil \lceil t \rceil$$

Proposition

1. If $\Gamma \vdash t : A$ is derivable in Church-style, then $\Gamma \vdash \lceil t \rceil : A$ is derivable in Curry-style.
2. If $\Gamma \vdash_{\text{Curry}} t : A$ then $\Gamma \vdash_{\text{Church}} t' : A$ for some t' in Church-style such that $\lceil t' \rceil = t$.

Proof. By induction on the derivation in Church (Point 1) or Curry (Point 2) style. □

Rmk: $\lambda x^X . x$ and $\lambda x^{X \Rightarrow X} . x$ are different terms in Church-style, because $X \neq X \Rightarrow X$.

Proposition (Uniqueness of type and derivation for typable terms in Church-style)

In Church-style, if $\mathcal{D}$ derives $\Gamma \vdash t : A$ and $\mathcal{D}'$ derives $\Gamma \vdash t : A'$, then $A = A'$ and $\mathcal{D} = \mathcal{D}'$.

Proof. By structural induction on t (exercise!). □

⇝ A bijection between typable terms in Church-style and derivations in ND/ND_{seq}.
⇝ As β-reduction and cut-elimination mimic each other, it is an isomorphism.

Curry-style versus Church-style

Church-style terms are related to Curry-style terms by the forgetful function $\lceil \cdot \rceil$:

$$\lceil x \rceil = x \qquad \lceil \lambda x^A.t \rceil = \lambda x.t \qquad \lceil st \rceil = \lceil s \rceil \lceil t \rceil$$

Proposition

1. If $\Gamma \vdash t : A$ is derivable in Church-style, then $\Gamma \vdash \lceil t \rceil : A$ is derivable in Curry-style.
2. If $\Gamma \vdash_{\mathsf{Curry}} t : A$ then $\Gamma \vdash_{\mathsf{Church}} t' : A$ for some t' in Church-style such that $\lceil t' \rceil = t$.

Proof. By induction on the derivation in Church (Point 1) or Curry (Point 2) style. □

Rmk: $\lambda x^X.x$ and $\lambda x^{X \Rightarrow X}.x$ are different terms in Church-style, because $X \neq X \Rightarrow X$.

Proposition (Uniqueness of type and derivation for typable terms in Church-style)

In Church-style, if $\mathcal{D}$ derives $\Gamma \vdash t : A$ and $\mathcal{D}'$ derives $\Gamma \vdash t : A'$, then $A = A'$ and $\mathcal{D} = \mathcal{D}'$.

Proof. By structural induction on t (exercise!). □

$\rightsquigarrow$ A bijection between typable terms in Church-style and derivations in ND/ND$_{\mathsf{seq}}$.
$\rightsquigarrow$ As β-reduction and cut-elimination mimic each other, it is an isomorphism.

Curry-style versus Church-style

Church-style terms are related to Curry-style terms by the forgetful function $\lceil \cdot \rceil$:

$$\lceil x \rceil = x \qquad \lceil \lambda x^A.t \rceil = \lambda x.\lceil t \rceil \qquad \lceil st \rceil = \lceil s \rceil \lceil t \rceil$$

Proposition

1. If $\Gamma \vdash t : A$ is derivable in Church-style, then $\Gamma \vdash \lceil t \rceil : A$ is derivable in Curry-style.
2. If $\Gamma \vdash_{\text{Curry}} t : A$ then $\Gamma \vdash_{\text{Church}} t' : A$ for some t' in Church-style such that $\lceil t' \rceil = t$.

Proof. By induction on the derivation in Church (Point 1) or Curry (Point 2) style. □

Rmk: $\lambda x^X.x$ and $\lambda x^{X \Rightarrow X}.x$ are different terms in Church-style, because $X \neq X \Rightarrow X$.

Proposition (Uniqueness of type and derivation for typable terms in Church-style)

In Church-style, if $\mathcal{D}$ derives $\Gamma \vdash t : A$ and $\mathcal{D}'$ derives $\Gamma \vdash t : A'$, then $A = A'$ and $\mathcal{D} = \mathcal{D}'$.

Proof. By structural induction on t (exercise!). □

$\rightsquigarrow$ A bijection between typable terms in Church-style and derivations in $\text{ND}/\text{ND}_{\text{seq}}$.

$\rightsquigarrow$ As β-reduction and cut-elimination mimic each other, it is an isomorphism.

Some properties of the simply typed λ-calculus (Curry and Church style)

Lemma (Substitution)

If $\Gamma, x : B \vdash t : A$ and $\Gamma \vdash s : B$ are derivable, then so is $\Gamma \vdash t\{s/x\} : A$.

Proof. By structural induction on t (exercise!). □

Theorem (Subject reduction)

If $\Gamma \vdash t : A$ is derivable and $t \to_\beta s$, then $\Gamma \vdash s : A$ is derivable.

Proof. By structural induction on t, using the substitution lemma in the key-case. □

Rmk. The converse (subject expansion) does not hold: let $A = Y \Rightarrow (X \Rightarrow X)$, and $t = (\lambda x.\lambda y.\lambda z.(xz)(yz))\lambda x.\lambda y.x$ and $s = \lambda y.\lambda z.z$, then $t \to_\beta^* s$ and $\vdash s : A$, but $\nvdash t : A$.

Theorem (Normalization)

If $\Gamma \vdash t : A$ is derivable, then $t \to_\beta^* s$ and for some derivation of $\Gamma \vdash s : A$ without redexes.

Proof. Exactly the proof of cut-elimination in ND (uppermost in ND $\rightsquigarrow$ innermost in λ). □

Some properties of the simply typed λ-calculus (Curry and Church style)

Lemma (Substitution)

If $\Gamma, x : B \vdash t : A$ and $\Gamma \vdash s : B$ are derivable, then so is $\Gamma \vdash t\{s/x\} : A$.

Proof. By structural induction on t (exercise!). □

Theorem (Subject reduction)

If $\Gamma \vdash t : A$ is derivable and $t \to_\beta s$, then $\Gamma \vdash s : A$ is derivable.

Proof. By structural induction on t, using the substitution lemma in the key-case. □

Rmk. The converse (subject expansion) does not hold: let $A = Y \Rightarrow (X \Rightarrow X)$, and $t = (\lambda x.\lambda y.\lambda z.(xz)(yz))\lambda x.\lambda y.x$ and $s = \lambda y.\lambda z.z$, then $t \to_\beta^* s$ and $\vdash s : A$, but $\nvdash t : A$.

Theorem (Normalization)

If $\Gamma \vdash t : A$ is derivable, then $t \to_\beta^* s$ and for some derivation of $\Gamma \vdash s : A$ without redexes.

Proof. Exactly the proof of cut-elimination in ND (uppermost in ND $\rightsquigarrow$ innermost in λ). □

Some properties of the simply typed λ-calculus (Curry and Church style)

Lemma (Substitution)

If $\Gamma, x : B \vdash t : A$ and $\Gamma \vdash s : B$ are derivable, then so is $\Gamma \vdash t\{s/x\} : A$.

Proof. By structural induction on t (exercise!). □

Theorem (Subject reduction)

If $\Gamma \vdash t : A$ is derivable and $t \to_\beta s$, then $\Gamma \vdash s : A$ is derivable.

Proof. By structural induction on t, using the substitution lemma in the key-case. □

Rmk. The converse (subject expansion) does not hold: let $A = Y \Rightarrow (X \Rightarrow X)$, and $t = (\lambda x.\lambda y.\lambda z.(xz)(yz))\lambda x.\lambda y.x$ and $s = \lambda y.\lambda z.z$, then $t \to_\beta^* s$ and $\vdash s : A$, but $\nvdash t : A$.

