Viewing λ-terms through Maps

Masahiko Sato

Graduate School of Informatics, Kyoto University

Joint work with Randy Pollack,
Helmut Schwichtenberg and Takafumi Sakurai

IFIP WG 2.2 meeting, Lisbon, September 25, 2013

Motivations

- The notion of binding is fundamental in mathematics (not only in proof theory and $\boldsymbol{\lambda}$-calculus.)
- What are lambda terms?
- How are they constructed?
- Can we define $\boldsymbol{\lambda}$-terms without using the notion of equivalence relation?
- A good definition of lambda term will contribute to the design and implementation of proof assistants.

Motivations

- The notion of binding is fundamental in mathematics (not only in proof theory and λ-calculus.)
- What are lambda terms?
- How are they constructed?
- Can we define λ-terms without using the notion of equivalence relation?
- A good definition of lambda term will contribute to the design and implementation of proof assistants.

We have formally verified all the technical results in this work in the proof assistants Minlog and Isabelle.

Our paper will soon appear in de Bruijn special issue of Indag. Math.

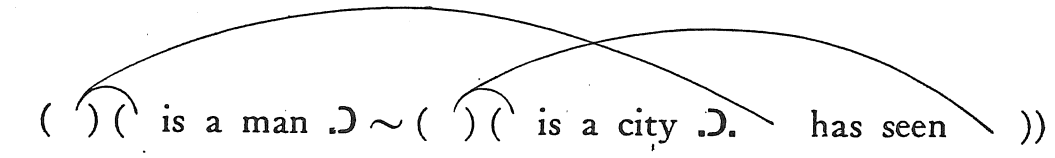
Good points of our approach

- The inductive structure of the terms is nicer compared to other approaches.
- Can define closed lambda terms directly without first defining the lambda terms containing free parameters.
- Can use the same technique to define sentences without first defining formulas containing free parameters.
- A special generic constant $\square$ must be included as a term, however.

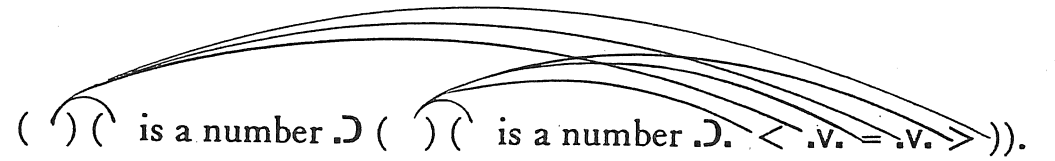
History

- 1930's. Church defined raw lambda terms ($\mathbf{\Lambda}$) and defined α-equivalence relation on them.
- 1940. Quine defined graphical representation of lambda terms. Later, Bourbaki (1954) rediscovered it.
- 1972. de Bruijn defined representation of lambda terms by indices ($\mathbb{D}$).
- 1980. S. defined representation of lambda terms by map and skeleton (precursor of $\mathbb{L}$).
- 2013. This talk clarifies the relationship among the above four representations.

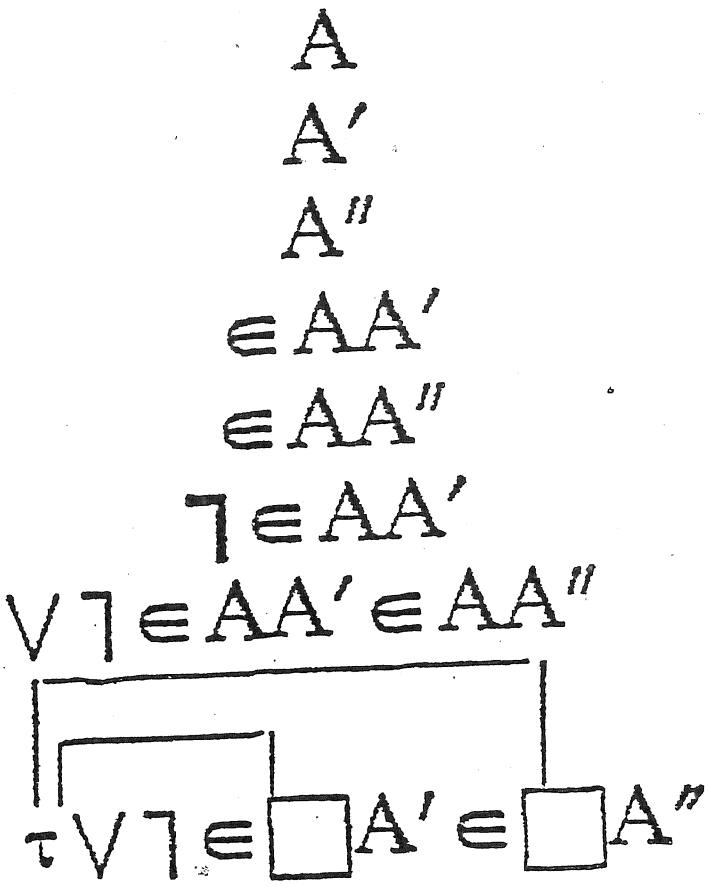
sideration for established usage, the "variation" connoted belongs to a vague metaphor which is best forgotten. The variables have no meaning beyond the pronominal sort of meaning which is reflected in translations such as (20); they serve merely to indicate cross-references to various positions of quantification. Such cross-references could be made instead by curved lines or *bonds*; e.g., we might render (27) thus:



