ソフトウェア構成特論 第11回

大学院理工学研究科 電気電子情報工学専攻　篠埜 功

2014年6月26日

1　はじめに

今回から、単純型付きラムダ計算にいくつかの構文を追加していき、実際の言語に近づけていく。

2　新たな構文を追加する前の計算体系

新たな構文を追加する前の計算体系として、単純型付きラムダ計算にブール式と数式が入っている、以下の計算体系を用いる。今後、以下の計算体系にさまざまな構文を追加していく。

2.1　式、値、型

式を

$$\begin{aligned} \mathtt{t} \;::=\; & \mathtt{x} \mid \lambda \mathtt{x} : \mathtt{T}.\mathtt{t} \mid \mathtt{t}\ \mathtt{t} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{if\ t\ then\ t\ else\ t} \mid \\ & \mathtt{0} \mid \mathtt{succ\ t} \mid \mathtt{pred\ t} \mid \mathtt{iszero\ t} \end{aligned}$$

と定義し、値を

$$\begin{aligned} \mathtt{v} \;::=\; & \lambda \mathtt{x} : \mathtt{T}.\mathtt{t} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{nv} \\ \mathtt{nv} \;::=\; & \mathtt{0} \mid \mathtt{succ\ nv} \end{aligned}$$

と定義し、型を

$$\mathtt{T} ::= \mathtt{T} \to \mathtt{T} \mid \mathtt{Bool} \mid \mathtt{Nat}$$

と定義し、型環境を

$$\Gamma ::= \emptyset \mid \Gamma, \mathtt{x} : \mathtt{T}$$

と定義する。

また、上記の式について、置換は以下のように定義される。

$$
\begin{array}{ll}
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{x} = \mathtt{s} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{y} = \mathtt{y} & \text{if } \mathtt{y} \neq \mathtt{x} \\
[\mathtt{x} \mapsto \mathtt{s}](\lambda \mathtt{y} : \mathtt{T}.\ \mathtt{t}_1) = \lambda \mathtt{y} : \mathtt{T}.\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1 & \text{if } \mathtt{y} \neq \mathtt{x} \text{ and } \mathtt{y} \notin FV(\mathtt{s}) \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1)([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2) & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{true} = \mathtt{true} & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{false} = \mathtt{true} & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3) = & \\
\quad \mathtt{if}\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1\ \mathtt{then}\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2\ \mathtt{else}\ [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_3 & \\
[\mathtt{x} \mapsto \mathtt{s}]\mathtt{0} = \mathtt{0} & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{succ}\ \mathtt{t}) = \mathtt{succ}\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}) & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{pred}\ \mathtt{t}) = \mathtt{pred}\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}) & \\
[\mathtt{x} \mapsto \mathtt{s}](\mathtt{iszero}\ \mathtt{t}) = \mathtt{iszero}\ ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}) &
\end{array}
$$

ただし、第 6 回の講義資料で説明した通り、ラムダ抽象に対する置換適用時には、必要に応じて置換対象のラムダ抽象式の束縛変数の付け替えを行う。

2.2 評価規則

評価規則は、以下に示す 13 個の評価規則とする。これは、第 5 回の資料で提示した算術式に関する 10 個の評価規則に、第 9 回の資料で提示したラムダ式に関する評価規則 (E-APP1), (E-APP2), (E-APPABS) を追加したものである。

$$\frac{}{\mathtt{if\ true\ then\ t}_2\ \mathtt{else\ t}_3 \to \mathtt{t}_2}\ \text{(E-IFTRUE)}$$

$$\frac{}{\mathtt{if\ false\ then\ t}_2\ \mathtt{else\ t}_3 \to \mathtt{t}_3}\ \text{(E-IFFALSE)}$$

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{if\ t}_1\ \mathtt{then\ t}_2\ \mathtt{else\ t}_3 \to \mathtt{if\ t}_1'\ \mathtt{then\ t}_2\ \mathtt{else\ t}_3}\ \text{(E-IF)}$$