Theorem (Normalization)

If $\Gamma \vdash t : A$ is derivable, then $t \to_\beta^* s$ and for some derivation of $\Gamma \vdash s : A$ without redexes.

Proof. Exactly the proof of cut-elimination in ND (uppermost in ND $\rightsquigarrow$ innermost in λ). □

The Curry-Howard correspondence

minimal logic	simply typed λ-calculus	computer science
formula	type	specification
derivation	term	program
cut-elimination step	β-reduction	computation step
derivation without redexes	normal form	result
cut-elimination theorem	normalization	termination

Concerning the correspondence between derivations and terms:

derivation in minimal logic	term in simply typed λ-calculus
hypotheses	variable
$\Rightarrow_i$	abstraction λ
$\Rightarrow_e$	application @

The Curry-Howard correspondence

minimal logic	simply typed λ-calculus	computer science
formula	type	specification
derivation	term	program
cut-elimination step	β-reduction	computation step
derivation without redexes	normal form	result
cut-elimination theorem	normalization	termination

Concerning the correspondence between derivations and terms:

derivation in minimal logic	term in simply typed λ-calculus
hypotheses	variable
$\Rightarrow_i$	abstraction λ
$\Rightarrow_e$	application @

Outline

Abstract rewriting systems: normalization versus strong normalization

We have seen different sets (of derivations, λ-terms) and reductions (cut elimination, β).
$\rightsquigarrow$ Let us consider them abstractly, to study their common properties uniformly.

Def: An abstract rewriting system (ARS) is a set A and a relation $\to \subseteq A \times A$ (reduction).
The reflexive-transitive closure of $\to$ is $\to^*$, that is, $t \to^* s$ means $t \underbrace{\to \cdots \to}_{n \in \mathbb{N} \text{ times } \to} s$.

- $t \in A$ is normal if there is no $s \in A$ such that $t \to s$.
- $t \in A$ is normalizing if there is $u \in A$ such that $t \to^* u$.
- $t \in A$ is strongly normalizing if there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ with $t_0 = t$ and $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$, i.e. every reduction sequence eventually reaches a normal form.
- $\to$ is normalizing/strongly normalizing if so is every $t \in A$.
- $\to$ is confluent if for all $t, r_1, r_2 \in A$ with $r_1 {}^*\!\leftarrow t \to^* r_2$, $r_1 \to^* s {}^*\!\leftarrow r_2$ for some $s \in A$.

Rmk: Strong normalization implies normalization but the converse fails.

Proposition (Uniqueness of normal form)

If $\to$ is confluent, then for all $t \in A$ there is at most one normal $s \in A$ with $t \to^* s$.

Proof. If $r {}^*\!\leftarrow t \to^* s$ with r, s normal, by confluence $\exists\, u \in A$: $r \to^* u {}^*\!\leftarrow s$, so $r = u = s$. $\square$

Abstract rewriting systems: normalization versus strong normalization

We have seen different sets (of derivations, λ-terms) and reductions (cut elimination, β).
$\rightsquigarrow$ Let us consider them abstractly, to study their common properties uniformly.

Def: An abstract rewriting system (ARS) is a set A and a relation $\to \subseteq A \times A$ (reduction).
The reflexive-transitive closure of $\to$ is $\to^*$, that is, $t \to^* s$ means $t \underbrace{\to \cdots \to}_{n \in \mathbb{N} \text{ times } \to} s$.

- $t \in A$ is normal if there is no $s \in A$ such that $t \to s$.
- $t \in A$ is normalizing if there is $u \in A$ such that $t \to^* u$.
- $t \in A$ is strongly normalizing if there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ with $t_0 = t$ and $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$, i.e. every reduction sequence eventually reaches a normal form.
- $\to$ is normalizing/strongly normalizing if so is every $t \in A$.
- $\to$ is confluent if for all $t, r_1, r_2 \in A$ with $r_1 {}^*\!\leftarrow t \to^* r_2$, $r_1 \to^* s {}^*\!\leftarrow r_2$ for some $s \in A$.

Rmk: Strong normalization implies normalization but the converse fails.

Proposition (Uniqueness of normal form)

If $\to$ is confluent, then for all $t \in A$ there is at most one normal $s \in A$ with $t \to^* s$.

Proof. If $r {}^*\!\leftarrow t \to^* s$ with r, s normal, by confluence $\exists\, u \in A$: $r \to^* u {}^*\!\leftarrow s$, so $r = u = s$. □

Abstract rewriting systems: normalization versus strong normalization

We have seen different sets (of derivations, λ-terms) and reductions (cut elimination, β).
$\rightsquigarrow$ Let us consider them abstractly, to study their common properties uniformly.

Def: An abstract rewriting system (ARS) is a set A and a relation $\to \subseteq A \times A$ (reduction).
The reflexive-transitive closure of $\to$ is $\to^*$, that is, $t \to^* s$ means $t \underbrace{\to \cdots \to}_{n \in \mathbb{N} \text{ times } \to} s$.

- $t \in A$ is normal if there is no $s \in A$ such that $t \to s$.
- $t \in A$ is normalizing if there is $u \in A$ such that $t \to^* u$.
- $t \in A$ is strongly normalizing if there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ with $t_0 = t$ and $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$, i.e. every reduction sequence eventually reaches a normal form.
- $\to$ is normalizing/strongly normalizing if so is every $t \in A$.
- $\to$ is confluent if for all $t, r_1, r_2 \in A$ with $r_1 \,{}^*\!\leftarrow t \to^* r_2$, $r_1 \to^* s \,{}^*\!\leftarrow r_2$ for some $s \in A$.

Rmk: Strong normalization implies normalization but the converse fails.

Proposition (Uniqueness of normal form)

If $\to$ is confluent, then for all $t \in A$ there is at most one normal $s \in A$ with $t \to^* s$.

Proof. If $r \,{}^*\!\leftarrow t \to^* s$ with r, s normal, by confluence $\exists\, u \in A$: $r \to^* u \,{}^*\!\leftarrow s$, so $r = u = s$. $\square$

Abstract rewriting systems: normalization versus strong normalization

We have seen different sets (of derivations, λ-terms) and reductions (cut elimination, β).
$\rightsquigarrow$ Let us consider them abstractly, to study their common properties uniformly.

Def: An abstract rewriting system (ARS) is a set A and a relation $\to \subseteq A \times A$ (reduction).
The reflexive-transitive closure of $\to$ is $\to^*$, that is, $t \to^* s$ means $t \underbrace{\to \cdots \to}_{n \in \mathbb{N} \text{ times } \to} s$.

- $t \in A$ is normal if there is no $s \in A$ such that $t \to s$.
- $t \in A$ is normalizing if there is $u \in A$ such that $t \to^* u$.
- $t \in A$ is strongly normalizing if there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ with $t_0 = t$ and $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$, i.e. every reduction sequence eventually reaches a normal form.
- $\to$ is normalizing/strongly normalizing if so is every $t \in A$.
- $\to$ is confluent if for all $t, r_1, r_2 \in A$ with $r_1 {}^*\!\!\leftarrow t \to^* r_2$, $r_1 \to^* s {}^*\!\!\leftarrow r_2$ for some $s \in A$.

Rmk: Strong normalization implies normalization but the converse fails.

