ソフトウェア基礎論配布資料 (4)
λ計算 (1)

五十嵐 淳
京都大学 大学院情報学研究科知能情報学専攻
e-mail: igarashi@kuis.kyoto-u.ac.jp

平成17年11月8日

1 関数とλ記法，λ計算

プログラムにおいて，似たような式・計算手順が複数の箇所で必要になった時には，関数や手続きを定義して，式・手順の再利用を図ることが一般的である．数学でも

$$2^2\pi + 7^2\pi + 20^2\pi$$

と書く代わりに，

$$f(2) + f(7) + f(20) \quad \text{ただし } f(x) = x^2\pi$$

と書けば，式の見通しがよくなる．ここで x はパラメータと呼ばれ，使われる場所によって異なる部分を表す役割を担っている．

この f のように，関数の概念は，集合論で扱われる「入力と出力の対の集合」という扱いよりも,「入力から出力を計算する式」とする扱いの方がより直感的かもしれない．このような「計算可能な関数」を扱うための理論がλ**計算**(*λ-calculus*) であり，その中心となるのがλ記法と呼ばれる関数の記法である．

λ 記法は，λ⟨ パラメータ ⟩.⟨ 式 ⟩ という形で「パラメータを入力として式の計算結果を出力とする関数」を表現する．上の例の f は $\lambda x.x^2\pi$ と書くことができる．この λ 記法の特徴的な点は，

その関数自体に名前をつけずに関数を表現することができる

という点である．これにより，関数の概念そのものと関数に名前をつけるという行為を切離すことができる．

λ 記法による関数に，その引数を与えた時 (関数を適用する，という) 関数の値は「パラメータを引数で置き換えること」で得ることができる．すなわち，$f = \lambda x.x^2\pi$ とすると，

$$\begin{aligned} & f(2) + f(7) + f(20) \\ (\text{定義より}) \quad &= (\lambda x.x^2\pi)(2) + f(7) + f(20) \\ (x \text{ を } 2 \text{ で置き換え}) \quad &= 2^2\pi + f(7) + f(20) \\ &\vdots \\ &= 453\pi \end{aligned}$$

という推論 (計算) が可能になる．λ 計算の体系は，λ 記法による関数，関数適用によるパラメータ置換の仕組みを形式的に実現したものであり，特に，パラメータ置換こそが計算ステップである，という立場をとる．

1.1 なぜ λ 計算なのか？

- 高級プログラミング言語にとって，関数・手続きの仕組みは必須のものであり，λ 計算は関数呼び出しの仕組みのモデルである．
- 様々なプログラム機構が λ 計算の枠組で「表現できる」．
- チューリング機械より単純，かつプログラミング言語に近い計算モデル．

上の例では，π という実数や乗算を仮定していたが，純粋な λ 計算においては，通常のプログラミング言語に見られるような整数などの基本的なデータ (と，その上の演算) すら取り扱わない，関数と関数適用しかない言語になっている．純粋な体系上に，基本的なデータ型 (例えば自然数や真偽値) を付加することは容易であるので，後で扱うことにする．

2 λ 項の定義

λ 計算では，プログラムに対応する対象を λ **項**(λ *term*) と呼ぶ．λ 項のためのメタ変数として t を使用する．上で見たように λ 記法には関数パラメータである変数が現れるので，ここでは前章で扱った変数宣言・定義の仕組みを最初から取り入れるのが自然である．そこで，$\Gamma \vdash t \in \mathbf{Term}$ という判断を考える．

変数・定義　変数・定義の取り扱いは算術式の時と全く同様である．図 1 に規則を示す．

$$\frac{}{\bullet \in \mathbf{Env}} \quad (\textsc{Env-Empty}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term}}{\Gamma, x \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var2})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \in \mathbf{Env}} \quad (\textsc{Env-Decl}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var3})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \in \mathbf{Env}} \quad (\textsc{Env-Defn}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad (x \neq y)}{\Gamma, y \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var4})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var0})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var1}) \qquad \frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term} \qquad (x \neq y)}{\Gamma, y = t \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var5})$$

