Symbolic Expressions and Variable Binding
Lecture 4

Masahiko Sato

Graduate School of Informatics, Kyoto University

September 6–10, 2010

Plan of the 5 lectures

1. Overview
2. Traditional definition of Lambda terms
3. Lambda terms by de Bruijn indices
4. Lambda terms as abstract data type
5. Derivations as abstract data type

Plan of this lecture

- Map based lambda expressions
- ⟨`Map`⟩: Map
- ⟨`Sxp`⟩: Schematic lambda expressions
- ⟨`Exp`⟩: Lambda Expressions
- Translation from ⟨`Txp`⟩.
- Comparison with ⟨`dxp`⟩.

Outline of Map-based expressions

- We introduce the key concept of *map* ($\langle \texttt{Map} \rangle$) which has a binary tree structre.
- A map plays a role similar to an index in a de Bruijn expression.
- We define *abstract lambda expressions* ($\langle \texttt{Sxp} \rangle$) , which are almost like lambda expressions except that an abstract may contain *holes* which when filled with an expression becomes a real expression.
- Schematic lambda expressions may contain maps which are used to specify the positions of variables and local holes.
- Finally, we define *lambda expressions* ($\langle \texttt{Exp} \rangle$) using abstracts.

The class $\langle \texttt{Map} \rangle$

The mother class $\langle \texttt{Map} \rangle$ (maps) has the following creation methods:

$$\frac{}{(\texttt{zro}) : \langle \texttt{Map} \rangle}\ \texttt{zro} \qquad \frac{}{(\texttt{one}) : \langle \texttt{Map} \rangle}\ \texttt{one}$$

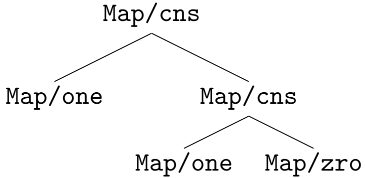
$$\frac{\boldsymbol{m} : \langle \texttt{Map} \rangle \quad \boldsymbol{n} : \langle \texttt{Map} \rangle}{(\texttt{cns}\ \boldsymbol{m}\ \boldsymbol{n}) : \langle \texttt{Map} \rangle}\ \texttt{cns}$$

The class ⟨Map⟩ (cont.)

An example

$$\dfrac{\dfrac{}{(\mathtt{one}) : \langle\mathtt{Map}\rangle}\;\mathtt{one} \qquad \dfrac{\dfrac{}{(\mathtt{one}) : \langle\mathtt{Map}\rangle}\;\mathtt{one} \quad \dfrac{}{(\mathtt{zro}) : \langle\mathtt{Map}\rangle}\;\mathtt{zro}}{(\mathtt{cns}\ (\mathtt{one})\ (\mathtt{zro})) : \langle\mathtt{Map}\rangle}\;\mathtt{cns}}{(\mathtt{cns}\ (\mathtt{one})\ (\mathtt{cns}\ (\mathtt{one})\ (\mathtt{zro}))) : \langle\mathtt{Map}\rangle}\;\mathtt{cns}$$

which can also be expressed as a tree:

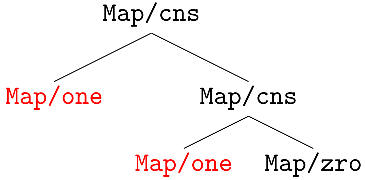


The class ⟨Map⟩ (cont.)

An example

$$\dfrac{\dfrac{}{(\texttt{one}) : \langle\texttt{Map}\rangle}\ \texttt{one} \qquad \dfrac{\dfrac{}{(\texttt{one}) : \langle\texttt{Map}\rangle}\ \texttt{one} \quad \dfrac{}{(\texttt{zro}) : \langle\texttt{Map}\rangle}\ \texttt{zro}}{(\texttt{cns (one) (zro)}) : \langle\texttt{Map}\rangle}\ \texttt{cns}}{(\texttt{cns (one) (cns (one) (zro))}) : \langle\texttt{Map}\rangle}\ \texttt{cns}$$

which can also be expressed as a tree:



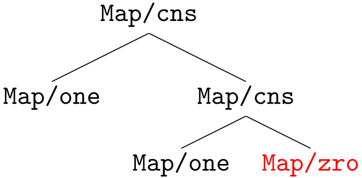
Remark 1 Map/one specifies occurrences of a *bound hole* in the scope of a λ-binder.