Proposition (Uniqueness of normal form)

If $\to$ is confluent, then for all $t \in A$ there is at most one normal $s \in A$ with $t \to^* s$.

Proof. If $r {}^*\!\!\leftarrow t \to^* s$ with r, s normal, by confluence $\exists\, u \in A$: $r \to^* u {}^*\!\!\leftarrow s$, so $r = u = s$. □

Abstract rewriting systems: normalization versus strong normalization

We have seen different sets (of derivations, λ-terms) and reductions (cut elimination, β).
$\rightsquigarrow$ Let us consider them abstractly, to study their common properties uniformly.

Def: An abstract rewriting system (ARS) is a set A and a relation $\to \subseteq A \times A$ (reduction).
The reflexive-transitive closure of $\to$ is $\to^*$, that is, $t \to^* s$ means $t \underbrace{\to \cdots \to}_{n \in \mathbb{N} \text{ times } \to} s$.

- $t \in A$ is normal if there is no $s \in A$ such that $t \to s$.
- $t \in A$ is normalizing if there is $u \in A$ such that $t \to^* u$.
- $t \in A$ is strongly normalizing if there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ with $t_0 = t$ and $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$, i.e. every reduction sequence eventually reaches a normal form.
- $\to$ is normalizing/strongly normalizing if so is every $t \in A$.
- $\to$ is confluent if for all $t, r_1, r_2 \in A$ with $r_1 \,{}^*\!\!\leftarrow t \to^* r_2$, $r_1 \to^* s \,{}^*\!\!\leftarrow r_2$ for some $s \in A$.

Rmk: Strong normalization implies normalization but the converse fails.

Proposition (Uniqueness of normal form)

If $\to$ is confluent, then for all $t \in A$ there is at most one normal $s \in A$ with $t \to^* s$.

Proof. If $r \,{}^*\!\!\leftarrow t \to^* s$ with r, s normal, by confluence $\exists\, u \in A$: $r \to^* u \,{}^*\!\!\leftarrow s$, so $r = u = s$. $\square$

How to prove strong normalization: the combinatorial approach

Given a set A and a reduction $\to$ on A, we want to prove that $\to$ is strongly normalizing:
$\rightsquigarrow$ there is no (infinite) sequence $(t_i)_{i \in \mathbb{N}}$ such that $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$.

Idea (combinatorial): For every $t \in A$, we define a measure $|t| \in S$ for some well-founded set $(S, <)$ — for instance $(\mathbb{N}, <)$ — such that: for *every* $s \in A$, if $t \to s$ then $|t| > |s|$.

Problem: It is doable for the simply typed λ-calculus, but it is very tricky.
$\rightsquigarrow$ After a single β-step the size ($\approx$ number of characters) of a term may not decrease.

$$(\lambda f^{X \Rightarrow X}.f(f(fx)))(z(z(z(zf)))) \quad \to_\beta \quad (z(z(z(zf))))\Big((z(z(z(zf))))\Big((z(z(z(zf))))x\Big)\Big)$$

$\rightsquigarrow$ The measure should be defined independently of (or cannot rely on) the size of terms.

Rmk: The normalization theorem above (p. 13) does not prove strong normalization.
$\rightsquigarrow$ The proof fires a specific redex (uppermost/innermost), otherwise the argument fails.

How to prove strong normalization: the combinatorial approach

Given a set A and a reduction $\to$ on A, we want to prove that $\to$ is strongly normalizing:
$\rightsquigarrow$ there is no (infinite) sequence $(t_i)_{i \in \mathbb{N}}$ such that $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$.

Idea (combinatorial): For every $t \in A$, we define a measure $|t| \in S$ for some well-founded set $(S, <)$ — for instance $(\mathbb{N}, <)$ — such that: for *every* $s \in A$, if $t \to s$ then $|t| > |s|$.

Problem: It is doable for the simply typed λ-calculus, but it is very tricky.
$\rightsquigarrow$ After a single β-step the size ($\approx$ number of characters) of a term may not decrease.

$$(\lambda f^{X \Rightarrow X}.f(f(fx)))(z(z(z(zf)))) \quad \to_\beta \quad (z(z(z(zf))))\Big((z(z(z(zf))))\Big((z(z(z(zf))))x\Big)\Big)$$

$\rightsquigarrow$ The measure should be defined independently of (or cannot rely on) the size of terms.

Rmk: The normalization theorem above (p. 13) does not prove strong normalization.
$\rightsquigarrow$ The proof fires a specific redex (uppermost/innermost), otherwise the argument fails.

How to prove strong normalization: the combinatorial approach

Given a set A and a reduction $\to$ on A, we want to prove that $\to$ is strongly normalizing:
$\rightsquigarrow$ there is no (infinite) sequence $(t_i)_{i \in \mathbb{N}}$ such that $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$.

Idea (combinatorial): For every $t \in A$, we define a measure $|t| \in S$ for some well-founded set $(S, <)$ — for instance $(\mathbb{N}, <)$ — such that: for *every* $s \in A$, if $t \to s$ then $|t| > |s|$.

Problem: It is doable for the simply typed λ-calculus, but it is very tricky.
$\rightsquigarrow$ After a single β-step the size ($\approx$ number of characters) of a term may not decrease.

$$(\lambda f^{X \Rightarrow X}.f(f(fx)))(z(z(z(zf)))) \quad \to_\beta \quad (z(z(z(zf))))\Big((z(z(z(zf))))\Big((z(z(z(zf))))x\Big)\Big)$$

$\rightsquigarrow$ The measure should be defined independently of (or cannot rely on) the size of terms.

Rmk: The normalization theorem above (p. 13) does not prove strong normalization.
$\rightsquigarrow$ The proof fires a specific redex (uppermost/innermost), otherwise the argument fails.

How to prove strong normalization: the combinatorial approach

Given a set A and a reduction $\to$ on A, we want to prove that $\to$ is strongly normalizing:
$\leadsto$ there is no (infinite) sequence $(t_i)_{i \in \mathbb{N}}$ such that $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$.

Idea (combinatorial): For every $t \in A$, we define a measure $|t| \in S$ for some well-founded set $(S, <)$ — for instance $(\mathbb{N}, <)$ — such that: for *every* $s \in A$, if $t \to s$ then $|t| > |s|$.

Problem: It is doable for the simply typed λ-calculus, but it is very tricky.
$\leadsto$ After a single β-step the size ($\approx$ number of characters) of a term may not decrease.

$$(\lambda f^{X \Rightarrow X}.f(f(fx)))(z(z(z(zf)))) \quad \to_\beta \quad (z(z(z(zf))))\Big((z(z(z(zf))))\Big((z(z(z(zf))))x\Big)\Big)$$

$\leadsto$ The measure should be defined independently of (or cannot rely on) the size of terms.

Rmk: The normalization theorem above (p. 13) does not prove strong normalization.
$\leadsto$ The proof fires a specific redex (uppermost/innermost), otherwise the argument fails.

Reducibility candidates: a non-combinatorial approach

Idea: We define a set Red_A of terms (reducibility candidates) by induction on the type A:

- for any ground type X, Red_X is the set of strongly normalizing (SN) terms of type X;
- $\mathrm{Red}_{A \Rightarrow B}$ is the set of the terms s of type $A \Rightarrow B$ such that $st \in \mathrm{Red}_B$ for all $t \in \mathrm{Red}_A$.

Rmk: For every type A, every term in Red_A is SN (easy proof by induction on A).