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{succ\ t}_1 \to \mathtt{succ\ t}_1'}\ \text{(E-SUCC)} \qquad \frac{}{\mathtt{pred\ 0} \to \mathtt{0}}\ \text{(E-PREDZERO)}$$

$$\frac{}{\mathtt{pred\ (succ\ nv}_1) \to \mathtt{nv}_1}\ \text{(E-PREDSUCC)} \qquad \frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{pred\ t}_1 \to \mathtt{pred\ t}_1'}\ \text{(E-PRED)}$$

$$\frac{}{\mathtt{iszero\ 0} \to \mathtt{true}}\ \text{(E-ISZEROZERO)}$$

$$\frac{}{\mathtt{iszero\ (succ\ nv}_1) \to \mathtt{false}}\ \text{(E-ISZEROSUCC)} \qquad \frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{iszero\ t}_1 \to \mathtt{iszero\ t}_1'}\ \text{(E-ISZERO)}$$

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \to \mathtt{t}_1'\ \mathtt{t}_2}\ \text{(E-APP1)} \quad \frac{\mathtt{t}_2 \to \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \to \mathtt{v}_1\ \mathtt{t}_2'}\ \text{(E-APP2)}$$

$$\frac{}{(\lambda \mathtt{x} : \mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \to [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}}\ \text{(E-APPABS)}$$

2.3 型システム

型付け規則は

$$\frac{\mathtt{x} : \mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}}\ \text{(T-VAR)} \qquad \frac{\Gamma, \mathtt{x} : \mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x} : \mathtt{T}_1.\ \mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2}\ \text{(T-ABS)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}}\ \text{(T-APP)}$$

$$\frac{}{\Gamma \vdash \mathtt{true} : \mathtt{Bool}}\ \text{(T-TRUE)} \qquad \frac{}{\Gamma \vdash \mathtt{false} : \mathtt{Bool}}\ \text{(T-FALSE)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}}\ \text{(T-IF)}$$

$$\frac{}{\Gamma \vdash \mathtt{0} : \mathtt{Nat}}\ \text{(T-ZERO)} \qquad \frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{succ}\ \mathtt{t}_1 : \mathtt{Nat}}\ \text{(T-SUCC)} \qquad \frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{pred}\ \mathtt{t}_1 : \mathtt{Nat}}\ \text{(T-PRED)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{iszero}\ \mathtt{t}_1 : \mathtt{Bool}}\ \text{(T-ISZERO)}$$

と定義する。

3 基本型

プログラミング言語は通常数、真偽値、文字など、構造を持たない値の集合を基本型として提供する。また、基本型の値を操作する演算も提供する。これまでに自然数の型 `Nat` と真偽値の型 `Bool` を用いてきた。その他の型としては、`"hello"`のような文字列の型 `String` や `3.14159` のような浮動小数点数の型 `Float` などを今後用いる。

基本型を表すメタ変数として `A` を用いる。また、基本型の名前として `A`, `B`, `C` などを用いる。例えば、term `λx:A.x` は空の型環境で `A→A` 型を持ち、term `λx:B.x` は空の型環境で `B→B` 型を持ち、term `λf:A→A. λx:A. f(f(x))` は空の型環境で `(A→A)→(A→A)` 型を持つ。

4 Unit 型

`unit` という、`Unit` 型の値を導入する。つまり、term 集合に `unit` を追加し、値に `unit` を追加し、型に `Unit` を追加する。また、`Unit` 型の term の評価結果は必ず `unit` になる。また、`unit` に関する型付け規則を以下のように定義する。

$$\frac{}{\Gamma \vdash \mathtt{unit} : \mathtt{Unit}}\ \text{(T-UNIT)}$$

`unit` は、（通常）副作用のあるプログラムで用いる。前回までに提示した計算体系では副作用のある term はないが、reference cell への代入や画面への出力などが副作用である。副作用が目的で、値が不要な場合でも何らかの値が必要であり、そのような場合のために `unit` が用いられる。C 言語や Java 言語でいえば `void` 型を使う場合がこれに相当する。