図 1: λ 項の定義 (抜粋)

関数適用　λ 計算では伝統的に関数 t_1 の引数 t_2 への適用を $t_1\ t_2$ と引数 t_2 に括弧をつけずに記述する．($\sin \pi$ などの記法と同じ!) ただし，算術式の時と同様，結合を示すためのメタな括弧 () が使われることはある．

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-App})$$

この形の項を**関数適用**(*function application*) と呼ぶ．

λ 抽象　λ 記法による関数は λ 計算では **λ 抽象**(*λ-abstraction*) と呼ぶ．$\lambda x.t$ において，$\lambda x.$ は t を有効範囲とする変数 x の局所的な宣言であると考えられる．例えば $\bullet \vdash \lambda x.x \in \mathbf{Term}$ はもっともらしい判断であるが，$\bullet \vdash x(\lambda x.x) \in \mathbf{Term}$ は，最初の変数参照 x は $\lambda x.$ の宣言の有効範囲にもなく，環境も空であるので，導出されない判断である．一方，$x \vdash x(\lambda x.x) \in \mathbf{Term}$ や $x \vdash x(\lambda x.\#^1 x)$ はもっともらしい判断である．

λ 抽象のための規則は以下のように与えられる．

$$\frac{\Gamma, x \vdash t \in \mathbf{Term}}{\Gamma \vdash \lambda x.t \in \mathbf{Term}} \quad (\textsc{Tm-Abs})$$

つまり，Γ の下で $\lambda x.t$ が λ 項であるのは，Γ, x と局所変数宣言を加えた環境の下で t が λ 項であるときである．

2.1　メタ括弧の省略

λ 計算では，結合を示すための (メタな) 括弧を以下の要領で省略する．

- 関数適用同士は左結合する．つまり $t_1\ t_2\ t_3 \equiv (t_1\ t_2)\ t_3$ である．
- λ 抽象は可能な限り右に伸びる．つまり，$\lambda x.t_1\ t_2 \equiv \lambda x.(t_1\ t_2) \not\equiv (\lambda x.t_1)\ t_2$ である．

2.1.1 練習問題: 判断 $\bullet \vdash \lambda x.\lambda y.\lambda z.x\ y\ z \in \mathbf{Term}$ を導出せよ．

3 簡約関係

簡約関係の判断は算術式と同様に $\Gamma \vdash t \longrightarrow t'$ と記述される．まずは，定義された変数の簡約を考えるが，λ 項に対する shift 操作の定義は算術式のそれよりも，項の中に変数宣言が含まれている可能性があるために多少複雑になる．復習すると， $t \Uparrow x$ は，Γ の下での式 t を Γ, x という環境で見直したものである．

例えば $(x\ y) \Uparrow x \equiv \#^1 x\ y$ であったり，$(\lambda x.x\ y) \Uparrow y \equiv \lambda x.x\ \#^1 y$ であるが，$(\lambda x.x\ y) \Uparrow x \equiv \lambda x.x\ y$ となるべきである．また，$(\lambda x.\#^1 x\ y) \Uparrow x \equiv \lambda x.\#^2 x\ y$ となるべきである．

より一般的に $t \Uparrow_i x$ という形で shift 関数を定義する．

$$
\begin{array}{lcll}
\#^i x \Uparrow_j x & \equiv & \#^{i+1} x & \text{if } i \geq j \\
\#^i x \Uparrow_j y & \equiv & \#^i x & \text{if } i < j \text{ or } x \not\equiv y \\
(\lambda x.t_0) \Uparrow_j x & \equiv & \lambda x.(t_0 \Uparrow_{j+1} x) & \\
(\lambda x.t_0) \Uparrow_j y & \equiv & \lambda x.(t_0 \Uparrow_j y) & \text{if } x \not\equiv y \\
(t_1\ t_2) \Uparrow_j x & \equiv & (t_1 \Uparrow_j x)\ (t_2 \Uparrow_j x) &
\end{array}
$$