The class ⟨Map⟩ (cont.)

An example

$$\dfrac{\dfrac{}{(\mathtt{one}) : \langle\mathtt{Map}\rangle}\,\mathtt{one} \qquad \dfrac{\dfrac{}{(\mathtt{one}) : \langle\mathtt{Map}\rangle}\,\mathtt{one} \quad \dfrac{}{(\mathtt{zro}) : \langle\mathtt{Map}\rangle}\,\mathtt{zro}}{(\mathtt{cns}\ (\mathtt{one})\ (\mathtt{zro})) : \langle\mathtt{Map}\rangle}\,\mathtt{cns}}{(\mathtt{cns}\ (\mathtt{one})\ (\mathtt{cns}\ (\mathtt{one})\ (\mathtt{zro}))) : \langle\mathtt{Map}\rangle}\,\mathtt{cns}$$

which can also be expressed as a tree:



Remark 2 `Map/zro` specifies an occurrence of a *free hole* in the scope of a $\boldsymbol{\lambda}$-binder.

Submap relation on ⟨Map⟩

We define the *submap* relation on ⟨Map⟩ by the function Map/sub?

$$\texttt{Map/sub?} : \langle\texttt{Map}\rangle\ \langle\texttt{Map}\rangle \to \langle\texttt{bool}\rangle$$

```
(defun Map/sub? (m n)
  (case m
    ((zro) (case n ((zro) t) ((one) t)))
    ((one) (=? m n))
    ((cns m1 m2)
     (case n
       ((cns n1 n2)
        (and (Map/sub? m1 n1) (Map/sub? m2 n2)))))))
```

Minus operation on ⟨Map⟩

We define the *minus* operation on ⟨Map⟩ by the function `minus`.

$$\texttt{Map/mns} : \langle\texttt{Map}\rangle\ \langle\texttt{Map}\rangle \to \langle\texttt{Map}\rangle$$

```
(defun Map/mns (m n)
  (if (Map/sub? n m)
      (case m
        ((zro) (case n ((zro) m)))
        ((one) (case n ((zro) m) ((one) (Map/zro))))
        ((cns m1 m2)
         (case n
           ((cns n1 n2)
            (Map/cns (Map/mns m1 n1) (Map/mns m2 n2))))))
    (error "Cannot subtract")))
```

Closed map

We can define the function

$$\texttt{Map/closed?} : \langle \texttt{Map} \rangle \rightarrow \langle \texttt{bool} \rangle$$

```
(defun Map/closed? (m)
  "Check if <Map> m is closed."
  (case m
    ((zro) true)
    ((one) nil)
    ((cns m1 m2)
     (and (Map/closed? m1) (Map/closed? m2)))))
```

Schematic lambda expressions

The class $\langle \texttt{Sxp} \rangle$ of *schematic lambda expressions* is defined by the followng creation methods:

$$\frac{}{(\texttt{box}) : \langle \texttt{Sxp} \rangle}\ \texttt{box} \qquad \frac{x : \langle \texttt{Nat} \rangle}{(\texttt{var}\ x) : \langle \texttt{Sxp} \rangle}\ \texttt{var}$$

$$\frac{M : \langle \texttt{Sxp} \rangle \quad N : \langle \texttt{Sxp} \rangle}{(\texttt{app}\ M\ N) : \langle \texttt{Sxp} \rangle}\ \texttt{app} \qquad \frac{m : \langle \texttt{Map} \rangle \quad M : \langle \texttt{Sxp} \rangle}{(\texttt{lam}\ m\ M) : \langle \texttt{Sxp} \rangle}\ \texttt{lam}$$