Goal: For any type A, if $u : A$ then $u \in \mathrm{Red}_A$ (so u is SN). Proof by induction on u. Cases:

1. If $u = st : A$ then $s : B \Rightarrow A$ and $t : B$; by IH, $s \in \mathrm{Red}_{B \Rightarrow A}$ and $t \in \mathrm{Red}_B$, so $u \in \mathrm{Red}_A$.
2. If $u = x : X$, then u is SN, so $u \in \mathrm{Red}_X$. If $u = x : B \Rightarrow C$, to prove that $x \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $xt \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$ $\rightsquigarrow$ A stronger hypothesis is needed.
3. If $u = \lambda x^B.s : B \Rightarrow C$, to prove that $u \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $(\lambda x^B.s)t \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$. $\rightsquigarrow$ How to prove that?

Idea: Suppose $\lambda x^B.s : B \Rightarrow C$ and $t \in \mathrm{Red}_B$. Let us prove that $s\{t/x\} \in \mathrm{Red}_C$ and that if $s\{t/x\} \in \mathrm{Red}_C$ then $(\lambda x^B.s)t \in \mathrm{Red}_C$. This way, Point 3 above is done.

Problem. The environments for $\lambda x^B.s$ and t may be differ in some free variable.
$\rightsquigarrow$ The application of $\lambda x^B.s$ to t may not be possible.

Reducibility candidates: a non-combinatorial approach

Idea: We define a set Red_A of terms (reducibility candidates) by induction on the type A:

- for any ground type X, Red_X is the set of strongly normalizing (SN) terms of type X;
- $\mathrm{Red}_{A \Rightarrow B}$ is the set of the terms s of type $A \Rightarrow B$ such that $st \in \mathrm{Red}_B$ for all $t \in \mathrm{Red}_A$.

Rmk: For every type A, every term in Red_A is SN (easy proof by induction on A).

Goal: For any type A, if $u : A$ then $u \in \mathrm{Red}_A$ (so u is SN). Proof by induction on u. Cases:

1. If $u = st : A$ then $s : B \Rightarrow A$ and $t : B$; by IH, $s \in \mathrm{Red}_{B \Rightarrow A}$ and $t \in \mathrm{Red}_B$, so $u \in \mathrm{Red}_A$.
2. If $u = x : X$, then u is SN, so $u \in \mathrm{Red}_X$. If $u = x : B \Rightarrow C$, to prove that $x \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $xt \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$ $\rightsquigarrow$ A stronger hypothesis is needed.
3. If $u = \lambda x^B.s : B \Rightarrow C$, to prove that $u \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $(\lambda x^B.s)t \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$. $\rightsquigarrow$ How to prove that?

Idea: Suppose $\lambda x^B.s : B \Rightarrow C$ and $t \in \mathrm{Red}_B$. Let us prove that $s\{t/x\} \in \mathrm{Red}_C$ and that if $s\{t/x\} \in \mathrm{Red}_C$ then $(\lambda x^B.s)t \in \mathrm{Red}_C$. This way, Point 3 above is done.

Problem. The environments for $\lambda x^B.s$ and t may be differ in some free variable.
$\rightsquigarrow$ The application of $\lambda x^B.s$ to t may not be possible.

Reducibility candidates: a non-combinatorial approach

Idea: We define a set Red_A of terms (reducibility candidates) by induction on the type A:

- for any ground type X, Red_X is the set of strongly normalizing (SN) terms of type X;
- $\mathrm{Red}_{A \Rightarrow B}$ is the set of the terms s of type $A \Rightarrow B$ such that $st \in \mathrm{Red}_B$ for all $t \in \mathrm{Red}_A$.

Rmk: For every type A, every term in Red_A is SN (easy proof by induction on A).

Goal: For any type A, if $u : A$ then $u \in \mathrm{Red}_A$ (so u is SN). Proof by induction on u. Cases:

1. If $u = st : A$ then $s : B \Rightarrow A$ and $t : B$; by IH, $s \in \mathrm{Red}_{B \Rightarrow A}$ and $t \in \mathrm{Red}_B$, so $u \in \mathrm{Red}_A$.
2. If $u = x : X$, then u is SN, so $u \in \mathrm{Red}_X$. If $u = x : B \Rightarrow C$, to prove that $x \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $xt \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$ $\rightsquigarrow$ A stronger hypothesis is needed.
3. If $u = \lambda x^B.s : B \Rightarrow C$, to prove that $u \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $(\lambda x^B.s)t \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$. $\rightsquigarrow$ How to prove that?

Idea: Suppose $\lambda x^B.s : B \Rightarrow C$ and $t \in \mathrm{Red}_B$. Let us prove that $s\{t/x\} \in \mathrm{Red}_C$ and that if $s\{t/x\} \in \mathrm{Red}_C$ then $(\lambda x^B.s)t \in \mathrm{Red}_C$. This way, Point 3 above is done.

Problem. The environments for $\lambda x^B.s$ and t may be differ in some free variable.
$\rightsquigarrow$ The application of $\lambda x^B.s$ to t may not be possible.

Reducibility candidates: a non-combinatorial approach

Idea: We define a set Red_A of terms (reducibility candidates) by induction on the type A:

- for any ground type X, Red_X is the set of strongly normalizing (SN) terms of type X;
- $\mathrm{Red}_{A\Rightarrow B}$ is the set of the terms s of type $A\Rightarrow B$ such that $st\in\mathrm{Red}_B$ for all $t\in\mathrm{Red}_A$.

Rmk: For every type A, every term in Red_A is SN (easy proof by induction on A).

Goal: For any type A, if $u:A$ then $u\in\mathrm{Red}_A$ (so u is SN). Proof by induction on u. Cases:

1. If $u = st : A$ then $s : B\Rightarrow A$ and $t : B$; by IH, $s \in \mathrm{Red}_{B\Rightarrow A}$ and $t \in \mathrm{Red}_B$, so $u \in \mathrm{Red}_A$.
2. If $u = x : X$, then u is SN, so $u\in\mathrm{Red}_X$. If $u = x : B\Rightarrow C$, to prove that $x \in \mathrm{Red}_{B\Rightarrow C}$ we have to show that $xt \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$ $\rightsquigarrow$ A stronger hypothesis is needed.
3. If $u = \lambda x^B.s : B\Rightarrow C$, to prove that $u \in \mathrm{Red}_{B\Rightarrow C}$ we have to show that $(\lambda x^B.s)t \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$. $\rightsquigarrow$ How to prove that?

Idea: Suppose $\lambda x^B.s : B \Rightarrow C$ and $t \in \mathrm{Red}_B$. Let us prove that $s\{t/x\} \in \mathrm{Red}_C$ and that if $s\{t/x\} \in \mathrm{Red}_C$ then $(\lambda x^B.s)t \in \mathrm{Red}_C$. This way, Point 3 above is done.

Problem. The environments for $\lambda x^B.s$ and t may be differ in some free variable.
$\rightsquigarrow$ The application of $\lambda x^B.s$ to t may not be possible.

Reducibility candidates: a non-combinatorial approach

Idea: We define a set Red_A of terms (reducibility candidates) by induction on the type A:

- for any ground type X, Red_X is the set of strongly normalizing (SN) terms of type X;
- $\mathrm{Red}_{A \Rightarrow B}$ is the set of the terms s of type $A \Rightarrow B$ such that $st \in \mathrm{Red}_B$ for all $t \in \mathrm{Red}_A$.

Rmk: For every type A, every term in Red_A is SN (easy proof by induction on A).