以降，$t \Uparrow x$ は $t \Uparrow_0 x$ の略記とする．

定義された変数に関する簡約規則は $t \Uparrow x$ の定義の変更以外は算術式の時と同じである．

$$\frac{\Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash x \longrightarrow t \Uparrow x} \quad \text{(E-Def)}$$

$$\frac{\Gamma \vdash \#^i x \longrightarrow t}{\Gamma, y \vdash \#^i x \Uparrow y \longrightarrow t \Uparrow y} \quad \text{(E-Shift1)}$$

$$\frac{\Gamma \vdash \#^i x \longrightarrow t \qquad \Gamma \vdash t' \in \mathbf{Term}}{\Gamma, y = t' \vdash \#^i x \Uparrow y \longrightarrow t \Uparrow y} \quad \text{(E-Shift2)}$$

3.1 β 簡約

最初に述べたように，関数を具体的な引数に適用した値は,「パラメータを引数で置き換えること」で得ることができる．つまり，$(\lambda x.t_0)\ t_1$ の計算結果は，t_0 中の x を t_1 に「置き換え」たような項 t' の計算結果である．このパラメータ置換を計算過程として表現したのが以下の簡約関係

$$\Gamma \vdash (\lambda x.t_0)\ t_1 \longrightarrow t'$$

$$\frac{\Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^0 x[\#^0 x] \Rightarrow t \Uparrow x} \quad \text{(S-Def1)}$$

$$\frac{\Gamma \vdash t'' \in \mathbf{Term} \qquad \Gamma \vdash \#^i x[\#^i x] \Rightarrow t'}{\Gamma, z = t'' \vdash (\#^i x \Uparrow z)[\#^i x \Uparrow z] \Rightarrow t' \Uparrow z} \quad \text{(S-Shift2)}$$

$$\frac{\Gamma \vdash t \in \mathbf{Term} \qquad (x \not\equiv y \text{ or } i \neq j)}{\Gamma, x = t \vdash \#^j y[\#^i x] \Rightarrow \#^j y} \quad \text{(S-Def2)}$$

$$\frac{\Gamma \vdash t_1[\#^i x] \Rightarrow t_1' \qquad \Gamma \vdash t_2[\#^i x] \Rightarrow t_2'}{\Gamma \vdash t_1\ t_2[\#^i x] \Rightarrow t_1'\ t_2'} \quad \text{(S-App)}$$

$$\frac{\Gamma \vdash \#^i x[\#^i x] \Rightarrow t'}{\Gamma, z \vdash (\#^i x \Uparrow z)[\#^i x \Uparrow z] \Rightarrow t' \Uparrow z} \quad \text{(S-Shift1)}$$

$$\frac{\Gamma, y \vdash t_0[\#^i x \Uparrow y] \Rightarrow t_0'}{\Gamma \vdash \lambda y.t_0[\#^i x] \Rightarrow \lambda y.t_0'} \quad \text{(S-Abs)}$$

図 2: 定義参照の一括解決

である．この簡約過程を β **簡約**(β-*reduction*)，置き換え操作を**代入**(*substitution*) と呼び，この左辺の形の項を β**(簡約) 基**(β-*redex*) と呼ぶ．以下ではこの β 簡約の定義について詳しく見ていく．

今，β 基の環境を Γ とする．つまり，$\Gamma \vdash (\lambda x.t_0)\ t_1 \in \mathbf{Term}$ であるとする． このとき，$\Gamma, x \vdash t_0 \in \mathbf{Term}$ であることに注意すると， 関数呼び出しとは，パラメータである x を実際の引数である t_1 と定義して t_0 を計算，つまり，$\Gamma, x = t_1$ という 環境の下で t_0 を計算していくことに相当する． すると，代入操作は，t_0 中の全ての x への定義参照をまとめて解決することと考えられる．(これまでの定義の形式化に沿うならば，計算 1 ステップでは x の参照ひとつしか t_1 で置き換えられないし，必ずしも置き換えを最初にやる必要はないのだが．)