Schematic lambda expressions

The class $\langle\texttt{Sxp}\rangle$ of *schematic lambda expressions* is defined by the followng creation methods:

$$\frac{}{(\texttt{box}) : \langle\texttt{Sxp}\rangle}\ \texttt{box} \qquad \frac{x : \langle\texttt{Nat}\rangle}{(\texttt{var}\ x) : \langle\texttt{Sxp}\rangle}\ \texttt{var}$$

$$\frac{M : \langle\texttt{Sxp}\rangle \quad N : \langle\texttt{Sxp}\rangle}{(\texttt{app}\ M\ N) : \langle\texttt{Sxp}\rangle}\ \texttt{app} \qquad \frac{m : \langle\texttt{Map}\rangle \quad M : \langle\texttt{Sxp}\rangle}{(\texttt{lam}\ m\ M) : \langle\texttt{Sxp}\rangle}\ \texttt{lam}$$

Remark 1 A (box) represents a hole which may be filled with an abstract later. Initially, a box is *free*, but it may become *bound* by the lam rule. We may think of a free box as representing the *schematic* (or, *meta*) variable.

Schematic lambda expressions

The class $\langle \texttt{Sxp} \rangle$ of *schematic lambda expressions* is defined by the followng creation methods:

$$\frac{}{(\texttt{box}) : \langle \texttt{Sxp} \rangle}\ \texttt{box} \qquad \frac{x : \langle \texttt{Nat} \rangle}{(\texttt{var}\ \ x) : \langle \texttt{Sxp} \rangle}\ \texttt{var}$$

$$\frac{M : \langle \texttt{Sxp} \rangle \quad N : \langle \texttt{Sxp} \rangle}{(\texttt{app}\ M\ \ N) : \langle \texttt{Sxp} \rangle}\ \texttt{app} \qquad \frac{m : \langle \texttt{Map} \rangle \quad M : \langle \texttt{Sxp} \rangle}{(\texttt{lam}\ m\ M) : \langle \texttt{Sxp} \rangle}\ \texttt{lam}$$

Remark 2 The rule `lam` may be applied only when m is a submap of $(\texttt{Sxp/2Map}\ \ M)$, where the function `Sxp/2Map` is defined in the next slide. We may think of m as representing Quine-Bourbaki *binding*.

Schematic lambda expressions (cont.)

We define:

$$\texttt{Sxp/2Map} : \langle \texttt{Sxp} \rangle \rightarrow \langle \texttt{Map} \rangle$$

```
(defun Sxp/2Map (M)
  (case M
    ((box) (Map/one))
    ((var x) (Map/zro))
    ((app M N) (Map/cns (Sxp/2Map M) (Sxp/2Map N)))
    ((lam m M) (Map/mns (Sxp/2Map M) m))))
```

Remark The Sxp/2Map function and the class $\langle \texttt{Sxp} \rangle$ are defined in a mutually inductive/recursive way.

Instantiation operation on $\langle \mathrm{Sxp} \rangle$

We define the *instantiation* function:

$$\texttt{Sxp/ist} : \langle \mathrm{Sxp} \rangle\ \langle \mathrm{Sxp} \rangle \to \langle \mathrm{Sxp} \rangle$$

```
(defun Sxp/ist (M N)
  "Instantiate <Sxp> M by <Sxp> N."
  (case M
    ((box) N)
    ((var x) M)
    ((app M1 M2) (Sxp/app (Sxp/ist M1 N) (Sxp/ist M2 N)))
    ((lam m M)
     (lam (Map/ist m (Sxp/2Map N)) (Sxp/ist M N)))))
```

We define `Map/ist` in the next slide.

Instantiation operation on $\langle \texttt{Map} \rangle$

We define the *instantiation* function:

$$\texttt{Map/ist} : \langle \texttt{Map} \rangle\ \langle \texttt{Map} \rangle \rightarrow \langle \texttt{Map} \rangle$$

```
(defun Map/ist (m n)
  "Instantiate <Map> m by <Map> n."
  (case m
    ((zro) m)
    ((one) n)
    ((var x) m)
    ((cns m1 m2)
     (Map/cns (Map/ist m1 n) (Map/ist m2 n)))))
```

Closed schematic expressions

We define the function

$$\texttt{Sxp/closed?} : \langle\texttt{Sxp}\rangle \to \langle\texttt{bool}\rangle$$

```
(defun Sxp/closed? (M)
  (Map/closed? (Sxp/2Map M)))
```

We define the class $\langle\texttt{Sxp0}\rangle$ as a subclass of $\langle\texttt{Sxp}\rangle$ consisting of *closed schematic expressions*.