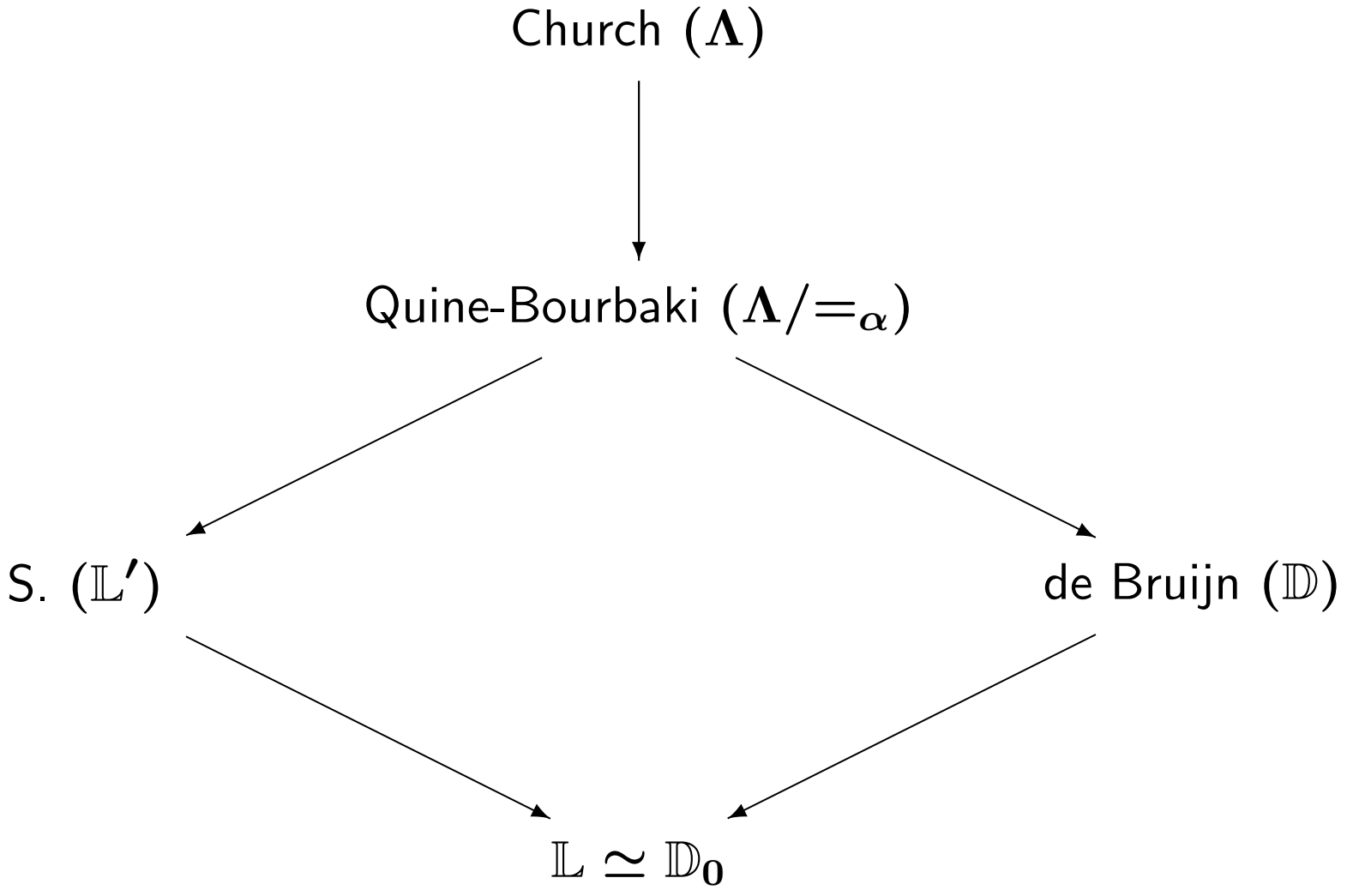
and (26) thus:



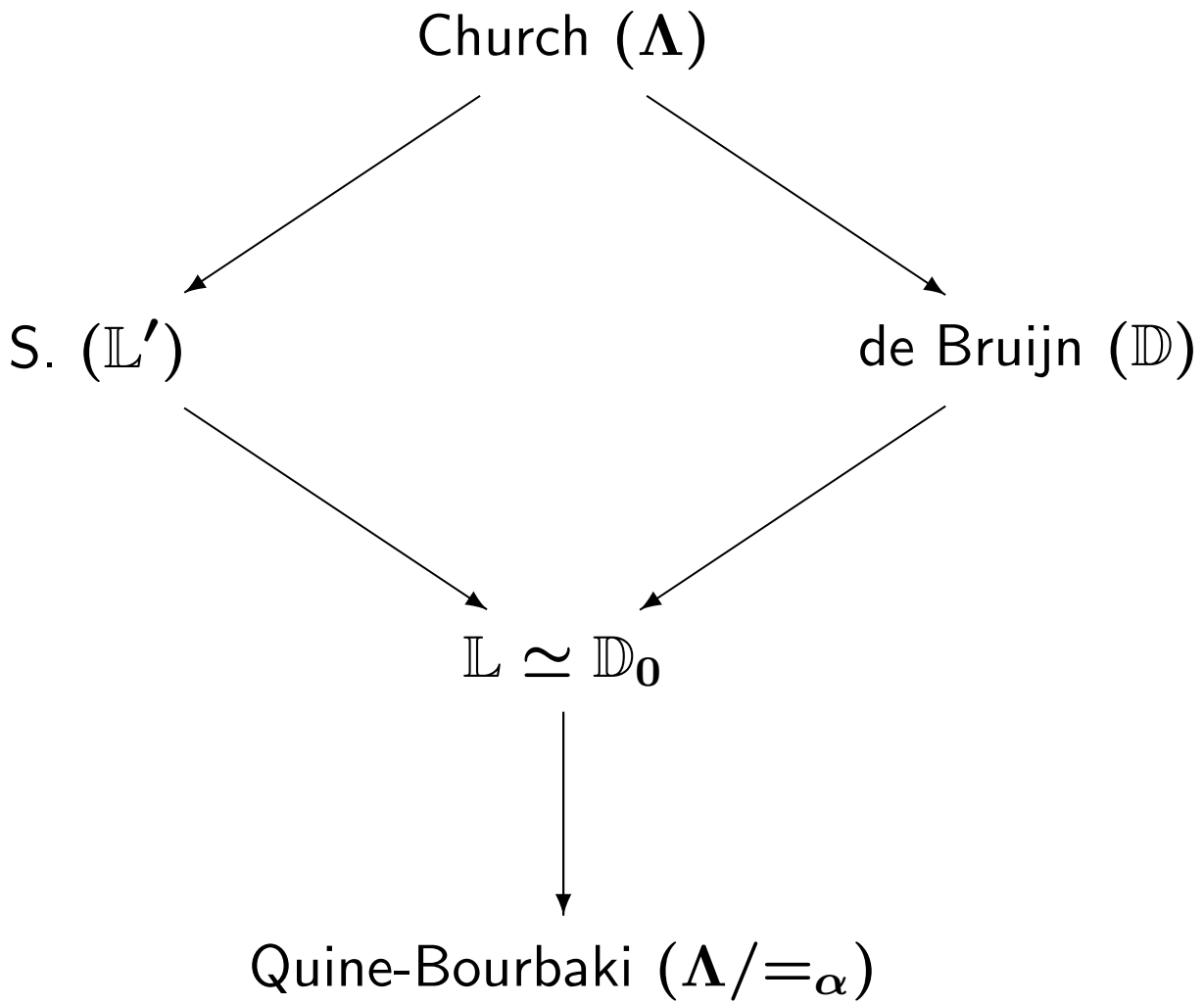
But these "quantificational diagrams" are too cumbersome to recommend themselves as a practical notation; hence the use of variables.