この「まとめて定義参照の解決をする」ことを規則で定義する.「Γ の下での項 t 中の全ての定義参照 $\#^i x$ を解決した項は t' である」という判断を $\Gamma \vdash t[\#^i x] \Rightarrow t'$ と書く．この判断の導出規則は図 2 で与えることができる．

このようにして，まとめて定義参照の解決を定義することができるが，得られた項は最早，定義 $x = t_1$ を参照していないので，$\Gamma, x = t_1$ という環境のもとでの項 t を Γ の下での表現に直すことができる．これは，逆 shift 関数とも呼べるもので，$t \Downarrow_i x$ と表記し，以下のように定義する．

$$\begin{array}{lcll}
\#^i x \Downarrow_j x & \equiv & \#^{i-1} x & \text{if } i \geq j \text{ and } i > 0 \\
\#^i x \Downarrow_j y & \equiv & \#^i x & \text{if } i < j \text{ or } x \not\equiv y \\
\lambda x.t_0 \Downarrow_j x & \equiv & \lambda x.(t_0 \Downarrow_{j+1} x) & \\
\lambda x.t_0 \Downarrow_j y & \equiv & \lambda x.(t_0 \Downarrow_j y) & \text{if } x \not\equiv y \\
(t_1\ t_2) \Downarrow_j x & \equiv & (t_1 \Downarrow_j x)\ (t_2 \Downarrow_j x) &
\end{array}$$

$t \Downarrow_0 x$ のことを $t \Downarrow x$ と略記する．

以上より，最終的に，β 簡約は，

$$\frac{\Gamma, x = t_2 \vdash t_1[\#^0 x] \Rightarrow t'}{\Gamma \vdash (\lambda x.t_1)\ t_2 \longrightarrow t' \Downarrow x} \quad \text{(E-Beta)}$$

と定義できる．

3.1.1 定義 [簡約関係]: 判断 $\Gamma \vdash t \longrightarrow t'$ は E-Def, E-Shift1, E-Shift2, E-Beta に加えて以下の規則により導出される.

$$\frac{\Gamma, x \vdash t \longrightarrow t'}{\Gamma \vdash \lambda x.t \longrightarrow \lambda x.t'} \quad \text{(E-Lam)}$$

$$\frac{\Gamma \vdash t_1 \longrightarrow t_1'}{\Gamma \vdash t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{\Gamma \vdash t_2 \longrightarrow t_2'}{\Gamma \vdash t_1\ t_2 \longrightarrow t_1\ t_2'} \quad \text{(E-App2)}$$

□

3.1.2 練習問題: 以下の判断を導出せよ:

- $\mathtt{v}, \mathtt{w} \vdash (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x})\ \mathtt{v}\ \mathtt{w} \longrightarrow (\lambda \mathtt{y}.\mathtt{v})\ \mathtt{w}$
- $\mathtt{v}, \mathtt{w} \vdash (\lambda \mathtt{y}.\mathtt{v})\ \mathtt{w} \longrightarrow \mathtt{v}$
- $\mathtt{x}, \mathtt{y}, \mathtt{f} = \lambda \mathtt{z}.\mathtt{x} \vdash (\lambda \mathtt{x}.\mathtt{f})\ \mathtt{y} \longrightarrow (\lambda \mathtt{x}.\lambda \mathtt{z}.\#^1\mathtt{x})\ \mathtt{y}$
- $\mathtt{x}, \mathtt{y}, \mathtt{f} = \lambda \mathtt{y}.\mathtt{x} \vdash (\lambda \mathtt{x}.\lambda \mathtt{z}.\#^1\mathtt{x})\ \mathtt{y} \longrightarrow \lambda \mathtt{z}.\mathtt{x}$

λ 計算における簡約も，合流性を満たす．以下では，$\Gamma \vdash t_1 \longrightarrow t_2$, $\Gamma \vdash t_2 \longrightarrow t_3$, $\ldots \Gamma \vdash t_{n-1} \longrightarrow t_n$ であることを $\Gamma \vdash t_1 \longrightarrow^* t_n$ と記述する.