Remark Compared to the definition of closedness of instances of $\langle\texttt{Dxp}\rangle$, the definition here is natural.

Lambda expressions

The class $\langle \texttt{Exp} \rangle$ of *lambda expressions* is defined by the followng creation methods:

$$\frac{x : \langle \texttt{Nat} \rangle}{(\texttt{var}\ \ x) : \langle \texttt{Exp} \rangle}\ \texttt{var}$$

$$\frac{M : \langle \texttt{Exp} \rangle \quad N : \langle \texttt{Exp} \rangle}{(\texttt{app}\ M\ \ N) : \langle \texttt{Exp} \rangle}\ \texttt{app} \qquad \frac{M : \langle \texttt{Sxp} \rangle}{(\texttt{lam}\ M) : \langle \texttt{Exp} \rangle}\ \texttt{lam}$$

Remark In the `lam` method, M must be an instance of $\langle \texttt{Sxp} \rangle$, but there are no extra side conditions on this rule.

Lambda expressions (cont.)

Since the creation method `lam` does not have extra side conditions, we can characterize $\langle\texttt{Exp}\rangle$ as the *free algebra* having the three operations below.

$$\begin{aligned} \texttt{Exp/var} &: \langle\texttt{Nat}\rangle \to \langle\texttt{Exp}\rangle \\ \texttt{Exp/app} &: \langle\texttt{Exp}\rangle\ \langle\texttt{Exp}\rangle \to \langle\texttt{Exp}\rangle \\ \texttt{Exp/lam} &: \langle\texttt{Sxp}\rangle \to \langle\texttt{Exp}\rangle \end{aligned}$$

Given an instance $\boldsymbol{M}$ of $\langle\texttt{Sxp}\rangle$, the `lam` method binds all the free boxes in $\boldsymbol{M}$.

For example, `(Exp/lam (Sxp/box))` represents the traditional lambda expression $\boldsymbol{\lambda x}[\boldsymbol{x}]$.

Comparison of the `lam` methods.

$$\frac{x : \langle \texttt{Nat} \rangle \quad M : \langle \texttt{Txp} \rangle}{(\texttt{lam}\ x\ M) : \langle \texttt{Txp} \rangle}\ \texttt{lam}$$

$$\frac{i : \langle \texttt{Nat} \rangle}{(\texttt{idx}\ \ i) : \langle \texttt{Dxp} \rangle}\ \texttt{idx} \qquad \frac{M : \langle \texttt{Dxp} \rangle}{(\texttt{lam}\ M) : \langle \texttt{Dxp} \rangle}\ \texttt{lam}$$

$$\frac{m : \langle \texttt{Map} \rangle \quad M : \langle \texttt{Sxp} \rangle}{(\texttt{lam}\ m\ M) : \langle \texttt{Sxp} \rangle}\ \texttt{lam} \qquad \frac{M : \langle \texttt{Sxp} \rangle}{(\texttt{lam}\ M) : \langle \texttt{Exp} \rangle}\ \texttt{lam}$$

Remark 1. In $\langle \texttt{Txp} \rangle$, `lam` is not injective.

Comparison of the `lam` methods.

$$\frac{x : \langle \texttt{Nat} \rangle \quad M : \langle \texttt{Txp} \rangle}{(\texttt{lam}\ x\ M) : \langle \texttt{Txp} \rangle}\ \texttt{lam}$$

$$\frac{i : \langle \texttt{Nat} \rangle}{(\texttt{idx}\ \ i) : \langle \texttt{Dxp} \rangle}\ \texttt{idx} \qquad \frac{M : \langle \texttt{Dxp} \rangle}{(\texttt{lam}\ M) : \langle \texttt{Dxp} \rangle}\ \texttt{lam}$$

$$\frac{m : \langle \texttt{Map} \rangle \quad M : \langle \texttt{Sxp} \rangle}{(\texttt{lam}\ m\ M) : \langle \texttt{Sxp} \rangle}\ \texttt{lam} \qquad \frac{M : \langle \texttt{Sxp} \rangle}{(\texttt{lam}\ M) : \langle \texttt{Exp} \rangle}\ \texttt{lam}$$

Remark 2. In $\langle \texttt{Dxp} \rangle$, `lam` binds indices determined by M.