Goal: For any type A, if $u : A$ then $u \in \mathrm{Red}_A$ (so u is SN). Proof by induction on u. Cases:

1. If $u = st : A$ then $s : B \Rightarrow A$ and $t : B$; by IH, $s \in \mathrm{Red}_{B \Rightarrow A}$ and $t \in \mathrm{Red}_B$, so $u \in \mathrm{Red}_A$.
2. If $u = x : X$, then u is SN, so $u \in \mathrm{Red}_X$. If $u = x : B \Rightarrow C$, to prove that $x \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $xt \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$ $\rightsquigarrow$ A stronger hypothesis is needed.
3. If $u = \lambda x^B.s : B \Rightarrow C$, to prove that $u \in \mathrm{Red}_{B \Rightarrow C}$ we have to show that $(\lambda x^B.s)t \in \mathrm{Red}_C$ for all $t \in \mathrm{Red}_B$. $\rightsquigarrow$ How to prove that?

Idea: Suppose $\lambda x^B.s : B \Rightarrow C$ and $t \in \mathrm{Red}_B$. Let us prove that $s\{t/x\} \in \mathrm{Red}_C$ and that if $s\{t/x\} \in \mathrm{Red}_C$ then $(\lambda x^B.s)t \in \mathrm{Red}_C$. This way, Point 3 above is done.

Problem. The environments for $\lambda x^B.s$ and t may be differ in some free variable.
$\rightsquigarrow$ The application of $\lambda x^B.s$ to t may not be possible.

Fixing the reducibility candidate method

Solution: Let us take the environment into account when defining Red_A, for all types A.

$$\mathsf{Red}_X = \{\langle \Gamma; t\rangle \mid t \text{ is SN},\ \Gamma \vdash t : X\}$$

$$\mathsf{Red}_{A\Rightarrow B} = \{\langle \Gamma; s\rangle \mid \Gamma \vdash s : A \Rightarrow B,\ \langle \Gamma, \Delta;\ st\rangle \in \mathsf{Red}_B \text{ for all } \langle \Gamma, \Delta;\ t\rangle \in \mathsf{Red}_A\}$$

Lemma

1. If $\langle \Gamma; t\rangle \in \mathsf{Red}_B$ then t is SN.
2. If $\Gamma \vdash xt_1 \dots t_n : B$ and $t_1, \dots, t_n$ are SN, then $\langle \Gamma; xt_1 \dots t_n\rangle \in \mathsf{Red}_B$.
3. If $\langle \Gamma; s\{t/x\}t_1 \dots t_n\rangle \in \mathsf{Red}_B$, $\Gamma \vdash t : A$ and t is SN, then $\langle \Gamma; (\lambda x^A.s)tt_1 \dots t_n\rangle \in \mathsf{Red}_B$.

Proof. Points 1–3 are proved simultaneously by induction on the type B. If $B = X$ then Point 1 is by definition of Red_X, for Points 2–3 see Exercise 14, p. 24. Let $B = C \Rightarrow D$.

1. Let $z \notin \mathrm{dom}(\Gamma)$, so $\langle \Gamma, z : C;\ z\rangle \in \mathsf{Red}_C$ by the induction hypothesis of Point 2 applied to C. As $\langle \Gamma; t\rangle \in \mathsf{Red}_{C\Rightarrow D}$, then $\langle \Gamma, z : C; tz\rangle \in \mathsf{Red}_D$ and hence tz is SN by the induction hypothesis of Point 1 applied to D; thus t is SN too.
2. Let $\langle \Gamma, \Delta;\ t\rangle \in \mathsf{Red}_C$, so t is SN by induction hypothesis of Point 1 applied to C; as $\Gamma, \Delta \vdash xt_1 \dots t_n t : D$ is derivable, $\langle \Gamma, \Delta; xt_1 \dots t_n t\rangle \in \mathsf{Red}_D$ by induction hypothesis of Point 2 applied to A; hence, $\langle \Gamma; xt_1 \dots t_n\rangle \in \mathsf{Red}_B$ by definition of $\mathsf{Red}_{C\Rightarrow D}$.
3. Let $\langle \Gamma, \Delta;\ r\rangle \in \mathsf{Red}_C$, so $\langle \Gamma, \Delta; s\{t/x\}t_1 \dots t_n r\rangle \in \mathsf{Red}_D$ and hence, by the induction hypothesis, $\langle \Gamma, \Delta; (\lambda x^A.s)tt_1 \dots t_n r\rangle \in \mathsf{Red}_D$; thus, $\langle \Gamma; (\lambda x^A.s)tt_1 \dots t_n\rangle \in \mathsf{Red}_B$. □

Fixing the reducibility candidate method

Solution: Let us take the environment into account when defining Red_A, for all types A.

$$\mathsf{Red}_X = \{\langle \Gamma; t\rangle \mid t \text{ is SN},\ \Gamma \vdash t : X\}$$
$$\mathsf{Red}_{A\Rightarrow B} = \{\langle \Gamma; s\rangle \mid \Gamma \vdash s : A \Rightarrow B,\ \langle \Gamma, \Delta;\ st\rangle \in \mathsf{Red}_B \text{ for all } \langle \Gamma, \Delta;\ t\rangle \in \mathsf{Red}_A\}$$

Lemma

1. If $\langle \Gamma; t\rangle \in \mathsf{Red}_B$ then t is SN.
2. If $\Gamma \vdash x t_1 \dots t_n : B$ and $t_1, \dots, t_n$ are SN, then $\langle \Gamma; x t_1 \dots t_n\rangle \in \mathsf{Red}_B$.
3. If $\langle \Gamma; s\{t/x\} t_1 \dots t_n\rangle \in \mathsf{Red}_B$, $\Gamma \vdash t : A$ and t is SN, then $\langle \Gamma; (\lambda x^A.s) t t_1 \dots t_n\rangle \in \mathsf{Red}_B$.

Proof. Points 1–3 are proved simultaneously by induction on the type B. If $B = X$ then Point 1 is by definition of Red_X, for Points 2–3 see Exercise 14, p. 24. Let $B = C \Rightarrow D$.

1. Let $z \notin \mathrm{dom}(\Gamma)$, so $\langle \Gamma, z : C;\ z\rangle \in \mathsf{Red}_C$ by the induction hypothesis of Point 2 applied to C. As $\langle \Gamma; t\rangle \in \mathsf{Red}_{C\Rightarrow D}$, then $\langle \Gamma, z : C; tz\rangle \in \mathsf{Red}_D$ and hence tz is SN by the induction hypothesis of Point 1 applied to D; thus t is SN too.
2. Let $\langle \Gamma, \Delta;\ t\rangle \in \mathsf{Red}_C$, so t is SN by induction hypothesis of Point 1 applied to C; as $\Gamma, \Delta \vdash x t_1 \dots t_n t : D$ is derivable, $\langle \Gamma, \Delta; x t_1 \dots t_n t\rangle \in \mathsf{Red}_D$ by induction hypothesis of Point 2 applied to A; hence, $\langle \Gamma; x t_1 \dots t_n\rangle \in \mathsf{Red}_B$ by definition of $\mathsf{Red}_{C\Rightarrow D}$.
3. Let $\langle \Gamma, \Delta;\ r\rangle \in \mathsf{Red}_C$, so $\langle \Gamma, \Delta; s\{t/x\} t_1 \dots t_n r\rangle \in \mathsf{Red}_D$ and hence, by the induction hypothesis, $\langle \Gamma, \Delta; (\lambda x^A.s) t t_1 \dots t_n r\rangle \in \mathsf{Red}_D$; thus, $\langle \Gamma; (\lambda x^A.s) t t_1 \dots t_n\rangle \in \mathsf{Red}_B$. □