3.1.3 定理 [合流性, confluence]:] 任意の $\Gamma \in \mathbf{Env}$, $\Gamma \vdash t_1, t_2, t_3 \in \mathbf{Term}$ に対し，$\Gamma \vdash t_1 \longrightarrow^* t_2$，$\Gamma \vdash t_1 \longrightarrow^* t_3$ ならば，$\Gamma \vdash t_4 \in \mathbf{Term}$ なる t_4 が存在し，$\Gamma \vdash t_2 \longrightarrow^* t_4$ かつ $\Gamma \vdash t_3 \longrightarrow^* t_4$ である.

4 プログラミング言語: λ 計算

純粋な λ 計算をプログラミング言語のモデルとして捉えるには,

- 関数呼び出しのモデルといいつつ，引数がふたつ以上ある関数すらない.
- 変数は全て関数パラメータで局所変数の宣言すらできない.
- 再帰的関数が考えられていない.
- 基本的なデータ型すら入っていない.

など一見すると余りにも貧弱にみえる．しかし，これらは実は全て λ 計算の中で表現できることである．以下では，様々なプログラミング要素が純粋な λ 計算の枠組みで表現できることを見る.

4.1 複数の引数をもつ関数と Curry 化

x と y をパラメータとして t を計算する関数
$= x$ をパラメータとして,「y をパラメータとして t を計算する関数」を返す関数

4.2 Church Boolean

$$
\begin{aligned}
\texttt{true} &= \lambda\texttt{x}.\lambda\texttt{y}.\texttt{x} \\
\texttt{false} &= \lambda\texttt{x}.\lambda\texttt{y}.\texttt{y} \\
\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 &\stackrel{\text{def}}{=} t_1\, t_2\, t_3
\end{aligned}
$$

4.3 Pairing

$$
\begin{aligned}
\langle t_1, t_2 \rangle &\stackrel{\text{def}}{=} \lambda\texttt{x}.\texttt{x}\ t_1\ t_2 \\
\texttt{fst} &= \lambda\texttt{p}.\texttt{p}\ (\lambda\texttt{f}.\lambda\texttt{s}.\texttt{f}) \\
\texttt{snd} &= \lambda\texttt{p}.\texttt{p}\ (\lambda\texttt{f}.\lambda\texttt{s}.\texttt{s})
\end{aligned}
$$

4.4 Church 数

$$
\begin{aligned}
\texttt{0} &\stackrel{\text{def}}{=} \lambda\texttt{s}.\lambda\texttt{z}.\texttt{z} \\
n &\stackrel{\text{def}}{=} \lambda\texttt{s}.\lambda\texttt{z}.\underbrace{\texttt{s}\ (\texttt{s}\ (\cdots\ \texttt{s}}_{n}(\texttt{z})\cdots)) \\
\texttt{succ} &= \lambda\texttt{n}.\lambda\texttt{s}.\lambda\texttt{z}.\texttt{s}\ (\texttt{n}\ \texttt{s}\ \texttt{z}) \\
\texttt{plus} &= \lambda\texttt{n}.\lambda\texttt{m}.\lambda\texttt{s}.\lambda\texttt{z}.\texttt{n}\ \texttt{s}\ (\texttt{m}\ \texttt{s}\ \texttt{z}) \\
\texttt{times} &= \lambda\texttt{n}.\lambda\texttt{m}.\lambda\texttt{s}.\lambda\texttt{z}.\texttt{n}\ (\texttt{m}\ \texttt{s})\ \texttt{z} \\
\texttt{pred} &= \lambda\texttt{n}.\texttt{fst}\ (\texttt{n}\ (\lambda\texttt{p}.\langle\texttt{snd p}, \texttt{succ}\ (\texttt{snd p})\rangle)\ \langle\texttt{0}, \texttt{0}\rangle)
\end{aligned}
$$