置換の定義に `unit` の場合を追加する。

$$[\mathtt{x} \mapsto \mathtt{s}]\mathtt{unit} = \mathtt{unit}$$

5 逐次式

副作用のある言語では、2つ以上の式を順番に評価するということがよく行われる。そのために、逐次式 $\mathtt{t}_1;\mathtt{t}_2$ を用いる。この式を評価するときは、まず $\mathtt{t}_1$ を評価し、その評価結果を捨ててから $\mathtt{t}_2$ の評価をし、$\mathtt{t}_2$ の評価結果を $\mathtt{t}_1;\mathtt{t}_2$ の評価結果とする。これを規則で書くと以下のようになる。

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{t}_1;\mathtt{t}_2 \to \mathtt{t}_1';\mathtt{t}_2}\ \text{(E-SEQ)} \qquad \frac{}{\mathtt{unit};\mathtt{t}_2 \to \mathtt{t}_2}\ \text{(E-SEQNEXT)}$$

また以下の型付け規則を追加する。

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Unit} \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{t}_1;\mathtt{t}_2 : \mathtt{T}_2}\ \text{(T-SEQ)}$$

また置換の定義に逐次式の場合を追加する。

$$[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t}_1;\mathtt{t}_2) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1);([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2)$$

$\mathtt{t}_1;\mathtt{t}_2$ と同等のことを、$(\lambda \mathtt{x} : \mathtt{Unit}.\mathtt{t}_2)\mathtt{t}_1$（ただし $\mathtt{x}$ は $\mathtt{t}_2$ の自由変数ではない変数）でできるので、$\mathtt{t}_1;\mathtt{t}_2$ を $(\lambda \mathtt{x} : \mathtt{Unit}.\mathtt{t}_2)\mathtt{t}_1$ の略記として定義してもよい。

定理 1. 単純型付きラムダ計算を Unit 型、逐次式、E-SEQ、E-SEQNEXT、T-SEQ 規則で拡張した体系を λ^E と書く。また、単純型付きラムダ計算を Unit 型で拡張しただけの体系を λ^I と書く。

λ^E の term が与えられたとき、その中の $\mathtt{t}_1;\mathtt{t}_2$ の形の term を全部 $(\lambda \mathtt{x} : \mathtt{Unit}.\mathtt{t}_2)\mathtt{t}_1$（ただし $\mathtt{x}$ は $\mathtt{t}_2$ の自由変数ではない変数）で置き換えて得られる λ^I の term を返す関数 $e \in \lambda^E \to \lambda^I$ を *elaboration function* と呼ぶ。このとき、λ^E の各 term $\mathtt{t}$ に対し、

- $\mathtt{t} \to_E \mathtt{t'}$ iff $e(\mathtt{t}) \to_I e(\mathtt{t}')$
- $\Gamma \vdash^E \mathtt{t} : \mathtt{T}$ iff $\Gamma \vdash^I e(\mathtt{t}) : \mathtt{T}$

が成り立つ。ここで、$\to$、$\vdash$ の添字により、どちらの計算体系の評価関係あるいは型付け関係であるかを示している。

証明. 2つの命題のそれぞれについて、両方向とも $\mathtt{t}$ の構造帰納法で示すことができる。 □

この定理により、derived form を使っていいということが分かる。derived form を使う利点は、内部の言語を変えずに表面の構文だけ追加することができるというところにあり、これにより、内部言語で成り立つ性質（Progress や Preservation のような性質）がそのまま成り立つことになる。

Derived form は syntactic sugar（構文糖衣）あるいは syntax sugar とも呼ばれる。Derived form をその定義で展開することを desugaring あるいは elaboration という。

練習問題 1. 以下の式は空の型環境で型を持つが、その型判定およびその導出木を書け。

$$(\mathtt{unit};\mathtt{unit});\mathtt{unit}$$

練習問題 2. 以下の式を正規形になるまで評価せよ。その際、各1ステップ評価の導出木も書け。

$$(\mathtt{unit};\mathtt{unit});\mathtt{unit}$$

6 Wildcard

もう一つ有用な derived form として、ラムダ抽象の束縛変数部分で wildcard を使うというものがある。ラムダ抽象の束縛変数が本体内で使われない場合に wildcard を使えば、束縛変数が使われないということが一目で分かるようになり、またプログラムを書く際に変数名を考える手間が省ける。束縛変数部分に wildcard を使った式 $\lambda\_ : \mathtt{S}.\mathtt{t}$ は、$\mathtt{t}$ の自由変数ではない任意の変数を $\mathtt{x}$ としたとき、$\lambda\mathtt{x} : \mathtt{S}.\mathtt{t}$ の derived form として定義すればよい。以下では、derived form ではなく、核言語に wildcard を追加する。