Fixing the reducibility candidate method

Solution: Let us take the environment into account when defining Red_A, for all types A.

$$\mathrm{Red}_X = \{\langle \Gamma; t\rangle \mid t \text{ is SN}, \ \Gamma \vdash t : X\}$$
$$\mathrm{Red}_{A \Rightarrow B} = \{\langle \Gamma; s\rangle \mid \Gamma \vdash s : A \Rightarrow B, \ \langle \Gamma, \Delta;\ st\rangle \in \mathrm{Red}_B \text{ for all } \langle \Gamma, \Delta;\ t\rangle \in \mathrm{Red}_A\}$$

Lemma

1. If $\langle \Gamma; t\rangle \in \mathrm{Red}_B$ then t is SN.
2. If $\Gamma \vdash x t_1 \dots t_n : B$ and $t_1, \dots, t_n$ are SN, then $\langle \Gamma; x t_1 \dots t_n\rangle \in \mathrm{Red}_B$.
3. If $\langle \Gamma; s\{t/x\} t_1 \dots t_n\rangle \in \mathrm{Red}_B$, $\Gamma \vdash t : A$ and t is SN, then $\langle \Gamma; (\lambda x^A\!.s) t t_1 \dots t_n\rangle \in \mathrm{Red}_B$.

Proof. Points 1–3 are proved simultaneously by induction on the type B. If $B = X$ then Point 1 is by definition of Red_X, for Points 2–3 see Exercise 14, p. 24. Let $B = C \Rightarrow D$.

1. Let $z \notin \mathrm{dom}(\Gamma)$, so $\langle \Gamma, z : C;\ z\rangle \in \mathrm{Red}_C$ by the induction hypothesis of Point 2 applied to C. As $\langle \Gamma; t\rangle \in \mathrm{Red}_{C \Rightarrow D}$, then $\langle \Gamma, z : C; tz\rangle \in \mathrm{Red}_D$ and hence tz is SN by the induction hypothesis of Point 1 applied to D; thus t is SN too.
2. Let $\langle \Gamma, \Delta;\ t\rangle \in \mathrm{Red}_C$, so t is SN by induction hypothesis of Point 1 applied to C; as $\Gamma, \Delta \vdash x t_1 \dots t_n t : D$ is derivable, $\langle \Gamma, \Delta; x t_1 \dots t_n t\rangle \in \mathrm{Red}_D$ by induction hypothesis of Point 2 applied to A; hence, $\langle \Gamma; x t_1 \dots t_n\rangle \in \mathrm{Red}_B$ by definition of $\mathrm{Red}_{C \Rightarrow D}$.
3. Let $\langle \Gamma, \Delta;\ r\rangle \in \mathrm{Red}_C$, so $\langle \Gamma, \Delta; s\{t/x\} t_1 \dots t_n r\rangle \in \mathrm{Red}_D$ and hence, by the induction hypothesis, $\langle \Gamma, \Delta; (\lambda x^A\!.s) t t_1 \dots t_n r\rangle \in \mathrm{Red}_D$; thus, $\langle \Gamma; (\lambda x^A\!.s) t t_1 \dots t_n\rangle \in \mathrm{Red}_B$. □

Fixing the reducibility candidate method

Solution: Let us take the environment into account when defining Red_A, for all types A.

$$\mathrm{Red}_X = \{\langle\Gamma; t\rangle \mid t \text{ is SN},\ \Gamma \vdash t : X\}$$
$$\mathrm{Red}_{A\Rightarrow B} = \{\langle\Gamma; s\rangle \mid \Gamma \vdash s : A \Rightarrow B,\ \langle\Gamma,\Delta;\ st\rangle \in \mathrm{Red}_B \text{ for all } \langle\Gamma,\Delta;\ t\rangle \in \mathrm{Red}_A\}$$

Lemma

1. If $\langle\Gamma; t\rangle \in \mathrm{Red}_B$ then t is SN.
2. If $\Gamma \vdash x t_1 \dots t_n : B$ and $t_1, \dots, t_n$ are SN, then $\langle\Gamma; x t_1 \dots t_n\rangle \in \mathrm{Red}_B$.
3. If $\langle\Gamma; s\{t/x\} t_1 \dots t_n\rangle \in \mathrm{Red}_B$, $\Gamma \vdash t : A$ and t is SN, then $\langle\Gamma; (\lambda x^A.s) t t_1 \dots t_n\rangle \in \mathrm{Red}_B$.

Proof. Points 1–3 are proved simultaneously by induction on the type B. If $B = X$ then Point 1 is by definition of Red_X, for Points 2–3 see Exercise 14, p. 24. Let $B = C \Rightarrow D$.

1. Let $z \notin \mathrm{dom}(\Gamma)$, so $\langle\Gamma, z : C;\ z\rangle \in \mathrm{Red}_C$ by the induction hypothesis of Point 2 applied to C. As $\langle\Gamma; t\rangle \in \mathrm{Red}_{C\Rightarrow D}$, then $\langle\Gamma, z : C; tz\rangle \in \mathrm{Red}_D$ and hence tz is SN by the induction hypothesis of Point 1 applied to D; thus t is SN too.
2. Let $\langle\Gamma,\Delta;\ t\rangle \in \mathrm{Red}_C$, so t is SN by induction hypothesis of Point 1 applied to C; as $\Gamma,\Delta \vdash x t_1 \dots t_n t : D$ is derivable, $\langle\Gamma,\Delta; x t_1 \dots t_n t\rangle \in \mathrm{Red}_D$ by induction hypothesis of Point 2 applied to A; hence, $\langle\Gamma; x t_1 \dots t_n\rangle \in \mathrm{Red}_B$ by definition of $\mathrm{Red}_{C\Rightarrow D}$.
3. Let $\langle\Gamma,\Delta;\ r\rangle \in \mathrm{Red}_C$, so $\langle\Gamma,\Delta; s\{t/x\} t_1 \dots t_n r\rangle \in \mathrm{Red}_D$ and hence, by the induction hypothesis, $\langle\Gamma,\Delta; (\lambda x^A.s) t t_1 \dots t_n r\rangle \in \mathrm{Red}_D$; thus, $\langle\Gamma; (\lambda x^A.s) t t_1 \dots t_n\rangle \in \mathrm{Red}_B$. □

Fixing the reducibility candidate method

Solution: Let us take the environment into account when defining Red_A, for all types A.

$$\mathrm{Red}_X = \{\langle \Gamma; t\rangle \mid t \text{ is SN},\ \Gamma \vdash t : X\}$$
$$\mathrm{Red}_{A\Rightarrow B} = \{\langle \Gamma; s\rangle \mid \Gamma \vdash s : A \Rightarrow B,\ \langle \Gamma, \Delta;\ st\rangle \in \mathrm{Red}_B \text{ for all } \langle \Gamma, \Delta;\ t\rangle \in \mathrm{Red}_A\}$$

Lemma

1. If $\langle \Gamma; t\rangle \in \mathrm{Red}_B$ then t is SN.
2. If $\Gamma \vdash x t_1 \dots t_n : B$ and $t_1, \dots, t_n$ are SN, then $\langle \Gamma; x t_1 \dots t_n\rangle \in \mathrm{Red}_B$.
3. If $\langle \Gamma; s\{t/x\} t_1 \dots t_n\rangle \in \mathrm{Red}_B$, $\Gamma \vdash t : A$ and t is SN, then $\langle \Gamma; (\lambda x^A.s) t t_1 \dots t_n\rangle \in \mathrm{Red}_B$.