4.5 再帰的関数

$$\begin{aligned} \texttt{omega} &= (\lambda\texttt{x.x x})\,(\lambda\texttt{x.x x}) \\ \texttt{fix} &= \lambda\texttt{f.}((\lambda\texttt{x.f (x x)})\,(\lambda\texttt{x.f (x x)})) \end{aligned}$$

$f(x) = e$ (式 e には f が出現) なる再帰的関数は $\texttt{fix}\ (\lambda\texttt{f.}\lambda\texttt{x.e})$ と表現される.

5 λ計算における lazy / eager 戦略

算術式のところで考えた，eager / lazy な戦略を，λ 計算の下で考え直すことにする．算術式の言語では +, * が基本的な演算であるが，λ 計算での基本的な演算は関数適用である．eager / lazy の区別は，演算を行うにあたってそれに関わる式 (a_1+a_2 の a_1 と a_2) をどこまで評価するか，ということに由来する．

- 算術式の eager な戦略が，a_1, a_2 を数値にまで計算してから演算を行なうのと同様，λ 計算での eager な戦略は $t_1\ t_2$ の t_1, t_2 ともに値にまで計算をすすめてから関数適用を行なうような戦略である．これを，別名**値呼び出し**(*call-by-value*) 戦略と呼ぶ.
- 算術式の lazy な戦略が，a_2 のみを演算をすすめるために最低限必要な状態 (Z か S(a) の形) に計算をして関数適用を行うのと同様，λ 計算での lazy な戦略は $t_1\ t_2$ の t_1 のみを関数適用ができる $\lambda x.t_0$ の形まで計算して，即，関数適用を行なうような戦略である．これを，歴史的な理由から **名前呼び出し**(*call-by-name*) 戦略と呼ぶ.

5.1 call-by-value 戦略

5.1.1 定義 [**call-by-value reduction**]: 環境 Γ のもとで項 t が値であるという判断を $\Gamma \vdash t \in \mathbf{VTerm}$, 環境 Γ の下で t が t' に call-by-value 戦略で 簡約される，という判断を $\Gamma \vdash t \longrightarrow_e t'$ と書き，E-Def, E-Shift1, E-Shift2 と以下の規則で定義される.

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \vdash \#^0 x \in \mathbf{VTerm}} \quad \text{(V-Var)}$$

$$\frac{\Gamma \vdash \lambda x.t \in \mathbf{Term}}{\Gamma \vdash \lambda x.t \in \mathbf{VTerm}} \quad \text{(V-Abs)}$$

$$\frac{\Gamma \vdash t \in \mathbf{VTerm}}{\Gamma, x \vdash t \Uparrow x \in \mathbf{VTerm}} \quad \text{(V-Shift)}$$

$$\frac{\Gamma \vdash t_2 \in \mathbf{VTerm} \qquad \Gamma, x = t_2 \vdash t_1[\#^0 x] \Rightarrow t'}{\Gamma \vdash (\lambda x.t_1)\ t_2 \longrightarrow_e t' \Downarrow x} \quad \text{(Ev-Beta)}$$

$$\frac{\Gamma \vdash t_1 \longrightarrow_e t_1'}{\Gamma \vdash t_1\ t_2 \longrightarrow_e t_1'\ t_2} \quad \text{(Ev-App1)}$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{VTerm} \qquad \Gamma \vdash t_2 \longrightarrow_e t_2'}{\Gamma \vdash t_1\ t_2 \longrightarrow_e t_1\ t_2'} \quad \text{(Ev-App2)}$$

□

5.2 call-by-name 戦略

5.2.2 定義 [**call-by-name reduction**]: 環境 Γ の下で t が t' に call-by-name 戦略で簡約される，という判断を $\Gamma \vdash t \longrightarrow_l t'$ と書き，E-Def, E-Shift1, E-Shift2, E-Beta と E-App1 の規則で定義される．□