A
A′
A″
∈AA′
∈AA″
¬∈AA′
∨¬∈AA′∈AA″
τ∨¬∈□A′∈□A″

History (cont.)



Logical View



▸ example

Summary of the talk

Three datatypes

We will relate the three datatypes $(\mathbf{\Lambda}, \mathbb{L}, \mathbb{D})$ of expressions introduced by Church, S. and de Bruijn.

$\mathbf{\Lambda}$ = The datatype of raw $\boldsymbol{\lambda}$-terms.

$\mathbb{L}$ = The datatype of lambda-expressions.

$\mathbb{D}$ = The datatype of de Bruijn expressions.

Three types of abstractions

$\mathbf{\Lambda}$:abstraction by parameters $\boldsymbol{x} \in \mathbb{X}$.

$\mathbb{L}$:abstraction by maps $\boldsymbol{m} \in \mathbb{M}$.

$\mathbb{D}$:abstraction by indices $\boldsymbol{i} \in \mathbb{I}$.

Summary of the talk (cont.)

$$\boldsymbol{K}, \boldsymbol{L} \in \boldsymbol{\Lambda} ::= \boldsymbol{x} \mid \Box \mid \mathsf{app}(\boldsymbol{K}, \boldsymbol{L}) \mid \mathsf{lam}(\boldsymbol{x}, \boldsymbol{K}).$$
$$\boldsymbol{M}, \boldsymbol{N} \in \mathbb{L} ::= \boldsymbol{x} \mid \Box \mid \mathsf{app}(\boldsymbol{M}, \boldsymbol{N}) \mid \mathsf{mask}(\boldsymbol{m}, \boldsymbol{M}) \quad (\boldsymbol{m} \mid \boldsymbol{M}).$$
$$\boldsymbol{D}, \boldsymbol{E} \in \mathbb{D} ::= \boldsymbol{x} \mid \Box \mid \mathsf{app}(\boldsymbol{D}, \boldsymbol{E}) \mid \boldsymbol{i} \mid \mathsf{bind}(\boldsymbol{D}).$$
$$\boldsymbol{x} \in \mathbb{X}.$$
$$\boldsymbol{m} \in \mathbb{M}.$$
$$\boldsymbol{i} \in \mathbb{I}.$$

$\Box$ (called a box) is a special constant denoting a hole to be filled with lambda expressions.

Datatypes and constructors

1. Every object of a datatype is uniqulely constructed by an application of a constructor function c to already constructed objects:
$$d = c(d_1, \ldots, d_k).$$
2. Every construtor has a unique type:
$$c : D_1 \times \cdots \times D_k \to D,$$
where c can be a partial function.
3. Given $d_1 \in D_1, \ldots, d_k \in D_k$, it is primitive recursively decidable if c can be successfully applied to these objects.

Hence, every object can be represented by a finite tree and we can naturally associcate an induction principle with every datatatype.

The notion of map

The notion of map is a generalization of the notion of occurrence of a symbol in syntactic expressions such as formulas or lambda terms.

Plan of the talk

- Part 1. $\mathbb{L}$.
- Part 2. $\mathbf{\Lambda}$. Will show $\mathbb{L} \simeq \mathbf{\Lambda}/\equiv_{\boldsymbol{\alpha}}$.
- Part 3. $\mathbb{D}$. Will show $\mathbb{L} \simeq \mathbb{D}_{\mathbf{0}}$.

map/skeleton functions will play important roles in all the 3 parts.

Part 1

$\mathbb{L}$

The Datatype of Lambda-exressions

The Datatype $\mathbb{M}$ of Maps

$$\frac{}{0 \in \mathbb{M}} \qquad \frac{}{1 \in \mathbb{M}}$$

$$\frac{m \in \mathbb{M} \quad n \in \mathbb{M} \quad m \neq 0 \text{ or } n \neq 0}{\mathsf{cons}(m, n) \in \mathbb{M}}$$

Note that

$$\mathsf{cons} : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$$

is a partial function.
We will write $(m\ n)$ or mn for $\mathsf{cons}(m, n)$.

Orthogonality relation on maps

$$\frac{}{m \perp 0} \qquad \frac{}{0 \perp n} \qquad \frac{m \perp n \quad m' \perp n'}{mm' \perp nn'}$$

Example: $(1\ 0) \perp (0\ 1)$ but not $(1\ 1) \perp (0\ 1)$.

