Symbolic Expressions and Variable Binding
Lecture 5

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 Natural Deduction system
- $\langle$`Map`$\rangle$: Map
- $\langle$`Fun`$\rangle$: Functions
- $\langle$`Obj`$\rangle$: Objects
- $\langle$`ObL`$\rangle$: List of objects
- $\langle$`obj`$\rangle$: Schematic Objects
- $\langle$`obL`$\rangle$: Schematic List of objects
- $\langle$`Pfn`$\rangle$: Propositional Functions
- $\langle$`prp`$\rangle$: Schematic Propositions
- $\langle$`Prp`$\rangle$: Propositions
- $\langle$`Der`$\rangle$: Derivations

Map based Natural Deduction system

We define a Natural Deduction system for a first-order predicate logic as an *abstract data type*.

This is done by creating the mother class $\langle \texttt{Der} \rangle$ of derivations.

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

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

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

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

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

We define the class of *functions* $\langle \texttt{Fun} \rangle$. Each function has a fixed arity, and for each arity there are countably many functions having the arity.

$$\texttt{Fun/fun} : \langle \texttt{Nat} \rangle\ \langle \texttt{Nat} \rangle \to \langle \texttt{Fun} \rangle$$

The first argument of `fun` is the name of the created function and the second its arity.
The creation method has no extra side-condition. So, we could *specify* it as above. The method may be specified in the usual form below.

$$\frac{\boldsymbol{f} : \langle \texttt{Nat} \rangle \quad \boldsymbol{n} : \langle \texttt{Nat} \rangle}{(\texttt{fun}\ \boldsymbol{f}\ \boldsymbol{n}) : \langle \texttt{Fun} \rangle}\ \texttt{fun}$$

The class ⟨Obj⟩

The class ⟨Obj⟩ of *objects* is defined by the following specification. This class is defined simultaneously with the class ⟨ObL⟩ of *list of objects*.

$$\mathtt{var} : \langle \mathtt{Nat} \rangle \to \langle \mathtt{Obj} \rangle$$
$$\mathtt{app} : \langle \mathtt{Fun} \rangle\ \langle \mathtt{ObL} \rangle \to \langle \mathtt{Obj} \rangle$$

The *application* method (app) may be applied only when the following side-condition is satisfied:

```
(defun Obj/app-ok? (F L)
  "Check if <Fun> F is applicable to <ObL> L."
  (case F
    ((fun f n) (=? n (ObL/length L)))))
```

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

The class $\langle \texttt{ObL} \rangle$ of *list of objects* is defined by the following specification.

$$\texttt{nil} : \; \to \langle \texttt{ObL} \rangle$$
$$\texttt{cns} : \langle \texttt{Obj} \rangle \; \langle \texttt{ObL} \rangle \to \langle \texttt{ObL} \rangle$$

The class ⟨obj⟩

The class ⟨obj⟩ of *schematic objects* is defined by the following specification. This class is defined simultaneously with the class ⟨obL⟩ of *schematic list of objects*.

$$\begin{aligned} \texttt{box} &: \;\to \langle\texttt{obj}\rangle \\ \texttt{var} &: \langle\texttt{Nat}\rangle \to \langle\texttt{obj}\rangle \\ \texttt{app} &: \langle\texttt{Fun}\rangle\ \langle\texttt{obL}\rangle \to \langle\texttt{obj}\rangle \end{aligned}$$

The *application* method (app) may be applied only when the following side-condition is satisfied:

```
(defun obj/app-ok? (F L)
  "Check if <Fun> F is applicable to <obL> L."
  (case F
    ((fun f n) (=? n (obL/length L)))))
```

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

The class $\langle \texttt{obL} \rangle$ of *list of schematic objects* is defined by the following specification.

$$\begin{aligned} \texttt{nil} &: \;\to \langle \texttt{obL} \rangle \\ \texttt{cns} &: \langle \texttt{obj} \rangle \; \langle \texttt{obL} \rangle \to \langle \texttt{obL} \rangle \end{aligned}$$

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

We define the class $\langle \texttt{Pfn} \rangle$ of *propositional functions* Each function has a fixed arity, and for each arity there are countably many functions having the arity.

$$\texttt{Pfn/fun} : \langle \texttt{Nat} \rangle \ \langle \texttt{Nat} \rangle \to \langle \texttt{Fun} \rangle$$

The first argument of `fun` is the name of the created function and the second its arity.