まず、式および値に wildcard を追加する。

$$\begin{aligned} \mathtt{t} &::= \ldots \mid \lambda\_ : \mathtt{T}.\mathtt{t} \\ \mathtt{v} &::= \ldots \mid \lambda\_ : \mathtt{T}.\mathtt{t} \end{aligned}$$

ワイルドカードに関する型付け規則は

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{T}}{\Gamma \vdash \lambda\_ : \mathtt{S}.\mathtt{t} : \mathtt{S} \to \mathtt{T}} \text{ (T-WILD)}$$

とすればよい。また評価規則は

$$\frac{}{(\lambda\_ : \mathtt{S}.\mathtt{t})\ \mathtt{v} \to \mathtt{t}} \text{ (E-WILD)}$$

とすればよい。また置換の定義にワイルドカードの場合を追加する。

$$[\mathtt{x} \mapsto \mathtt{s}](\lambda\_ : \mathtt{S}.\mathtt{t}) = \lambda\_ : \mathtt{S}.[\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}$$

練習問題 3. term

$$(\lambda\_ : \mathtt{Nat}.\mathtt{true})\ ((\lambda\mathtt{x} : \mathtt{Nat}.\mathtt{x})\ 0)$$

は空の型環境で型を持つが、その型判定の導出木を書け。

練習問題 4. term

$$(\lambda\_ : \mathtt{Nat}.\mathtt{true})\ ((\lambda\mathtt{x} : \mathtt{Nat}.\mathtt{x})\ 0)$$

を評価せよ。各 1 ステップ評価の導出木も書け。

7 Ascription

term に型を書く (ascribe) ことができると便利がよい場合がある。ある term がある型を持つというつもりでプログラムを書いているとき、その型をそこに書くことによって、もしその term がその型を持っていなかったらコンパイル時に型エラーになるので、プログラマーの意図通りになっているかどうかの確認に使える。また、大きな term の型はプログラムを読む人（あるいはプログラムを書いている人）が一目で分からない場合があり、そのとき、型が書いてあればプログラムを読むのに役に立つ。

ここでは、そのための構文として $\mathtt{t}$ `as` $\mathtt{T}$ と書く。これに関する型付け規則は、

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{T}}{\Gamma \vdash \mathtt{t}\ \mathtt{as}\ \mathtt{T} : \mathtt{T}} \text{ (T-ASCRIBE)}$$

を用いる。また、評価規則は

$$\frac{}{\mathtt{v\ as\ T} \to \mathtt{v}}\ \text{(E-ASCRIBE)}$$

$$\frac{\mathtt{t} \to \mathtt{t'}}{\mathtt{t\ as\ T} \to \mathtt{t'\ as\ T}}\ \text{(E-ASCRIBE1)}$$

を用いる。また、置換の定義に ascription の場合を追加する。

$$[\mathtt{x} \mapsto \mathtt{s}](\mathtt{t\ as\ T}) = ([\mathtt{x} \mapsto \mathtt{s}]\mathtt{t})\ \mathtt{as\ T}$$

練習問題 5. 型判定

$$\emptyset \vdash ((\lambda \mathtt{x} : \mathtt{Nat}.\mathtt{x})\ \mathtt{0})\ \mathtt{as\ Nat}\ :\ \mathtt{Nat}$$

は成り立つが、これの導出木を書け。

練習問題 6. term

$$((\lambda \mathtt{x} : \mathtt{Nat}.\mathtt{x})\ \mathtt{0})\ \mathtt{as\ Nat}$$