The Datatype $\mathbb{X}$ of Parameters

We assume a countably infinite set $\mathbb{X}$ of parameters.
We will write x, y, z for parameters.
We assume that equality relation on $\mathbb{X}$ is decidable.

The Datatype $\mathbb{L}$ and the Divisibility Relation

$$\frac{}{x \in \mathbb{L}}\ \textit{par} \qquad \frac{}{\Box \in \mathbb{L}}\ \textit{box}$$

$$\frac{M \in \mathbb{L} \quad N \in \mathbb{L}}{\mathsf{app}(M, N) \in \mathbb{L}}\ \mathsf{app} \qquad \frac{m \in \mathbb{M} \quad M \in \mathbb{L} \quad m \mid M}{\mathsf{mask}(m, M) \in \mathbb{L}}\ \mathsf{mask}$$

$$\frac{}{0 \mid x} \qquad \frac{}{0 \mid \Box} \qquad \frac{}{1 \mid \Box}$$

$$\frac{m \mid M \quad n \mid N}{\mathsf{mapp}(m, n) \mid \mathsf{app}(M, N)} \qquad \frac{m \mid N \quad n \mid N \quad m \perp n}{m \mid \mathsf{mask}(n, N)}$$

The Datatype $\mathbb{L}$ of lambda-expressions (cont.)

Notational Convention

- We use $\boldsymbol{M}, \boldsymbol{N}, \boldsymbol{P}$ as metavariables ranging over lambda-expressions.
- We write $(\boldsymbol{M}\ \boldsymbol{N})$ and also $\boldsymbol{MN}$ for app$(\boldsymbol{M}, \boldsymbol{N})$.
- We write $\boldsymbol{m}\backslash\boldsymbol{M}$ for mask$(\boldsymbol{m}, \boldsymbol{M})$.
- A lambda-expression of the form $\boldsymbol{m}\backslash\boldsymbol{M}$ is called an abstract.
- We use $\boldsymbol{A}, \boldsymbol{B}$ as metavariables ranging over abstarcts, and write $\mathbb{A}$ for the subset of $\mathbb{L}$ consisting of all the abstracts.

map : $\mathbb{X} \times \mathbb{L} \to \mathbb{M}$ and skel : $\mathbb{X} \times \mathbb{L} \to \mathbb{L}$

We write M_x for map(x, M), and M^x for skel(x, M).

$$y_x := \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{if } x \neq y. \end{cases}$$

$$\Box_x := 0$$

$$(M\ N)_x := (M_x\ N_x).$$

$$(m \backslash M)_x := M_x.$$

$$y^x := \begin{cases} \Box & \text{if } x = y, \\ y & \text{if } x \neq y. \end{cases}$$

$$\Box^x := \Box$$

$$(M\ N)^x := (M^x\ N^x).$$

$$(m \backslash M)^x := m \backslash M^x.$$

Lambda Abstraction in $\mathbb{L}$

We define lam : $\mathbb{X} \times \mathbb{L} \to \mathbb{L}$ by:

$$\mathsf{lam}(x, M) := M_x \backslash M^x.$$

Examples. We assume that x, y and z are distinct parameters.

$$\begin{aligned}
\mathsf{lam}(x, x) &= 1\backslash\Box. \\
\mathsf{lam}(x, y) &= 0\backslash y. \\
\mathsf{lam}(x, \mathsf{lam}(y, x)) &= \mathsf{lam}(x, 0\backslash x) \\
&= 1\backslash 0\backslash\Box. \\
\mathsf{lam}(x, \mathsf{lam}(y, y)) &= \mathsf{lam}(x, \mathsf{lam}(1, \Box)) \\
&= 0\backslash 1\backslash\Box. \\
\mathsf{lam}(x, \mathsf{lam}(y, \mathsf{lam}(z, (xz\ yz)))) &= \\
&(10\ 00)\backslash(00\ 10)\backslash(01\ 01)\backslash(\Box\Box\ \Box\Box)
\end{aligned}$$

Hole Filling and Instantiation

We write $M_m[P]$ for the result of filling boxes (holes) in M specified by map m with P. $M_m[P]$ is defined only if $m \mid M$. We write $A \blacktriangledown P$ for the result of instantiating abstract A with P.