Comparison of the `lam` methods.

$$\frac{x : \langle\texttt{Nat}\rangle \quad M : \langle\texttt{Txp}\rangle}{(\texttt{lam}\ x\ M) : \langle\texttt{Txp}\rangle}\ \texttt{lam}$$

$$\frac{i : \langle\texttt{Nat}\rangle}{(\texttt{idx}\ \ i) : \langle\texttt{Dxp}\rangle}\ \texttt{idx} \qquad \frac{M : \langle\texttt{Dxp}\rangle}{(\texttt{lam}\ M) : \langle\texttt{Dxp}\rangle}\ \texttt{lam}$$

$$\frac{m : \langle\texttt{Map}\rangle \quad M : \langle\texttt{Sxp}\rangle}{(\texttt{lam}\ m\ M) : \langle\texttt{Sxp}\rangle}\ \texttt{lam} \qquad \frac{M : \langle\texttt{Sxp}\rangle}{(\texttt{lam}\ M) : \langle\texttt{Exp}\rangle}\ \texttt{lam}$$

Remark 3. In $\langle\texttt{Sxp}\rangle$, `lam` binds only boxes specified by m from free boxes in M. `lam` is *injective* since the method may be applied only when m is a submap of $(\texttt{Sxp/2Map}\ M)$.

Comparison of the `lam` methods.

$$\frac{x : \langle \mathtt{Nat} \rangle \quad M : \langle \mathtt{Txp} \rangle}{(\mathtt{lam}\ x\ M) : \langle \mathtt{Txp} \rangle}\ \mathtt{lam}$$

$$\frac{i : \langle \mathtt{Nat} \rangle}{(\mathtt{idx}\ \ i) : \langle \mathtt{Dxp} \rangle}\ \mathtt{idx} \qquad \frac{M : \langle \mathtt{Dxp} \rangle}{(\mathtt{lam}\ M) : \langle \mathtt{Dxp} \rangle}\ \mathtt{lam}$$

$$\frac{m : \langle \mathtt{Map} \rangle \quad M : \langle \mathtt{Sxp} \rangle}{(\mathtt{lam}\ m\ M) : \langle \mathtt{Sxp} \rangle}\ \mathtt{lam} \qquad \frac{M : \langle \mathtt{Sxp} \rangle}{(\mathtt{lam}\ M) : \langle \mathtt{Exp} \rangle}\ \mathtt{lam}$$

Remark 4. In $\langle \mathtt{Exp} \rangle$, `lam` does not have any extra side condition.

Isomorphism between $\langle\texttt{Exp}\rangle$ and $\langle\texttt{Sxp0}\rangle$

We define a function

$$\texttt{Exp/2Sxp} : \langle\texttt{Exp}\rangle \to \langle\texttt{Sxp}\rangle$$

```
(defun Exp/2Sxp (M)
  "Convert <Exp> M to an <Sxp>."
  (case M
    ((var x) (Sxp/var x))
    ((app M N) (Sxp/app (Exp/2Sxp M) (Exp/2Sxp N)))
    ((lam M) (Sxp/lam (Sxp/2Map M) M))))
```

We define Sxp/2Exp as the inverse of Exp/2Sxp.

β-conversion

We can define the function

$$\texttt{Exp/beta} : \langle\texttt{Exp}\rangle\ \langle\texttt{Exp}\rangle \rightarrow \langle\texttt{Exp}\rangle$$

```
(defun Exp/beta (M N)
  (case M
    ((lam M) (Sxp/2Exp (Sxp/ist M (Exp/2Sxp N))))
    (_ (error "Cannot convert"))))
```

Translation from ⟨Txp⟩

[demo]