を評価せよ。各 1 ステップ評価の導出木も書け。

8 let 式

大きな式を書くときに、その部分式に名前を付けることができると便利がよい。ほとんどの言語はこのような機構を備えている。ML 系言語では、$\mathtt{let\ x = t_1\ in\ t_2}$ を評価するときは、まず式 $\mathtt{t_1}$ を評価して、それに x という名前を付けてから、$\mathtt{t_2}$ を評価する。

let 式の評価規則は、

$$\frac{}{\mathtt{let\ x = v_1\ in\ t_2} \to [\mathtt{x} \mapsto \mathtt{v_1}]\mathtt{t_2}}\ \text{(E-LETV)}$$

$$\frac{\mathtt{t_1} \to \mathtt{t_1'}}{\mathtt{let\ x = t_1\ in\ t_2} \to \mathtt{let\ x = t_1'\ in\ t_2}}\ \text{(E-LET)}$$

とする。また、let 式に関する型付け規則は

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1} \quad \Gamma, \mathtt{x} : \mathtt{T_1} \vdash \mathtt{t_2} : \mathtt{T_2}}{\Gamma \vdash \mathtt{let\ x = t_1\ in\ t_2} : \mathtt{T_2}}\ \text{(T-LET)}$$

とする。また、置換の定義に let 式の場合を追加する。

$$[\mathtt{x} \mapsto \mathtt{s}](\mathtt{let\ x = t_1\ in\ t_2}) = \mathtt{let\ x} = ([\mathtt{x} \mapsto \mathtt{s}]\ \mathtt{t_1})\ \mathtt{in}\ ([\mathtt{x} \mapsto \mathtt{s}]\ \mathtt{t_2})$$

練習問題 7. let 式

$$\mathtt{let\ x = true\ in\ if\ x\ then\ 0\ else\ succ\ 0}$$

は空の型環境で型を持つが、その型判定の導出木を書け。

練習問題 8. let 式

$$\mathtt{let\ x = true\ in\ if\ x\ then\ 0\ else\ succ\ 0}$$

を評価せよ。各 1 ステップ評価の導出木も書け。

let 式は derived form として定義することもできる。let 式 $\texttt{let x} = \texttt{t}_1 \texttt{ in t}_2$ は $(\lambda \texttt{x} : \texttt{T}_1.\texttt{t}_2)\ \texttt{t}_1$ の略記として定義すればよい。ただ、ラムダ抽象に $\texttt{T}_1$ という型注釈があり、これは let 式にはない。つまり、desugaring 時に型 $\texttt{T}_1$ を何らかの手段で得なければならない。これは型検査器 (type checker) から情報を得ればよく、$\texttt{t}_1$ の型を $\texttt{T}_1$ にすればよい。ただし、let 式のこの derived form による定義は逐次式などの場合とは違い、単なる term の変換ではなく、型判定の導出木の変換である。具体的には、

$$\frac{\vdots \qquad \vdots}{\dfrac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \quad \Gamma, \texttt{x} : \texttt{T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \texttt{let x} = \texttt{t}_1 \texttt{ in t}_2 : \texttt{T}_2}\ \text{(T-LET)}}$$

という形の導出木から

$$\frac{\dfrac{\dfrac{\vdots}{\Gamma, \texttt{x} : \texttt{T}_1 \vdash \texttt{t}_2 : \texttt{T}_2}}{\Gamma \vdash \lambda \texttt{x} : \texttt{T}_1.\texttt{t}_2 : \texttt{T}_1 \to \texttt{T}_2}\ \text{(T-ABS)} \quad \dfrac{\vdots}{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1}}{\Gamma \vdash (\lambda \texttt{x} : \texttt{T}_1.\texttt{t}_2)\ \texttt{t}_1 : \texttt{T}_2}\ \text{(T-APP)}$$

という形の導出木への変換である。

参考 let 多相のある言語では let 式は上記のようにラムダ抽象と関数適用により derived form として定義することはできない。

9 対 (pair)