$$\begin{aligned}
\text{fill}: \quad & \mathbb{L} \times \mathbb{M} \times \mathbb{L} \to \mathbb{L} \\
\Box_1[P] &:= P. \\
\Box_0[P] &:= \Box. \\
x_0[P] &:= x. \\
(M\ N)_{(m\ n)}[P] &:= (M_m[P]\ N_n[P]). \\
(n\backslash N)_m[P] &:= n\backslash N_m[P].
\end{aligned}$$

$$\begin{aligned}
\blacktriangledown: \quad & \mathbb{A} \times \mathbb{L} \to \mathbb{L}_{\mathbf{\Lambda}} \\
(m\backslash M) \blacktriangledown P &:= M_m[P].
\end{aligned}$$

Substitution

We can now define substitution operation:
subst : $\mathbb{L} \times \mathbb{X} \times \mathbb{L} \to \mathbb{L}$ as follows.

$$[P/x]y := \begin{cases} P & \text{if } x = y, \\ y & \text{if } x \neq y. \end{cases}$$

$$[P/x]\Box := \Box.$$

$$[P/x](M\ N) := ([P/x]M\ [P/x]N).$$

$$[P/x](m\backslash M) := (m\backslash [P/x]M).$$

subst enjoys the following property.

$$[P/x]M = \mathsf{lam}(x, M)\blacktriangledown P.$$

Substitution (cont.)

Example.

$$\begin{aligned}
[y/x]\mathsf{lam}(y, yx) &= [y/x](10\backslash\Box x) \\
&= 10\backslash[y/x](\Box\ x) \\
&= 10\backslash([y/x]\Box\ [y/x]x) \\
&= 10\backslash\Box y \\
&= \mathsf{lam}(z, zy)
\end{aligned}$$

Remark. By internalizing the substitution operation, we can easily get an explicit substitution calculus.

Substitution Lemma

If $x \neq y$ and $x \notin \mathrm{FP}(P)$, then

$$[P/y][N/x]M = [[P/y]N/x][P/y]M.$$

Proof. By induction on $M \in \mathbb{L}$. Here, we only treat the case where $M = m\backslash M'$.

$$\begin{aligned}
&[P/y][N/x]M \\
&= [P/y][N/x](m\backslash M') \\
&= m\backslash[P/y][N/x]M' \\
&= m\backslash[[P/y]N/x][P/y]M' && \text{(by IH)} \\
&= [[P/y]N/x][P/y](m\backslash M') \\
&= [[P/y]N/x][P/y]M.
\end{aligned}$$

The $\mathbb{L}_\beta$-calculus

$$\frac{}{\boldsymbol{AM} \to_\beta \boldsymbol{A}\blacktriangledown\boldsymbol{M}}\ \beta$$

$$\frac{\boldsymbol{M} \to_\beta \boldsymbol{M'}}{\boldsymbol{MN} \to_\beta \boldsymbol{M'N}}\ \text{appl} \qquad \frac{\boldsymbol{M} \in \mathbb{L}}{\boldsymbol{MN} \to_\beta \boldsymbol{MN'}}\ \text{appr}$$

$$\frac{\boldsymbol{M} \to_\beta \boldsymbol{N}}{\mathsf{lam}(\boldsymbol{x}, \boldsymbol{M}) \to_\beta \mathsf{lam}(\boldsymbol{x}, \boldsymbol{N})}\ \boldsymbol{\xi}$$

Remark. Traditional way of formulating β-conversion rule is:

$$(\mathsf{lam}(\boldsymbol{x}, \boldsymbol{M})\ \boldsymbol{N}) \to_\beta [\boldsymbol{N}/\boldsymbol{x}]\boldsymbol{M}.$$

Part 2

Λ

The Datatype of Raw Lambda-terms

The Datatype Λ of Raw λ-terms

$$\frac{}{x \in \Lambda}\ \textit{par} \qquad \frac{}{\Box \in \Lambda}\ \textit{box}$$

$$\frac{K \in \Lambda \quad L \in \Lambda}{\mathsf{app}(K, L) \in \Lambda}\ \mathsf{app} \qquad \frac{x \in \mathbb{X} \quad K \in \Lambda}{\mathsf{lam}(x, K) \in \Lambda}\ \mathsf{lam}$$

$$K, L \in \Lambda ::= x \mid \Box \mid \mathsf{app}(K, L) \mid \mathsf{lam}(x, K).$$

Remark. lam binds parameter x in M.