6 整数 + 真偽値をもつ値呼び λ 計算

6.1 動機

- Church 数表現は call-by-value, call-by-name reduction では，自然数との対応関係が明らかでなくなる．(`plus 1 1` $\longrightarrow_e$ λ`s.`λ`z.1 s (1 s z)` であり 2 には簡約されない．ただし，振舞としては λ`s.`λ`z.1 s (1 s z)` も 2 の代わりに使える．)
- すこぶる読みにくい
- `plus true 1` みたいな式でも，簡約がすすむ．(その結果，意味のわからない λ 項が得られる．)

$\Longrightarrow$ 自然数・真偽値などの基本的なデータ型を言語のプリミティブとして導入する．(演習システムのゲーム untyped の対象でもある．)

6.2 定義

ここでは項の定義のみを示すのみとする．簡約などの定義は算術式や純粋な λ 計算のそれから，素直に導くことができる．項を導出する判断は純粋な λ 計算のそれを流用して $\Gamma \vdash t \in \mathbf{Term}$ という形式である．

追加された項のほとんどは説明不要であろうと思われる．`fix` は，再帰関数を定義するための機構で，`fix` $\lambda x.t$ という項で，t 中の x が再帰的に自分自身を指すことができる．具体的には，以下のような簡約規則となる．

$$\frac{\Gamma, x = \texttt{fix}\ \lambda x.t \vdash t[\#^0 x] \Rightarrow t'}{\Gamma \vdash \texttt{fix}\ \lambda x.t \longrightarrow t' \Downarrow x} \quad \text{(E-Fix)}$$

$$\frac{}{\bullet \in \mathbf{Env}} \quad (\textsc{Env-Empty})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \in \mathbf{Env}} \quad (\textsc{Env-Decl})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \in \mathbf{Env}} \quad (\textsc{Env-Defn})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma, x \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var0})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^0 x \in \mathbf{Term}} \quad (\textsc{Tm-Var1})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term}}{\Gamma, x \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var2})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term}}{\Gamma, x = t \vdash \#^{i+1} x \in \mathbf{Term}} \quad (\textsc{Tm-Var3})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad (x \not\equiv y)}{\Gamma, y \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var4})$$

$$\frac{\Gamma \vdash \#^i x \in \mathbf{Term} \qquad \Gamma \vdash t \in \mathbf{Term} \qquad (x \not\equiv y)}{\Gamma, y = t \vdash \#^i x \in \mathbf{Term}} \quad (\textsc{Tm-Var5})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-App})$$

$$\frac{\Gamma, x \vdash t \in \mathbf{Term}}{\Gamma \vdash \lambda x.t \in \mathbf{Term}} \quad (\textsc{Tm-Abs})$$

$$\frac{\Gamma \in \mathbf{Env} \qquad (n \text{ is a natural number})}{\Gamma \vdash n \in \mathbf{Term}} \quad (\textsc{Tm-Num})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ \texttt{-}\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-Minus})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma \vdash \texttt{true} \in \mathbf{Term}} \quad (\textsc{Tm-True})$$

$$\frac{\Gamma \in \mathbf{Env}}{\Gamma \vdash \texttt{false} \in \mathbf{Term}} \quad (\textsc{Tm-False})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term}}{\Gamma \vdash t_1\ \texttt{>}\ t_2 \in \mathbf{Term}} \quad (\textsc{Tm-Gt})$$

$$\frac{\Gamma \vdash t_1 \in \mathbf{Term} \qquad \Gamma \vdash t_2 \in \mathbf{Term} \qquad \Gamma \vdash t_3 \in \mathbf{Term}}{\Gamma \vdash \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \in \mathbf{Term}} \quad (\textsc{Tm-If})$$

$$\frac{\Gamma \vdash t \in \mathbf{Term}}{\Gamma \vdash \texttt{fix}\ t \in \mathbf{Term}} \quad (\textsc{Tm-Fix})$$

図 3: 拡張 λ 項の定義