Proof. Points 1–3 are proved simultaneously by induction on the type B. If $B = X$ then Point 1 is by definition of Red_X, for Points 2–3 see Exercise 14, p. 24. Let $B = C \Rightarrow D$.

1. Let $z \notin \mathrm{dom}(\Gamma)$, so $\langle \Gamma, z : C;\ z\rangle \in \mathrm{Red}_C$ by the induction hypothesis of Point 2 applied to C. As $\langle \Gamma; t\rangle \in \mathrm{Red}_{C\Rightarrow D}$, then $\langle \Gamma, z : C; tz\rangle \in \mathrm{Red}_D$ and hence tz is SN by the induction hypothesis of Point 1 applied to D; thus t is SN too.
2. Let $\langle \Gamma, \Delta;\ t\rangle \in \mathrm{Red}_C$, so t is SN by induction hypothesis of Point 1 applied to C; as $\Gamma, \Delta \vdash x t_1 \dots t_n t : D$ is derivable, $\langle \Gamma, \Delta; x t_1 \dots t_n t\rangle \in \mathrm{Red}_D$ by induction hypothesis of Point 2 applied to A; hence, $\langle \Gamma; x t_1 \dots t_n\rangle \in \mathrm{Red}_B$ by definition of $\mathrm{Red}_{C\Rightarrow D}$.
3. Let $\langle \Gamma, \Delta;\ r\rangle \in \mathrm{Red}_C$, so $\langle \Gamma, \Delta; s\{t/x\} t_1 \dots t_n r\rangle \in \mathrm{Red}_D$ and hence, by the induction hypothesis, $\langle \Gamma, \Delta; (\lambda x^A.s) t t_1 \dots t_n r\rangle \in \mathrm{Red}_D$; thus, $\langle \Gamma; (\lambda x^A.s) t t_1 \dots t_n\rangle \in \mathrm{Red}_B$. □

Strong normalization proved via reducibility candidates

Rmk. In the previous lemma, Point 1 needs Point 2 in its proof, and vice versa.
Point 3 is independent of Points 1–2 and is used in the proof of the lemma below.

Lemma (Substitution)

If $x_1 : B_1, \dots, x_n : B_n \vdash t : A$ and $\langle \Gamma; s_i \rangle \in \mathrm{Red}_{B_i}$, then $\langle \Gamma; t\{s_1/x_1, \dots, s_n/x_n\} \rangle \in \mathrm{Red}_A$.

Proof. By structural induction on the term t, using Point 3 above (exercise!). □

Theorem (Strong normalization of the simply typed λ-calculus)

Every typed term in the simply typed λ-calculus is SN.

Proof. Let $x_1 : B_1, \dots, x_n : B_n \vdash t : A$ be derivable. Let $\Gamma = x_1 : B_1, \dots, x_n : B_n$ and $s_i = x_i$ for all $1 \leq i \leq n$, hence $\langle \Gamma; s_i \rangle \in \mathrm{Red}_{B_i}$ by Point 2 of the lemma on p. 19, for all $1 \leq i \leq n$. By the substitution lemma above, $\langle \Gamma; t \rangle = \langle \Gamma; t\{s_1/x_1, \dots, s_n/x_n\} \rangle \in \mathrm{Red}_A$. By Point 1 of the lemma on p. 19, t is SN. □

Strong normalization proved via reducibility candidates

Rmk. In the previous lemma, Point 1 needs Point 2 in its proof, and vice versa. Point 3 is independent of Points 1–2 and is used in the proof of the lemma below.

Lemma (Substitution)

If $x_1 : B_1, \ldots, x_n : B_n \vdash t : A$ and $\langle \Gamma; s_i \rangle \in \mathrm{Red}_{B_i}$, then $\langle \Gamma; t\{s_1/x_1, \ldots, s_n/x_n\} \rangle \in \mathrm{Red}_A$.

Proof. By structural induction on the term t, using Point 3 above (exercise!). □

Theorem (Strong normalization of the simply typed λ-calculus)

Every typed term in the simply typed λ-calculus is SN.

Proof. Let $x_1 : B_1, \ldots, x_n : B_n \vdash t : A$ be derivable. Let $\Gamma = x_1 : B_1, \ldots, x_n : B_n$ and $s_i = x_i$ for all $1 \leq i \leq n$, hence $\langle \Gamma; s_i \rangle \in \mathrm{Red}_{B_i}$ by Point 2 of the lemma on p. 19, for all $1 \leq i \leq n$. By the substitution lemma above, $\langle \Gamma; t \rangle = \langle \Gamma; t\{s_1/x_1, \ldots, s_n/x_n\} \rangle \in \mathrm{Red}_A$. By Point 1 of the lemma on p. 19, t is SN. □

Strong normalization proved via reducibility candidates

Rmk. In the previous lemma, Point 1 needs Point 2 in its proof, and vice versa.
Point 3 is independent of Points 1–2 and is used in the proof of the lemma below.

Lemma (Substitution)

If $x_1 : B_1, \dots, x_n : B_n \vdash t : A$ and $\langle \Gamma; s_i \rangle \in \mathrm{Red}_{B_i}$, then $\langle \Gamma; t\{s_1/x_1, \dots, s_n/x_n\} \rangle \in \mathrm{Red}_A$.

Proof. By structural induction on the term t, using Point 3 above (exercise!). □

Theorem (Strong normalization of the simply typed λ-calculus)

Every typed term in the simply typed λ-calculus is SN.

Proof. Let $x_1 : B_1, \dots, x_n : B_n \vdash t : A$ be derivable. Let $\Gamma = x_1 : B_1, \dots, x_n : B_n$ and $s_i = x_i$ for all $1 \leq i \leq n$, hence $\langle \Gamma; s_i \rangle \in \mathrm{Red}_{B_i}$ by Point 2 of the lemma on p. 19, for all $1 \leq i \leq n$. By the substitution lemma above, $\langle \Gamma; t \rangle = \langle \Gamma; t\{s_1/x_1, \dots, s_n/x_n\} \rangle \in \mathrm{Red}_A$. By Point 1 of the lemma on p. 19, t is SN. □

Outline

What have we learned today?

1. How to decorate derivations in natural deduction for minimal logic with λ-terms.
2. The procedure of β-reduction on λ-terms.
3. Church and Curry styles for the simply typed λ-calculus.
4. The Curry–Howard correspondence between natural deduction for minimal logic and the simply typed λ-calculus.
5. Some properties of the simply typed λ-calculus (subject reduction, normalization).
6. The proof of strong normalization via reducibility candidates.

Questions?

What have we learned today?

1. How to decorate derivations in natural deduction for minimal logic with λ-terms.
2. The procedure of β-reduction on λ-terms.
3. Church and Curry styles for the simply typed λ-calculus.
4. The Curry–Howard correspondence between natural deduction for minimal logic and the simply typed λ-calculus.
5. Some properties of the simply typed λ-calculus (subject reduction, normalization).
6. The proof of strong normalization via reducibility candidates.

Questions?

What have we learned today?