The class ⟨prp⟩

The class ⟨prp⟩ of *schematic propositions* is defined by the following specification. This class is defined simultaneously with the class ⟨prp⟩ of *schematic propositions*.

$$
\begin{aligned}
\texttt{var} &: \langle\texttt{Nat}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{app} &: \langle\texttt{Pfn}\rangle\ \langle\texttt{obL}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{imp} &: \langle\texttt{prp}\rangle\ \langle\texttt{prp}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{all} &: \langle\texttt{Map}\rangle\ \langle\texttt{prp}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{som} &: \langle\texttt{Map}\rangle\ \langle\texttt{prp}\rangle \to \langle\texttt{prp}\rangle
\end{aligned}
$$

The *application* method (app) may be applied only when the following side-condition is satisfied:

```
(defun Prp/app-ok? (P L)
  "Check if <Pfn> P is applicable to <obL> L."
  (case P
    ((fun p n) (=? n (ObL/length L)))))
```

The class ⟨prp⟩

The class ⟨prp⟩ of *schematic propositions* is defined by the following specification. This class is defined simultaneously with the class ⟨prp⟩ of *schematic propositions*.

$$\begin{aligned}
\texttt{var} &: \langle\texttt{Nat}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{app} &: \langle\texttt{Pfn}\rangle\ \langle\texttt{obL}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{imp} &: \langle\texttt{prp}\rangle\ \langle\texttt{prp}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{all} &: \langle\texttt{Map}\rangle\ \langle\texttt{prp}\rangle \to \langle\texttt{prp}\rangle \\
\texttt{som} &: \langle\texttt{Map}\rangle\ \langle\texttt{prp}\rangle \to \langle\texttt{prp}\rangle
\end{aligned}$$

The methods `all` and `som` are applicable to $\boldsymbol{m} : \langle\texttt{Map}\rangle$ and $\boldsymbol{A} : \langle\texttt{prp}\rangle$ only when $\boldsymbol{m}$ is a submap of $(\texttt{prp/2Map}\ \ \boldsymbol{A})$.

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

The class $\langle \texttt{Prp} \rangle$ of *propositions* is defined by the following specification. This class is defined using the class $\langle \texttt{prp} \rangle$.

$$
\begin{aligned}
\texttt{var} &: \langle \texttt{Nat} \rangle \to \langle \texttt{Prp} \rangle \\
\texttt{app} &: \langle \texttt{Pfn} \rangle\ \langle \texttt{ObL} \rangle \to \langle \texttt{Prp} \rangle \\
\texttt{imp} &: \langle \texttt{Prp} \rangle\ \langle \texttt{Prp} \rangle \to \langle \texttt{Prp} \rangle \\
\texttt{all} &: \langle \texttt{prp} \rangle \to \langle \texttt{Prp} \rangle \\
\texttt{som} &: \langle \texttt{prp} \rangle \to \langle \texttt{Prp} \rangle
\end{aligned}
$$

The *application* method (app) may be applied only when the following side-condition is satisfied:

```
(defun Prp/app-ok? (P L)
  "Check if <Pfn> P is applicable to <ObL> L."
  (case P
    ((fun p n) (=? n (ObL/length L)))))
```

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

We are now ready to define the mother class $\langle \texttt{Der} \rangle$ of *derivations*. Simultaneously with the definition of the class, we define the functions:

$$\texttt{Der/FA} : \langle \texttt{Der} \rangle \to \langle \texttt{Map} \rangle$$
$$\texttt{Der/ccl} : \langle \texttt{Der} \rangle \to \langle \texttt{Prp} \rangle$$

The function `Der/FA` computes the positions of *free assumptions* as a map.

The function `Der/ccl` computes the *conclusion* of a given derivation.

In this way, we can *extract* both the free assumptions and the conclusion of any derivation by computation.

The class $\langle \texttt{Der} \rangle$ (cont.)

The class $\langle \texttt{Der} \rangle$ of *derivations* is defined by the following specification.