map : $\mathbb{X} \times \Lambda \to \mathbb{M}$ and skel : $\mathbb{X} \times \Lambda \to \Lambda$

$$y_x := \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{if } x \neq y. \end{cases}$$

$$\Box_x := 0.$$

$$(K\ L)_x := (K_x\ L_x).$$

$$\mathsf{lam}(y, K)_x := \begin{cases} 0 & \text{if } x = y, \\ K_x & \text{if } x \neq y. \end{cases}$$

$$y^x := \begin{cases} \Box & \text{if } x = y, \\ y & \text{if } x \neq y. \end{cases}$$

$$\Box^x := \Box.$$

$$(K\ L)^x := (K^x\ L^x).$$

$$\mathsf{lam}(y, K)^x := \begin{cases} \mathsf{lam}(y, K) & \text{if } x = y, \\ \mathsf{lam}(y, K^x) & \text{if } x \neq y. \end{cases}$$

Map and Skeleton (cont.)

$$
\begin{aligned}
& x \text{ does not occur free in } K \\
\Longleftrightarrow & K_x = 0 \\
\Longleftrightarrow & K^x = K
\end{aligned}
$$

Remark. This shows that the notion of map is a generalization of the notion of occurrence.

α-equivalence Relation

We define the α-equivalence relation, $=_\alpha$, using the map/skeleton functions.

$$\frac{}{x =_\alpha x} \qquad \frac{}{\square =_\alpha \square}$$

$$\frac{K =_\alpha K' \quad L =_\alpha L'}{KL =_\alpha K'L'} \qquad \frac{K_x = L_y \quad K^x =_\alpha L^y}{\mathsf{lam}(x, K) =_\alpha \mathsf{lam}(y, L)}$$

Remark. No renaming is needed in this definition, and it is easy to verify that this is indeed a decidable equivalence relation.

α-equivalence Relation

We can show that $\mathsf{lam}(x, \mathsf{lam}(y, yx)) =_\alpha \mathsf{lam}(y, \mathsf{lam}(x, xy))$ as follows.

$$\dfrac{01 = 01 \qquad \dfrac{10 = 10 \qquad \dfrac{\dfrac{}{\square =_\alpha \square} \quad \dfrac{}{\square =_\alpha \square}}{\square\square =_\alpha \square\square}}{\mathsf{lam}(y, y\square) =_\alpha \mathsf{lam}(x, x\square)}}{\mathsf{lam}(x, \mathsf{lam}(y, yx)) =_\alpha \mathsf{lam}(y, \mathsf{lam}(x, xy))}$$

Substitution

We think that it is more natural to define substitution as a relation (which is invariant under α-equivalence) than to define it as a function (using a choice function which chooses a fresh parameter).

But we skip the discussion here.

Interpretation of Λ in $\mathbb{L}$

We define the interpretation function $[\![-]\!] : \Lambda \to \mathbb{L}$ as follows.

$$
\begin{aligned}
[\![x]\!] &:= x. \\
[\![\Box]\!] &:= \Box. \\
[\![KL]\!] &:= [\![K]\!][\![L]\!]. \\
[\![\mathsf{lam}(x, K)]\!] &:= \mathsf{lam}(x, [\![K]\!]).
\end{aligned}
$$

Remark. Two raw λ-terms K and L are α-equivalent iff $[\![M]\!] = [\![N]\!]$.

Part 3

$\mathbb{D}$

The Datatype of de Bruijn-expressions

The Datatype $\mathbb{I}$ of Indices

We use natural numbers as indices.

$$i, j \in \mathbb{I} ::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \cdots$$