多くの言語では複合データ型を構築する機構を備える。最も簡単なものでは対 (pair)、より一般には tuple がある。対を追加するために、まず、構文定義に対し

$$\texttt{t} ::= \ldots \mid \{\texttt{t}, \texttt{t}\} \mid \texttt{t.1} \mid \texttt{t.2}$$

のように 3 つの構文を追加する。対を中括弧を使って書くのは、集合の表記と紛らわしく、ML 系言語では普通の括弧を使って書く。ML 系言語ではレコード（後述）は中括弧を使って書くので、参考書ではレコードとの関連を強調するために対を中括弧を使って書いている。

また、値については、

$$\texttt{v} ::= \ldots \mid \{\texttt{v}, \texttt{v}\}$$

のように、値の対を追加する。また、型の定義に product type を追加する。

$$\texttt{T} ::= \ldots \mid \texttt{T}_1 \times \texttt{T}_2$$

また、評価規則は

$$\frac{}{\{\texttt{v}_1, \texttt{v}_2\}.\texttt{1} \to \texttt{v}_1}\ \text{(E-PAIRBETA1)} \quad \frac{}{\{\texttt{v}_1, \texttt{v}_2\}.\texttt{2} \to \texttt{v}_2}\ \text{(E-PAIRBETA2)}$$

$$\frac{\texttt{t}_1 \to \texttt{t}_1'}{\texttt{t}_1.\texttt{1} \to \texttt{t}_1'.\texttt{1}}\ \text{(E-PROJ1)} \quad \frac{\texttt{t}_1 \to \texttt{t}_1'}{\texttt{t}_1.\texttt{2} \to \texttt{t}_1'.\texttt{2}}\ \text{(E-PROJ2)}$$

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\{\mathtt{t}_1, \mathtt{t}_2\} \to \{\mathtt{t}_1', \mathtt{t}_2\}}\ \text{(E-PAIR1)} \quad \frac{\mathtt{t}_2 \to \mathtt{t}_2'}{\{\mathtt{v}_1, \mathtt{t}_2\} \to \{\mathtt{v}_1, \mathtt{t}_2'\}}\ \text{(E-PAIR2)}$$

を用いる。また、型付け規則は、

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \{\mathtt{t}_1, \mathtt{t}_2\} : \mathtt{T}_1 \times \mathtt{T}_2}\ \text{(T-PAIR)}$$

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t.1} : \mathtt{T}_1}\ \text{(T-PROJ1)} \quad \frac{\Gamma \vdash \mathtt{t} : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t.2} : \mathtt{T}_2}\ \text{(T-PROJ2)}$$

を用いる。また、置換の定義に対の場合を追加する。

$$[\mathtt{x} \mapsto \mathtt{s}]\{\mathtt{t}_1, \mathtt{t}_2\} = \{[\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_1, [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}_2\}$$

(E-PAIR1)、(E-PAIR2) 規則により、対の要素は左が先に評価されることになる。例えば、term

```
{pred 4, if true then false else false}.1
```

は以下のように評価される。

$$\begin{aligned} & \{\mathtt{pred}\ 4,\ \mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{false}\ \mathtt{else}\ \mathtt{false}\}.1 \\ \to\ & \{3,\ \mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{false}\ \mathtt{else}\ \mathtt{false}\}.1 \\ \to\ & \{3,\ \mathtt{false}\}.1 \\ \to\ & 3 \end{aligned}$$

ただし、4 は succ(succ(succ(succ 0))) の略記であり、3 は succ(succ(succ 0)) の略記である。上記の各 1 ステップ評価の導出木は黒板で説明する。

練習問題 9. term

$$(\lambda \mathtt{x} : \mathtt{Nat} \times \mathtt{Nat}.\ \mathtt{x.2})\ \{\mathtt{pred}\ 4,\ \mathtt{pred}\ 5\}$$

は空の型環境で型を持つ。その型判定とその導出木を書け。

練習問題 10. term

$$(\lambda \mathtt{x} : \mathtt{Nat} \times \mathtt{Nat}.\ \mathtt{x.2})\ \{\mathtt{pred}\ 4,\ \mathtt{pred}\ 5\}$$

を評価せよ。各 1 ステップ評価の導出木も書け。