$$\begin{aligned}
\texttt{asm} &: \langle \texttt{Nat} \rangle\ \langle \texttt{Prp} \rangle \to \langle \texttt{Der} \rangle \\
\texttt{imI} &: \langle \texttt{Nat} \rangle\ \langle \texttt{Prp} \rangle\ \langle \texttt{Der} \rangle \to \langle \texttt{Der} \rangle \\
\texttt{imE} &: \langle \texttt{Der} \rangle\ \langle \texttt{Der} \rangle \to \langle \texttt{Der} \rangle \\
\texttt{alI} &: \langle \texttt{Nat} \rangle\ \langle \texttt{Der} \rangle \to \langle \texttt{Der} \rangle \\
\texttt{alE} &: \langle \texttt{Der} \rangle\ \langle \texttt{Obj} \rangle \to \langle \texttt{Der} \rangle \\
\texttt{smI} &: \langle \texttt{Der} \rangle\ \langle \texttt{prp} \rangle\ \langle \texttt{Obj} \rangle \to \langle \texttt{Der} \rangle \\
\texttt{smE} &: \langle \texttt{Der} \rangle\ \langle \texttt{Nat} \rangle\ \langle \texttt{Der} \rangle \to \langle \texttt{Der} \rangle
\end{aligned}$$

Of these, only the first two methods do not have extra side-conditions.

Assumption

The method asm enables us to *assume* that a proposition holds.

$$\frac{i : \langle \texttt{Nat} \rangle \quad A : \langle \texttt{Prp} \rangle}{(\texttt{asm}\ \ i\ A) : \langle \texttt{Der} \rangle}\ \texttt{asm}$$

The number i is used by the imI method to refer to the assumption.

Note that $A : \langle \texttt{Prp} \rangle$ means that the second argument of the method asm must be a proposition (that is, $\langle \texttt{Prp} \rangle$).

The same holds for the first argument as well.

Implication

The two methods below *introduce* and *eliminate* an implication proposition from its major argument.

$$\frac{\boldsymbol{n} : \langle \texttt{Nat} \rangle \quad \boldsymbol{A} : \langle \texttt{Prp} \rangle \quad \boldsymbol{d} : \langle \texttt{Der} \rangle}{(\texttt{imI}\ \boldsymbol{n}\ \boldsymbol{A}\ \boldsymbol{d}) : \langle \texttt{Der} \rangle}\ \texttt{imI} \qquad \frac{\boldsymbol{d} : \langle \texttt{Der} \rangle \quad \boldsymbol{e} : \langle \texttt{Der} \rangle}{(\texttt{imE}\ \boldsymbol{d}\ \boldsymbol{e}) : \langle \texttt{Der} \rangle}\ \texttt{imE}$$

The method `imE` has the following side-condition.

```
(defun Der/imE-ok? (d e)
  (case (Der/ccl d)
    ((imp A B) (=? A (Der/ccl e)))
    (_ nil)))
```

Universal proposition

The two methods below *introduce* and *eliminate* an implication proposition from its major argument.

$$\frac{x : \langle\texttt{Nat}\rangle \quad d : \langle\texttt{Der}\rangle}{(\texttt{alI}\ x\ d) : \langle\texttt{Der}\rangle}\ \texttt{alI} \qquad \frac{d : \langle\texttt{Der}\rangle \quad a : \langle\texttt{Obj}\rangle}{(\texttt{alE}\ d\ a) : \langle\texttt{Der}\rangle}\ \texttt{alE}$$

Remark 1 The method $\langle\texttt{alI}\rangle$ has the following side-condition.

```
(defun Der/alI-ok? (x d)
  (Der/eigen? x (Der/FA d) d))
```

We define `Der/eigen?` later.

Universal proposition

The two methods below *introduce* and *eliminate* an implication proposition from its major argument.

$$\frac{x : \langle \texttt{Nat} \rangle \quad d : \langle \texttt{Der} \rangle}{(\texttt{alI}\ \ x\ d) : \langle \texttt{Der} \rangle}\ \texttt{alI} \qquad \frac{d : \langle \texttt{Der} \rangle \quad a : \langle \texttt{Obj} \rangle}{(\texttt{alE}\ \ d\ a) : \langle \texttt{Der} \rangle}\ \texttt{alE}$$

Remark 2 The method $\langle \texttt{alE} \rangle$ has the following side-condition.

```
(defun Der/alE-ok? (d a)
  (case (Der/ccl d)
    ((all A) true)
    (_ nil)))
```

Existential proposition

The two methods below *introduce* and *eliminate* an implication proposition from its major argument.

$$\frac{\boldsymbol{d} : \langle \texttt{Der} \rangle \quad \boldsymbol{A} : \langle \texttt{prp} \rangle \quad \boldsymbol{a} : \langle \texttt{Obj} \rangle}{(\texttt{smI}\ \boldsymbol{d}\ \boldsymbol{A}\ \boldsymbol{a}) : \langle \texttt{Der} \rangle}\ \texttt{smI}$$