1. How to decorate derivations in natural deduction for minimal logic with λ-terms.
2. The procedure of β-reduction on λ-terms.
3. Church and Curry styles for the simply typed λ-calculus.
4. The Curry–Howard correspondence between natural deduction for minimal logic and the simply typed λ-calculus.
5. Some properties of the simply typed λ-calculus (subject reduction, normalization).
6. The proof of strong normalization via reducibility candidates.

Questions?

What have we learned today?

1. How to decorate derivations in natural deduction for minimal logic with λ-terms.
2. The procedure of β-reduction on λ-terms.
3. Church and Curry styles for the simply typed λ-calculus.
4. The Curry–Howard correspondence between natural deduction for minimal logic and the simply typed λ-calculus.
5. Some properties of the simply typed λ-calculus (subject reduction, normalization).
6. The proof of strong normalization via reducibility candidates.

Questions?

What have we learned today?

1. How to decorate derivations in natural deduction for minimal logic with λ-terms.
2. The procedure of β-reduction on λ-terms.
3. Church and Curry styles for the simply typed λ-calculus.
4. The Curry–Howard correspondence between natural deduction for minimal logic and the simply typed λ-calculus.
5. Some properties of the simply typed λ-calculus (subject reduction, normalization).
6. The proof of strong normalization via reducibility candidates.

Questions?

What have we learned today?

1. How to decorate derivations in natural deduction for minimal logic with λ-terms.
2. The procedure of β-reduction on λ-terms.
3. Church and Curry styles for the simply typed λ-calculus.
4. The Curry–Howard correspondence between natural deduction for minimal logic and the simply typed λ-calculus.
5. Some properties of the simply typed λ-calculus (subject reduction, normalization).
6. The proof of strong normalization via reducibility candidates.

Questions?

Exercises

1. Find the simply typed λ-terms (in Curry-style and Church-style) associated with the derivations in ND found for the facts below (see Exercise 1 from Day 1).
   1. $\vdash X \Rightarrow ((X \Rightarrow Y) \Rightarrow Y)$.
   2. $(X \Rightarrow Y) \Rightarrow (X \Rightarrow Z) \vdash Y \Rightarrow X \Rightarrow Z$.
   3. $(X \Rightarrow Y) \Rightarrow X \vdash Y \Rightarrow X$.
   4. $X \Rightarrow (Y \Rightarrow Z) \vdash Y \Rightarrow X \Rightarrow Z$.
   5. $X \Rightarrow Y \Rightarrow Z, X \Rightarrow Y \vdash X \Rightarrow Z$.
   6. $(X \Rightarrow X) \Rightarrow Y \vdash (Y \Rightarrow Z) \Rightarrow Z$.
2. Perform all possible β-reduction steps from the λ-term decorating the derivation $\mathcal{D}$ in ND on p. 24 of Day 1, until you get a β-normal form. Is it always the same? Compare it with the normal derivation obtained by cut-elimination steps from $\mathcal{D}$.
3. Prove rigorously the following facts ($f^n x = \overbrace{f(\dots(f}^{n \text{ times } f}\, x)\dots)$ for any $n \in \mathbb{N}$):
   1. $\lambda x.xx$ is untypable in Curry-style, $\lambda x^A.xx$ is untypable in Church-style for any type A;
   2. in Church-style, $\lambda f^Y.\lambda x^X.f^n x$ is not typable for any $n > 0$ but $\lambda f^Y.\lambda x^X.x$ is typable;
   3. $\lambda f.\lambda x.f^n x$ is typable in Curry-style, for all $n \in \mathbb{N}$.
4. Prove that if t is typable in Church or Curry style, then so is every subterm of t.
5. Prove rigorously the propositions on pp. 9 and 12, the lemma and theorems on p. 13.
6. Prove rigorously the lemma and the theorems on p. 13.
7. Let $A\{B/X\}$ be the type obtained from the type A by substituting B for each occurrence of the ground type X. Let $\Gamma\{A/X\}$ be its generalization to environments. Show that if $\Gamma \vdash t : A$ is derivable in Curry-style, then so is $\Gamma\{B/X\} \vdash t : A\{B/X\}$.
8. Is the previous point valid in Church-style? What change is needed to make it true?

More Exercises

9. Prove that $(\lambda x.(\lambda y.y))\lambda z.zz$ is not typable (in Curry-style). Deduce that subject expansion (see p. 13) does not hold in the simply typed λ-calculus.
10. Prove that if $\langle \Gamma; t \rangle \in \mathrm{Red}_B$, then $\langle \Gamma, x\!:\!A;\ t \rangle \in \mathrm{Red}_B$, by induction on the type B.
11. Prove rigorously the lemma on p. 20.
12. Define four ARSs $(A, \to)$: in the first $\to$ is normalizing but not strongly normalizing, in the second $\to$ is not normalizing, in the third $\to$ is strongly normalizing, in the fourth $\to$ is not confluent but every $t \in A$ has a unique normal form.
13. In a ARS $(A, \to)$, prove that $t \in A$ is SN iff for every $t' \in A$, if $t \to t'$ then t' is SN.
14. Prove the following facts for untyped terms:
    1. for all $n \in \mathbb{N}$, if $t_1, \dots, t_n$ are SN, then so is $xt_1 \dots t_n$;
    2. if t is SN, then so is $\lambda x.s$;
    3. for all $n \in \mathbb{N}$, if t and $s\{t/x\}t_1 \dots t_n$ are SN, then so is $(\lambda x.s)tt_1 \dots t_n$;
    4. for all $n \in \mathbb{N}$, if $x \in \mathrm{fv}(s)$ and $s\{t/x\}t_1 \dots t_n$ is SN then so is $(\lambda x.s)tt_1 \dots t_n$.
15. Prove that if t can be constructed by applying $n \in \mathbb{N}$ times the rules below, then the maximal length of the reduction sequences from t to its β-normal form is $\leq n$.

$$\frac{n \in \mathbb{N} \quad (t_i \text{ is SN})_{1 \leq i \leq n}}{xt_1 \dots t_n \text{ is SN}} \qquad \frac{t \text{ is SN}}{\lambda x.t \text{ is SN}} \qquad \frac{n \in \mathbb{N} \quad t \text{ is SN} \quad s\{t/x\}t_1 \dots t_n \text{ is SN}}{(\lambda x.s)tt_1 \dots t_n \text{ is SN}}$$

Deduce that the set of SN untyped terms is the least set closed under those 3 rules.

Bibliography

- For more about the simply typed λ-calculus:

Ralph Loader. *Notes on Simply Typed λ-Calculus*. Technical report ECS-LFCS-98-381, University of Edinburgh, 1998. http://www.lfcs.inf.ed.ac.uk/reports/98/ECS-LFCS-98-381/ECS-LFCS-98-381.pdf. [Chapters 1–3]

Henk Barendregt. *Lambda Calculi with Types*. In S. Abramsky et al. (eds), Handbook of Logic in Computer Science; vol. 2, 117-309, 1992. https://repository.ubn.ru.nl/bitstream/handle/2066/17231/17231.pdf [Chapter 3]

- For more about the Curry-Howard correspondence:

Jean-Yves Girard, Yves Lafont, Paul Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science, Vol. 7, Cambridge University Press, 1989. https://www.paultaylor.eu/stable/prot.pdf. [Chapters 2–4, 6]

- For a combinatorial proof of normalization for the simply typed λ-calculus:

Pablo Barenbaum, Cristian Sottile. Two Decreasing Measures for Simply Typed λ-Terms. FSCD 2023, LIPIcs, vol. 260, 11:1–11:19, 2023. https://doi.org/10.4230/LIPIcs.FSCD.2023.11.