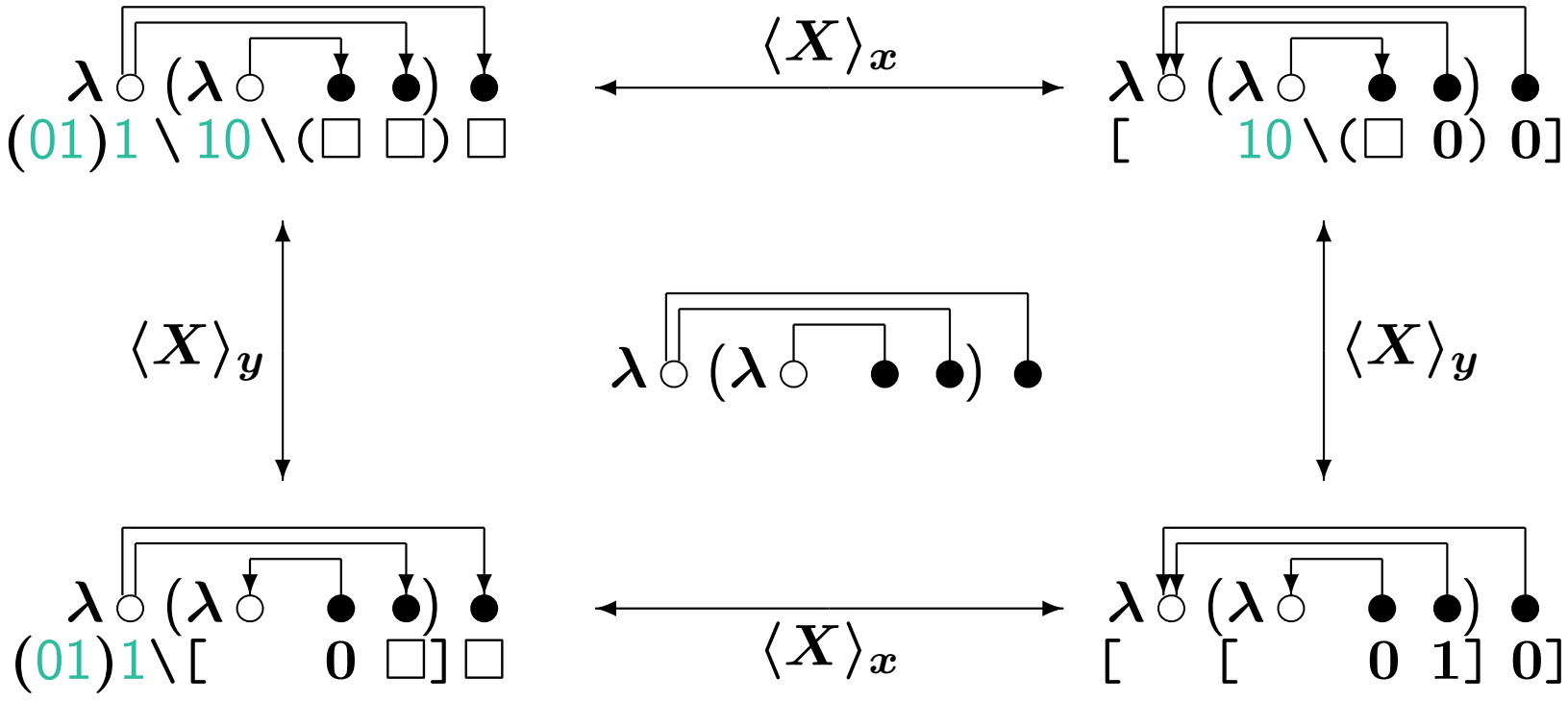
The Datatype $\mathbb{D}$ of de Bruijn-expressions

$$\frac{}{\boldsymbol{x} \in \mathbb{D}}\ \textit{par} \qquad \frac{}{\Box \in \mathbb{D}}\ \textit{box} \qquad \frac{}{\boldsymbol{i} \in \mathbb{D}}\ \textit{idx}$$

$$\frac{\boldsymbol{D} \in \mathbb{D} \quad \boldsymbol{E} \in \mathbb{D}}{\mathsf{app}(\boldsymbol{D}, \boldsymbol{E}) \in \mathbb{D}}\ \mathsf{app} \qquad \frac{\boldsymbol{D} \in \mathbb{D}}{\mathsf{bind}(\boldsymbol{D}) \in \mathbb{D}}\ \mathsf{bind}$$

We write $[\boldsymbol{D}]$ for $\mathsf{bind}(\boldsymbol{D})$.

Example: $\lambda x.\,(\lambda y.\,yx)x$



summary

Conclusion

- We have introduced a new datatype ($\mathbb{L}$) whose elements canonically represent the lambda terms.
- In particular, abstracts are represented as pairs of the map parts and the skeleteton parts of the abstracts.
- Substitution operation on $\mathbb{L}$ is a homomorphism, and can be computed by first-order term rewriting (explicit substitution without renaming).
- Induction principle on $\mathbb{L}$ is structural and follow the pattern of the inductive definition of the datatype.
- $\mathbb{L}$ is isomorphic to the datatype of raw $\boldsymbol{\lambda}$-terms modulo α-equivalence, and also is isomprphic to the datatype of de Bruijn-expressions. These isomorphisms respect substitution operations.
- We are almost finishing formal verifications of all the technical results in the proof assistants Isabelle/{HOL, Nominal} and Minlog.