The method $\langle \texttt{smI} \rangle$ has the following side-condition.

```
(defun Der/smI-ok? (d A a)
  (=? (Der/ccl d)
      (prp/2Prp (prp/ist A (Obj/2obj a)))))
```

Existential proposition (cont.)

$$\frac{\boldsymbol{d} : \langle \texttt{Der} \rangle \quad \boldsymbol{x} : \langle \texttt{Nat} \rangle \quad \boldsymbol{e} : \langle \texttt{Der} \rangle}{(\texttt{smE}\ \boldsymbol{d}\ \boldsymbol{x}\ \boldsymbol{e}) : \langle \texttt{Der} \rangle}\ \texttt{smE}$$

The method $\langle \texttt{smE} \rangle$ has the following side-condition.

```
(defun Der/smE-ok? (d x e)
  (case (Der/ccl d)
    ((som A)
     (case (Der/ccl e)
       ((imp A1 C)
        ;; A1 = A(x)
        (and (=? A1 (prp/2Prp (prp/ist A (prp/var x))))
             (Der/eigen? x (Der/FA e) e)
             (not (Prp/occ x C)))))
       (_ nil))
     (_ nil))))
```

The conclusion of a derivation

We define `Der/ccl` : ⟨`Der`⟩ → ⟨`Prp`⟩ as follows.

```
(defun Der/ccl (d)
  (case d
    ((asm n A) A)
    ((imI n A d) (Prp/imp A (Der/ccl d)))
    ((imE d e)
     (case (Der/ccl d) ((imp A B) B)))
    ((alI x d) (Prp/All x (Der/ccl d)))
    ((alE d a)
     (case (Der/ccl d)
       ((all A) (prp/2Prp (prp/ist A (Obj/2obj a))))))
    ((smI d A a) (Prp/som A))
    ((smE d x e)
     (case (Der/ccl e) ((imp A C) C)))))
```

Free assumptions of a derivation

We define Der/FA : ⟨Der⟩ → ⟨Map⟩ as follows.

```
(defun Der/FA (d)
  ((asm n A) (Map/one))
  ((imI n A d) (Map/mns (Der/FA d) (Der/occ n A d)))
  ((imE d e) (Map/cns (Der/FA d) (Der/FA e)))
  ((alI x d) (Der/FA d))
  ((alE d a) (Der/FA d))
  ((smI d A a) (Der/FA d))
  ((smE d x e) (Map/cns (Der/FA d) (Der/FA e))))
```

Free occurrences of an assumption

We define Der/occ : ⟨Nat⟩ ⟨Prp⟩ ⟨Der⟩ → ⟨Map⟩ as follows.

```
(defun Der/occ (i A d)
  (case d
    ((asm j B)
     (if (and (=? A B) (=? i j)) (Map/one) (Map/zro)))
    ((imI j B d)
     (if (and (=? A B) (=? i j)) (Der/shp d) (Der/occ i A d)))
    ((imE d e) (Map/cns (Der/occ i A d) (Der/occ i A e)))
    ((alI x d) (Der/occ i A d))
    ((alE d a) (Der/occ i A d))
    ((smI d A a) (Der/occ i A d))
    ((smE d x e) (Map/cns (Der/occ i A d) (Der/occ i A e)))))
```

The shape of a derivation

We define Der/shp : ⟨Der⟩ → ⟨Map⟩ as follows.

```
(defun Der/shp (d)
  "Compute the shape of <Der> d as a map."
  (case d
    ((asm j B) (Map/zro))
    ((imI j B d) (Der/shp d))
    ((imE d e) (Map/cns (Der/shp d) (Der/shp e)))
    ((alI x d) (Der/shp d))
    ((alE d a) (Der/shp d))
    ((smI d A a) (Der/shp d))
    ((smE d x e) (Map/cns (Der/shp d) (Der/shp e)))))
```

Eigenvariable condition

```
(defun Der/eigen? (x m d)
  (case d
    ((asm i A) (if (=? m (Map/zro)) true (not (Prp/occ? x A))))
    ((imI i A d) (Der/eigen? x m d))
    ((imE d e)
     (case m
       ((cns m1 m2)
        (and (Der/eigen? x m1 d) (Der/eigen? x m2 e)))))
    ((alI y d)
     (if (=? x y) true (Der/eigen? x m d)))
    ((alE d a) (Der/eigen? x m d))
    ((smI d A a) (Der/eigen? x m d))
    ((smE d y e)
     (if (=? x y) true
       (case m
         ((cns m1 m2)
          (and (Der/eigen? x m1 d) (Der/eigen? x m2 e)))))))))
```